GeophPy Documentation

Release 0.32

Lionel Darras, Philippe Marty & Quentin Vitale

Aug 01, 2019

TABLE OF CONTENTS

LIST OF FIGURES

ABOUT GEOPHPY

1.1 Introduction

`GeophPy` is a project initiated in 2014 through cooperation between two units of the CNRS[1] (UMR5133-Archeorient and UMR7619-Metis). Since 2017, it is also developed by Geo-Heritage (a cooperation between UMR5133-Archeorient and Eveha International).

`GeophPy` is an open source Python package that offers tools for sub-surface geophysical survey data processing, in the field of archaeology, geology, and other sub-surface applications.

It mainly focuses on ground surveys data and offers tools to process the data and create geophysical maps that can be imported to GIS softwares.

`GeophPy` contains *general tools*, such as data *destaggering*, *destriping* and *method-specific tools*, such as *reduction to the pole* or *magnetic data continuation*.

`GeophPy` builds a geophysical `DataSet` object composed by series of data in the format (X,Y,Z) with (X,Y) being the point position of the geophysical value Z, in order to process and/or display as maps of Z values.

This package can be used as command line, but a specific Graphical User Interface, WuMapPy, is designed for it.

1.2 Features

- Building dataset from one or severals data files.

- Displaying geophysical maps in 2-D or 3-D.

- Processing datasets with general or method-specific geophysical filters.

- Export processed datasets into georeferenced format.

- Compatibility with Python 3.x.

1.3 Main authors

- **Lionel DARRAS**

  CNRS, UMR5133-Archeorient, Lyon, France

  lionel.darras@mom.fr

- **Philippe MARTY**

---

[1] French National Center for Scientific Research

CNRS, UMR7619-Metis, Paris, France

philippe.marty@upmc.fr

- **Quentin VITALE**

  Eveha International, Lyon, France

  quentin.vitale@eveha.fr

1.4 License

`GeophPy` is developed on a GNU GPL v3 license.

Copyright (c) 2014-2019 by Lionel Darras, Philippe Marty, Quentin Vitale and contributors. See AUTHORS for more details.

Some rights reserved.

Redistribution and use in source and binary forms of the software as well as documentation, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE AND DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

INSTALLATION

A Python (3.x) installation is necessary to install the **GeophPy** package.

2.1 Using pip

You can install **GeophPy** directly from the PyPI repository using `pip`.

First, make sure you have an up-to-date version of `pip` using the command:

```
>>>  pip install --upgrade pip
>>>  or
>>>  python -m pip install --upgrade pip
```

Then, install, upgrade (or uninstall) **GeophPy** directly from PyPI repository using `pip` with these commands:

```
>>>  pip install geophpy
>>>  pip install --upgrade geophpy
>>>  pip uninstall  geophpy
```

You can also download the zip file "GeophPy-vx.y" from the PyPI repository, and from the download folder use:

```
>>>  pip install GeophPy-vx.y.zip
```

2.2 Building from sources

Download the zip file "GeophPy-vx.y" and unzip it. Go to the unzipped folder and run the install script with the following command:

```
>>>  python setup.py install
>>>  or
>>>  python -m setup.py install
```

2.3 Dependencies

GeophPy depends on the following Python packages:

Name	Version	Comment
numpy		
SciPy		
matplotlib		
netCDF4		
Pillow		
PySide		
pyshp		
simplekml		
utm		
Sphinx	1.4.3 (or greater)	

Tip: Failure on **Windows**

GeophPy uses others Python modules and packages that should be automatically installed. However, the installation of these modules may failed on **Windows** (in the absence of a C++ compiler for instance).

They can be installed independently using the useful website: http://www.lfd.uci.edu/~gohlke/pythonlibs/. This website provides many popular scientific Python packages precompiled for **Windows** distributions.

To install a package independently:

1. Download the precompiled package sources corresponding to your Python version and **Windows** distribution (SomePackage-vx.y-cp3x-cp3xm_winxx.whl);

   NumPy, a fundamental package needed for scientific computing with Python.
   Numpy+MKL is linked to the Intel® Math Kernel Library and includes required DLLs in the numpy.core directory.
   numpy-1.14.6+mkl-cp27-cp27m-win32.whl
   numpy-1.14.6+mkl-cp27-cp27m-win_amd64.whl
   numpy-1.14.6+mkl-cp34-cp34m-win32.whl
   numpy-1.14.6+mkl-cp34-cp34m-win_amd64.whl
   numpy-1.14.6+mkl-cp35-cp35m-win32.whl
   numpy-1.14.6+mkl-cp35-cp35m-win_amd64.whl
   numpy-1.14.6+mkl-cp36-cp36m-win32.whl

2. In download folder, use a command prompt and install the package using `pip` with the name of the downloaded archive:

   ```
   >>> python setup.py install SomePackage-vx.y-cp3x-cp3xm_winxx.whl
   >>> or
   >>> python -m setup.py install SomePackage-vx.y-cp3x-cp3xm_winxx.whl
   ```

3. Repeat the process for all packages which installation failed before re-installing **GeophPy**.

2.4 Uninstallation

The Python package can simply be uninstalled using `pip`:

```
>>> pip uninstall geophpy
>>> or
>>>  python -m pip uninstall geophpy
```

GETTING STARTED

3.1 Quick start

First, import the *DataSet* class from the *dataset* module using the `import` command at the top of your script:

```
>>> from geophpy.dataset import DataSet
```

You can a so import the complete dataset module content using (not recommended for script readability):

```
>>> from geophpy.dataset import *
```

Reading data from files

Read your data from an ASCII delimiter-separated values files using the *from_file()* method from the *DataSet* class:

```
>>> # Opening data from an ascii file
>>> dataset = DataSet.from_file(["test.dat"], fileformat='ascii',
x_colnum=1, y_colnum=2, z_colnum=5)
```

If not provided, the file format will be estimated from the file extension and the first three columns will be considered as the data (X, Y and Z respectively).

You can optionally give the file delimiter, the number of header lines to skip and the header lines that contains the different field names.

Displaying the dataset

Display your raw data as a scatter plot or interpolate it to display 2-D surface plot:

```
>>> # Raw data scatter plot
>>> fig, cmap = dataset.plot(plottype='2D-SCATTER', cmmin=-20, cmmax=20)
>>> fig.show()
```

```
>>> # Raw data surface plot
>>> dataset.interpolate(interpolation='linear')  # or 'none'
>>> fig, cmap = dataset.plot(plottype='2D-SURFACE', cmmin=-20, cmmax=20)
>>> fig.show()
```
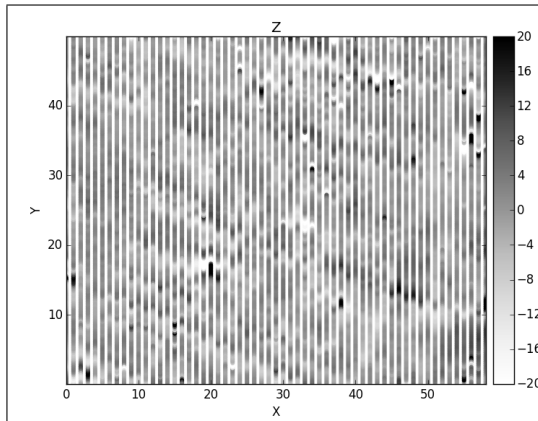
Fig. 3.1: Quick Start - Raw dataset scatter plot.
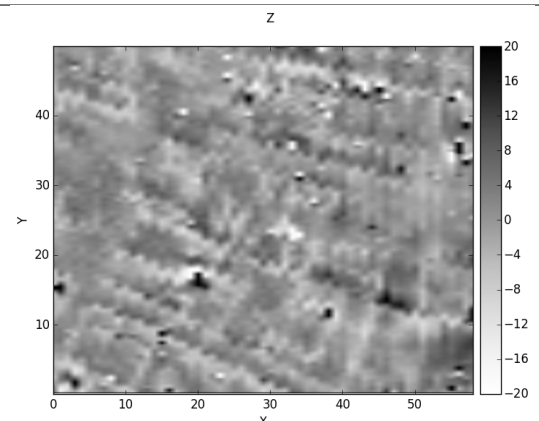
Fig. 3.2: Quick Start - Raw dataset surface plot.

You can also display the position of the measurement points in a plot or directly onto the 2-D surface plot:

```
>>> # Data point position (postmap plot)
>>> fig, cmap = dataset.plot(plottype='2D-POSTMAP')
>>> fig.show()
```

```
>>> # Data point position onto the 2-D surface
>>> fig, cmap = dataset.plot(plottype='2D-SURFACE', pointsdisplay=True)
>>> fig.show()
```
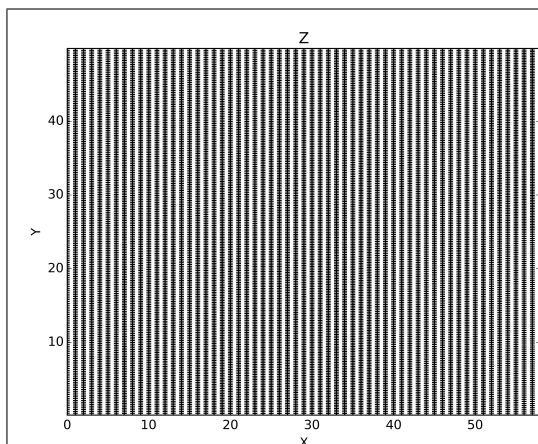

Fig. 3.3: Quick Start - Raw dataset postmap plot.
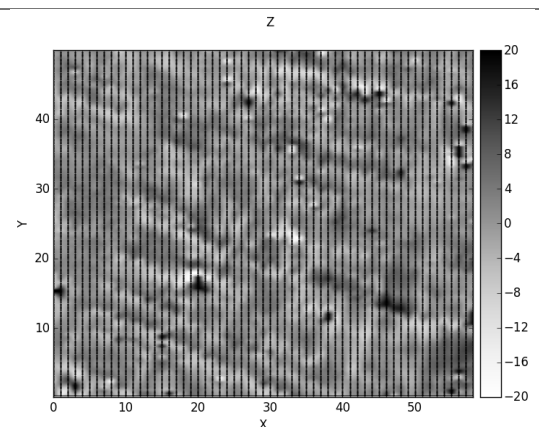
Fig. 3.4: Quick Start - Raw dataset surface plot with data points.

See *Plotting functions* for the complete list of available plot possibilities.

General processing

Use the available processing (despike, destaggering, destriping etc.) to filter your `DataSet`:

```
>>> # Dataset destaggering
>>> dataset.festoonfilt(corrmin=0.5, setmin=-20, setmax=20)
```

```
>>> # Dataset destriping
>>> dataset.destripecon(Nprof='all', setmin=-20, setmax=20)
```
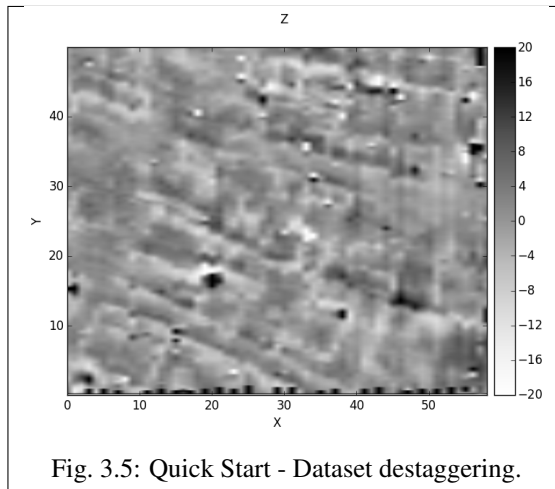

Fig. 3.5: Quick Start - Dataset destaggering.
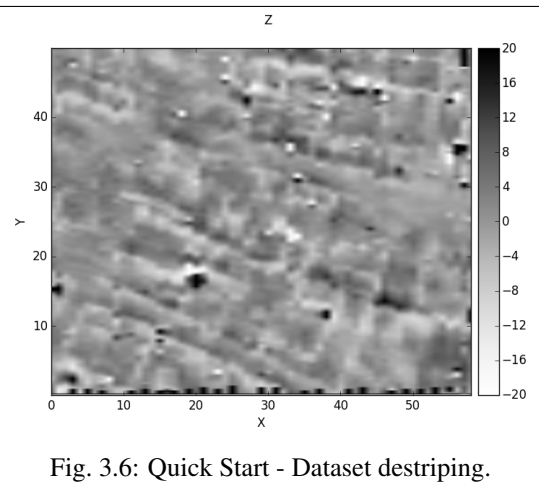
Fig. 3.6: Quick Start - Dataset destriping.

See *General Processing* for the complete list of available processing.

Method-specific processing

Use the method-specific available processing (for magnetic survey for instance):

```
>>> # Dataset reduction to the pole
>>> dataset.polereduction()
```

See *Magnetic Processing* for the complete list of available magnetic-survey-specific processing.

Saving data in a file

Save your data to a file using the `to_file()` method from the `DataSet` class:

```
>>> # Saving to a NetCDF file
>>> dataset.to_file("save.nc", description='My Processed Data')
```

```
>>> # Saving to a Surfer Grid file
>>> dataset.to_file("save.grd")
```

```
>>> # Saving to an ascii file
>>> dataset.to_file("save.dat", fileformat='ascii', delimiter=',')
```

> **Warning:** If you only processed the gridded `DataSet` (valfilt=False), use the `sample()` method to re-sample the gridded dataset at the original dataset value positions. Otherwise you will be saving the imported raw data value.

Georeferencing grid nodes

Georeference your data using Ground Control Points (survey nodes known in both local and geographic coordinate systems). Use the `setgeoref()` from the `DataSet` class

```
>>> # Reading GCPs file
>>> import geophy.geoposset as pset
>>> gpos = pset.GeoPosSet.from_file("GCPs.csv")
>>> gpos.plot(long_label=True)
```

```
>>> # Georeferencing ungridded values
>>> dataset.setgeoref('UTM', gpos.points_list, 'T', 32)
```

```
>>> # Georeferencing gridded values
>>> dataset.interpolate(x_step=0.5, y_step=0.25)
>>> dataset.setgeoref('UTM', gpos.points_list, 'T', 32)
```
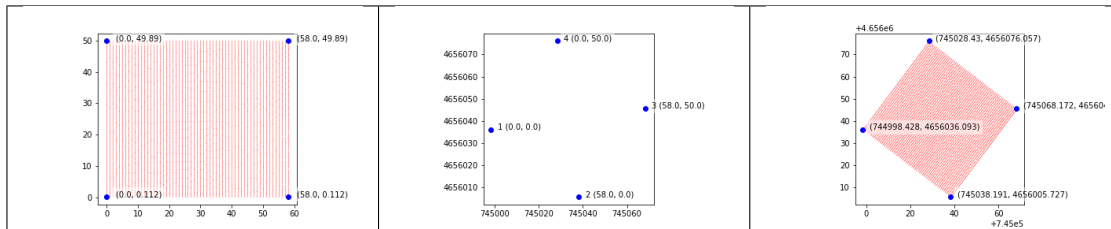

3.2 DataSet overview

All imported data are stored into a `DataSet` class that contains the different plotting and processing methods.

The `DataSet` class is composed of 3 objects:

DataSet.info

The `Info` class that contains the informations about the dataset:

- `x_min` = minimal x coordinate of the data set.
- `x_max` = maximal x coordinate of the data set.
- `y_min` = minimal y coordinate of the data set.
- `y_max` = maximal y coordinate of the data set.
- `z_min` = minimal z value of the data set.
- `z_max` = maximal z value of the data set.
- `x_gridding_delta` = delta between 2 x values in the interpolated image grid.
- `y_gridding_delta` = delta between 2 y values in the interpolated image grid.
- `gridding_interpolation` = interpolation name used for the building of the image grid.

DataSet.data

The `Data` class that contains:

- `fields` = the names of the fields (columns): ['X', 'Y', 'Z'].
- `values` = 2D array of raw values before interpolation (array with (x, y, z) values).
- `z_image` = 2D array of the current gridded data values.
- `easting_image` = None # easting grid (to use with z_image)

- *northing_image* = None # northing grid(to use with z_image)
- easting = None # easting array (to use with values)
- northing = None # northing array (to use with values)

Warning: The z_image object is NOT AUTOMATICALLY BUILT after opening a file but by explicitly using the gridding interpolation method *interpolate()*. See *Dataset operation* for details.

DataSet.georef

The *GeoRefSystem* class object contains:

- active = A flag for the georeferencing status.
- refsystem = The georeferencing system ('UTM', 'WGS84', . . .).
- utm_zoneletter = The optional UTM letter.
- utm_zonenumber = The optional UTM zone number.
- points_list = the list of the Ground Control Points coordinates in both the local and georeferenced system.

3.3 Opening files

All the reading possibilities are available through the *from_file()* method of the *DataSet* class.

GeophPy manages different types of files. You can obtain the list of accepted file formats with the command:

```
>>> from geophpy.dataset import fileformat_getlist
>>> fileformat_getlist()
['ascii', 'netcdf', 'surfer']
```

The file format is automatically recognized from the file extension using an internal dictionary:

```
>>> from geophpy.dataset import format_chooser
>>> format_chooser
{'.cdf': 'netcdf', '.nc': 'netcdf',
'.grd': 'surfer',
'.xyz': 'ascii', '.csv': 'ascii', '.txt': 'ascii', '.dat': 'ascii'}
```

If the file format is not in the dictionary or is not properly recognized from the extension, it can be forced to a specific format using the fileformat keyword of the *from_file()* method.

ASCII files

You can open comma-separated values (CSV) files, or any other delimiter-separated values files, by indicating the number and type of the columns of interest for the dataset to be processed:

```
>>> # Opening a ".dat" file from Geometrics Magnetometer G-858
>>> ## (format 'ascii' with delimiter ' ')
>>> dataset = DataSet.from_file(["test.dat"], fileformat='ascii',
delimiter=' ', z_colnum=5)
```

Geometrics Magnetometer G-858 *.dat* file example:

```
>>>          X           Y      TOP_RDG   BOTTOM_RDG    VRT_GRAD
>>>      50.000       0.059   46406.028   46390.698     -23.585
>>>      50.000       0.178   46407.275   46394.028     -20.380
>>>      50.000       0.296   46409.165   46397.987     -17.197
>>>         ...          ...        ...         ...          ...
```

```
>>> # Opening an *.xyz* file
>>> ## (column titles on the first line,
>>> ## X, Y and data values on the others lines,
>>> ## separated by a delimiter)
>>> dataset = DataSet.from_file(["test.xyz"], fileformat='ascii',
delimiter='\t', fields_row=1)
```

.xyz file examples:

```
>>> X       Y       Z
>>> 0       0       0.34
>>> 0       1       -0.21
>>> 0       2       2.45
>>> ...     ...     ...
```

```
>>> X,Y,Z
>>> 0,0,0.34
>>> 0,1,-0.21
>>> 0,2,2.45
>>> ...,...,...
```

You can also specify the number of header lines to skip or the specific columns for x, y and values:

```
>>> # Number of header lines to skip
>>> dataset = DataSet.from_file(["test.txt"], fileformat='ascii',
delimiter=',', skip_rows=4)
```

```
>>> # Specific x, y and value column numbers
>>> dataset = DataSet.from_file(["test.txt"], fileformat='ascii',
delimiter=',', x_colnum=xcol, y_colnum=ycol, z_colnum=zcol)
```

Note: If unspecified, the `delimiter` is estimated directly from the file content and the `fileformat` is determined from the file extension.

Surfer Grid files

GeophPy manages Golden Software **Surfer** Grid files (Surfer 7 binary grids, Surfer 6 binary grids and Surfer 6 ASCII grids). The grid type is automatically determined from the *.grd* file. To open a Surfer Grid simply use:

```
>>> # Opening a Surfer Grid file
>>> dataset = DataSet.from_file(["test.grd"])
```

NetCDF files

Previously processed dataset are by default save in NetCDF format (*.nc*). To open previously processed datasets, simply use:

```
>>> # Opening previously processed dataset (.nc)
>>> dataset = DataSet.from_file(["dataset1.nc"])
```

Concatenating Multiple files

It is possible to build a dataset from a concatenation of severals ASCII files of the same format:

```
>>> # Opening several selected files
>>> dataset = DataSet.from_file(["file1.dat","file2.dat"],
             format='ascii', delimiter=' ', z_colnum = 5)
```

```
>>> # Opening all files beginning by "file"
>>> dataset = DataSet.from_file(["file*.dat"], format='ascii',
             delimiter=' ', z_colnum = 5)
```

Note: When reading multiple files directly using the *from_file()* method, no edge-matching method are used so the original limits of the datasets in the mosaic maybe highly visible.

Checking files compatibility

Opening several files to build a data set needs to make sure that all files selected are in the same format.

It's possible to check it by reading the headers of each files:

```
>>> compatibility = True
>>> columns_nb = None
>>> for file in fileslist :
>>>     col_nb, rows = getlinesfrom_file(file)
>>>     if ((columns_nb != None) and (col_nb != columns_nb)) :
>>>         compatibility = False
>>>         break
>>>     else :
>>>         columns_nb = col_nb
```

3.4 Dataset operation

Besides actual *DataSet* processing, basic *DataSet* operations (geometrical transformation, math operations, interpolation etc.) are available through simple commands.

Duplicating dataset

Duplicate a *DataSet* before processing it to save the raw data:

```
>>> rawdataset = dataset.copy()
```

Dataset interpolation

Interpolate the data value with severals gridding interpolation methods ('none', 'nearest', 'linear', 'cubic') to build the dataset z_image object:

```
>>> dataset.interpolate(interpolation="none")
```
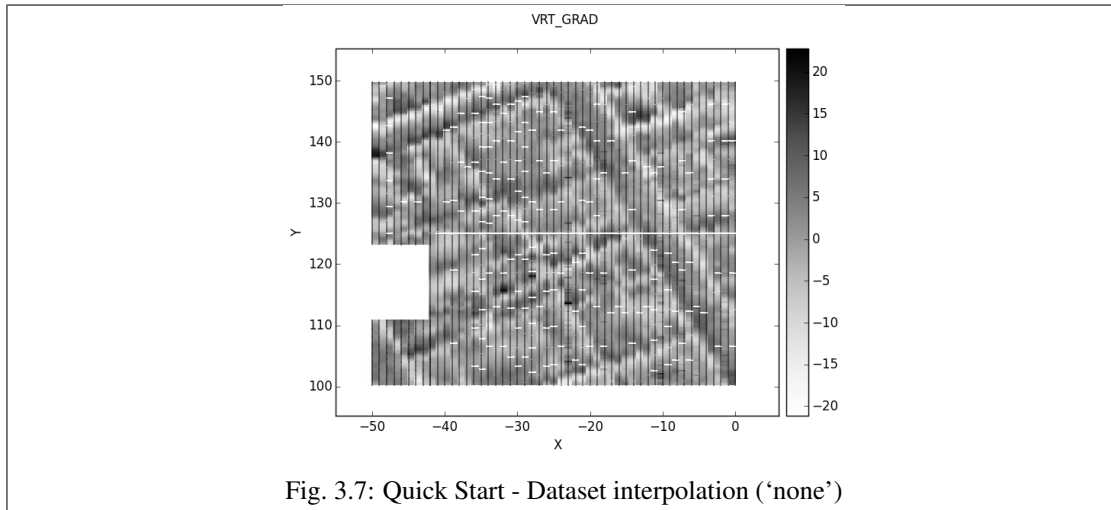
Fig. 3.7: Quick Start - Dataset interpolation ('none')

See *High level API* for calling details.

Retrieving grid coordinates

The `DataSet` grid (`z_image`) coordinate vectors and matrices can be retrieved with the following commands:

```
>>> # Grid coordinate matrices
>>> dataset.get_xgrid()   # x-coordinate matrix
>>> dataset.get_ygrid()   # x-coordinate matrix
>>> dataset.get_xygrid()   # both x and y-coordinate matrices
```

```
>>> # Grid coordinate vectors
>>> dataset.get_xvect()   # x-coordinate vector
>>> dataset.get_yvect()   # y-coordinate vector
>>> dataset.get_xyvect()   # both x and y-coordinate vectors
>>> dataset.get_gridextent()   # xmin, xmax, ymin, ymax
>>> dataset.get_gridcorners() # corners x and y-coordinates
```

The `DataSet` ungridded values (`values`) can be retrieved with the following commands:

```
>>> # Data sample coordinates
>>> dataset.get_xvalues()   # x-coordinates
>>> dataset.get_yvalues()   # y-coordinates
>>> dataset.get_yvalues()   # x, y-coordinates
>>> dataset.get_values()   # data values
>>> dataset.get_xyzvalues()   # both x, y-coordinates and data values
```

```
>>> # Bounding box
>>> dataset.get_boundingbox() # corners coordinates (equivalent get_
↪gridcorners() for a gridded dataset)
```

See *High level API* for calling details.

Basic math operations

You can add or multiply the `DataSet` values by a constant with the following commands:

```
>>> # Dataset addition/subtraction
>>> dataset.add(val=14, valfilt=True, zimfilt=True)
>>> dataset.add(val=-14, valfilt=True, zimfilt=True)
```

```
>>> # Dataset multiplication/division
>>> dataset.times(val=30, valfilt=True, zimfilt=True)
>>> dataset.add(val=1/30, valfilt=True, zimfilt=True)
```

See *High level API* for calling details.

Basic geometrical operations

You can translate or rotate the *DataSet* with the following commands:

```
>>> # Dataset translation
>>> dataset.translate(shiftx=20, shifty=-19)
```

```
>>> # Dataset rotation
>>> dataset.rotate(angle=90, center='BL')
```

See *High level API* for calling details.

3.5 Saving DataSet

You can save the *DataSet* in different a file formats using the *to_file()* method of the *DataSet* class.

For the time being, only three formats are available. The list of the available file formats can be obtained with the command:

```
>>> from geophpy.dataset import fileformat_getlist
>>> fileformat_getlist()
['ascii', 'netcdf', 'surfer']
```

When saving a file, the file format is automatically recognized from the file extension using an internal dictionary:

```
>>> from geophpy.dataset import format_chooser
>>> format_chooser
{'.cdf': 'netcdf', '.nc': 'netcdf',
'.grd': 'surfer',
'.xyz': 'ascii', '.csv': 'ascii', '.txt': 'ascii', '.dat': 'ascii'}
```

If the file format is not in the dictionary or is not properly recognized from the extension, it can be forced to a specific format using the `fileformat` keyword of the *from_file()* method.

NetCDF files

Saving your data in NetCDF file format allow the conservation of the gridded dataset, the dataset values and the georeferencing system. An optional description can be added using the `description` keyword.

```
>>> dataset.to_file('save.nc')
>>> dataset.to_file('save.nc', description='My dataset example')
>>> dataset.to_file('save.extension', fileformat='netcdf')
```

Surfer Grid files

Saving your data in Surfer Grid format only conserves the gridded dataset. The different available grid types can be obtained using the command:

```
>>> gridtype_getlist()
['surfer7bin', 'surfer6bin', 'surfer6ascii']
```

By default, the Surfer 7 binary grid type is used but you can use the `gridtype` keyword of the *from_file()* method to choose another grid type:

```
>>> dataset.to_file('save.grd')
>>> dataset.to_file('save.extension', fileformat='surfer')
>>> dataset.to_file('save.grd', gridtype='surfer6ascii')
```

Ascii files

Saving your data in Surfer Grid format only conserves the dataset (ungridded) values.

```
>>> dataset.to_file('save.csv')
>>> dataset.to_file('save.csv', delimiter='\t')
>>> dataset.to_file('save.extension', delimiter='\t', fileformat='ascii')
```

> **Warning:** If you only processed the gridded dataset (`valfilt=False`), use the *sample()* method to re-sample the gridded *DataSet* at the original value positions. Otherwise you will be saving the imported raw data value.

PLOTTING FUNCTIONS

The different plotting functions are based on Matplotlib and will return a Matplotlib figure object. The dataset can hence be plotted using commands of the form:

```
>>> fig = dataset.PlotFunction(option1=1, option2=True)
>>> fig.show()
```

Each plot types can be stored into a new or existing figure object with the `plot()` method of the `DataSet` class:

```
>>> # Plot in a new figure
>>> fig = dataset.PlotType(option1=1, ...)
>>> fig.show()
```

```
>>> # Plot in an existing figure
>>> fig = plt.figure()
>>> dataset.PlotType(fig=fig)
>>> fig.show()
```

and/or saved into a file providing a file name to the `plot()` method:

```
>>> # Saving plot in a file
>>> dataset.PlotType(filename='MyPlot.png')
```

The different available plotting functions are listed in the sections below.

4.1 Map plotting

All map displays are done through the `plot()` method of the `DataSet` class:

```
>>> fig, cmap = dataset.plot(plottype='2D-SURFACE', ...)
```

The `plot()` method returns both a `Figure` and the associated `Colorbar` so that the typical display sequence is:

```
>>> # Classic display command
>>> fig, cmap = dataset.plot(plottype='2D-SURFACE', ...)
>>> fig.show()
```

```
>>> # One-line command
>>> dataset.plot(plottype='2D-SURFACE', ...)[0].show()
```

4.1.1 Interpolation

To be displayed as a map, most of the plot types need a gridded dataset (i.e. *z_image*). Use the *interpolate()* method of the *DataSet* class to grid the dataset:

```
>>> # Linear interpolation
>>> dataset.interpolate(interpolation='linear')
```

```
>>> # No interpolation
>>> dataset.interpolate(interpolation='none')
```

To get the list of the available plotting interpolations use the command:

```
>>> interpolation_getlist()
['none', 'nearest', 'bilinear', 'bicubic', 'spline16', 'sinc']
```

see *Dataset operation*

4.1.2 Color maps

The list of the available color maps can be obtained with the command:

```
>>> colormap_getlist()
['Blues', 'BrBG', 'BuGn', 'BuPu', 'CMRmap', 'GnBu', 'Greens', 'Greys',
 'OrRd', 'Oranges', 'PRGn', 'PiYG', 'PuBu', 'PuOr', 'PuRd', 'Purples',
 'RdBu', 'RdGy', 'RdPu', 'RdYlBu', 'RdYlGn', 'Reds', 'Spectral', 'Wistia',
 'YlGn', 'YlGnBu', 'YlOrBr', 'YlOrRd', 'afmhot', 'autumn', 'binary',
 'bone', 'bwr', 'copper', 'gist_earth', 'gist_gray', 'gist_heat',
 'gist_yarg', 'gnuplot', 'gray', 'hot', 'hsv', 'jet', 'ocean', 'pink',
 'spectral', 'terrain']
```

You can plot a specific color map using the following command:

```
>>> # Plotting a color map
>>> fig = colormap_plot('BrBG')
>>> fig.show()
```

```
>>> # Plotting the reverse color map
>>> colormap_plot('BrBG', creversed=True).show()
```

Fig. 4.1: Plot - BrBG color map.	Fig. 4.2: Plot - BrBG reverse color map.
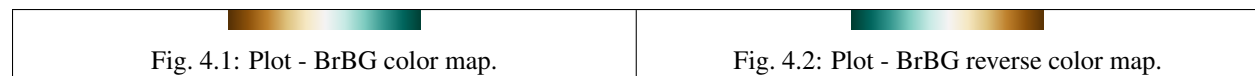
And you can save it to a file using:

```
>>> # Saving the color map as a .png file
>>> cmapfilename = 'colormap.png'
>>> colormap_plot(cmname, filename=cmapfilename)
```

You can also manually add "_r" (instead of using `creversed=True`) to the color map name to obtain the reversed color map:

```
>>> cmapnb = 6
>>> cmaplist = colormap_getlist()
>>> for i in range (cmapnb):
>>>     colormap_plot(cmaplist[i-1], filename="CMAP_" + str(i) + ".png")
```

Blues	BrBG	BuGn	BuPu	CMRmap	GnBu

or you can build figure and plot objects to display them in a new window:

```
>>> cm_fig = None
>>> first_time = True
>>> for cmapname in cmaplist:
>>>     cm_fig = colormap_plot(cmapname, fig=cm_fig)
>>>     if (first_time == True):
>>>         fig.show()
>>>         first_time = False
>>>     fig.draw()
```

4.1.3 Map types

It is possible to plot the dataset as a map using different plot types. To display the dataset as a map simply use the *plot()* method of the *DataSet* class:

```
>>> dataset.plot(plottype=PlotType, option1=..., option2=...)
```

The list of the a available plot types can be obtained by the command:

```
>>> from geophpy.dataset import plottype_getlist
>>> plottype_getlist()
['2D-SCATTER', '2D-SURFACE', '2D-CONTOUR', '2D-CONTOURF', '2D-POSTMAP']
```

You can overlays the measured data points to the map using the comand:

```
>>> # Data points display
>>> dataset.plot(plottype='2D-SURFACE', pointsdisplay=True)
```

2-D scatter and postmap plots

Display your raw data as a scatter plot to get a quick look at it or your data point position as a postmap:

```
>>> dataset.info.cmapname = 'gist_ncar'
>>> fig, cmap = dataset.plot(plottype='2D-SCATTER', cmmin=-20, cmmax=20)
>>> fig.show()
```

```
>>>  fig, cmap = dataset.plot(plottype='2D-POSTMAP')
>>> fig.show()
```
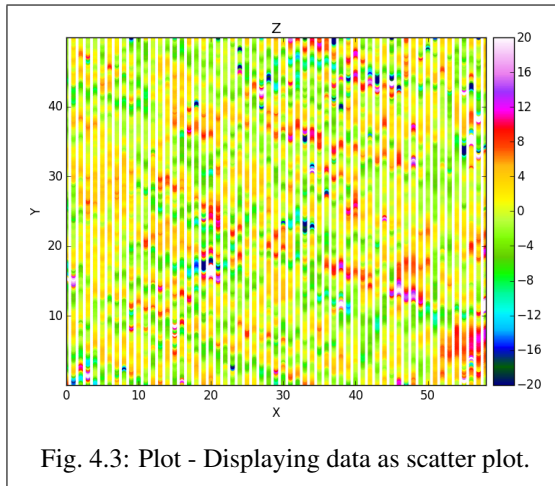
Fig. 4.3: Plot - Displaying data as scatter plot.
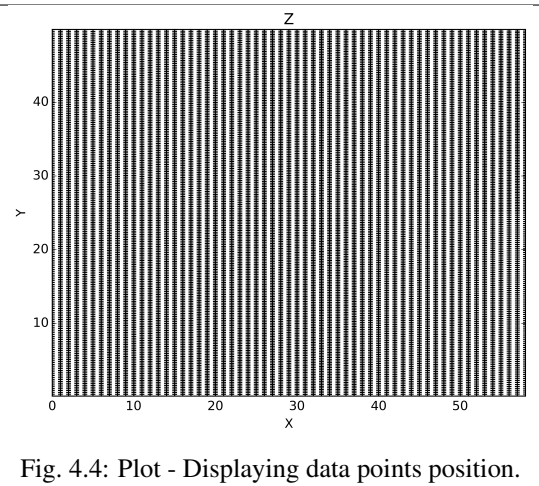
Fig. 4.4: Plot - Displaying data points position.

2-D surface plot

You can plot a dataset as a 2-D surface map using different interpolation for the display:

```
>>> # Dataset 2-D surface plot
>>> dataset.plot('2D-SURFACE', 'gray_r', plot.png,
interpolation='bilinear', transparent=True, dpi=400)
```

```
>>> dataset.plot('2D-SURFACE', 'gray_r', plot.png,
interpolation='bicubic', transparent=True, dpi=400)
```
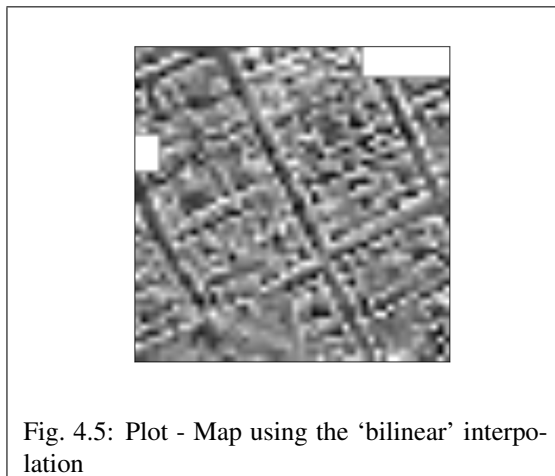

Fig. 4.5: Plot - Map using the 'bilinear' interpolation
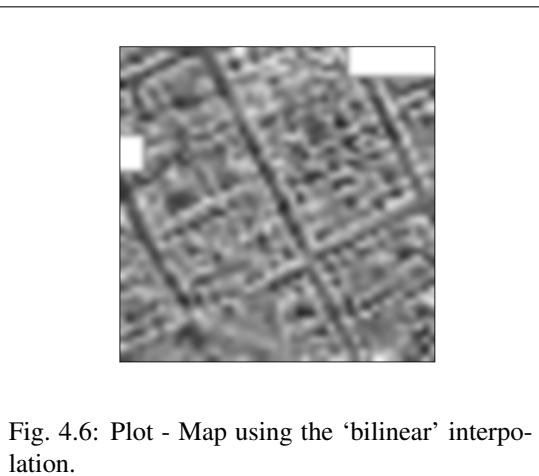
Fig. 4.6: Plot - Map using the 'bilinear' interpolation.

2-D Contour plots

You can plot a dataset as a 2-D (filled or unfilled) contour plot using:

```
>>> # Dataset 2-D contour plot
>>> fig, cmap = dataset.plot('2D-CONTOUR', levels=100, cmmin=-20, cmmax=20)
>>> fig.show()
```

```
>>> # Dataset 2-D Filled-contour plot
>>> fig, cmap = dataset.plot('2D-CONTOURF', levels=100, cmmin=-20, cmmax=20)
>>> fig.show()
```
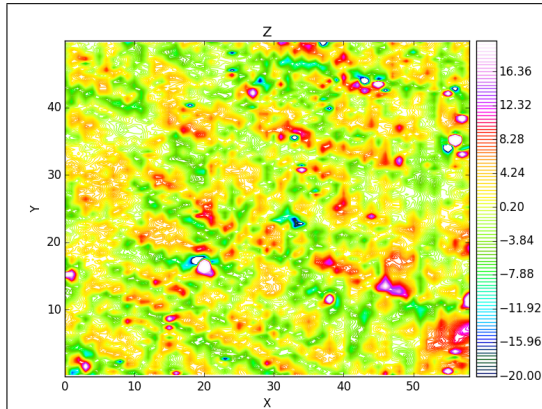

Fig. 4.7: Plot - Displaying data as a contour plot.
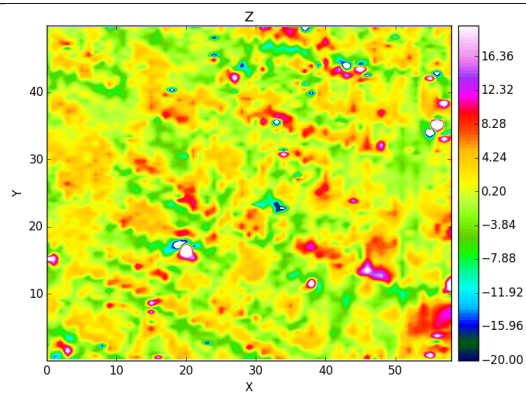
Fig. 4.8: Plot - Displaying data as a filled contour plot.

4.1.4 Plot options

Axis, label and color bar display

You can customize the display by enabling/disabling the axis, label and color bar display:

```
>>> # Custom axis, label and color map display
>>> dataset.plot(plottype='2D-SURFACE', labeldisplay=False)
>>> dataset.plot(plottype='2D-SURFACE', axisdisplay=False, cmapdisplay=False)
```
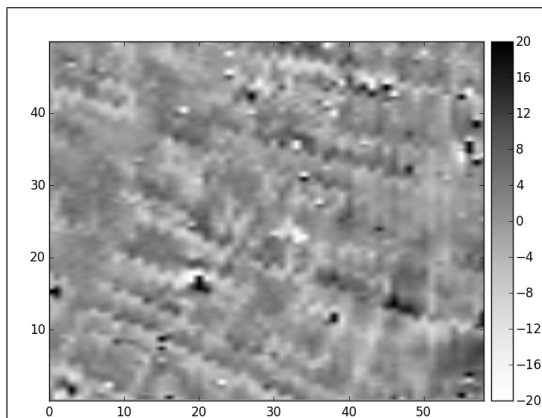

Fig. 4.9: Plot - Disabling the label display.
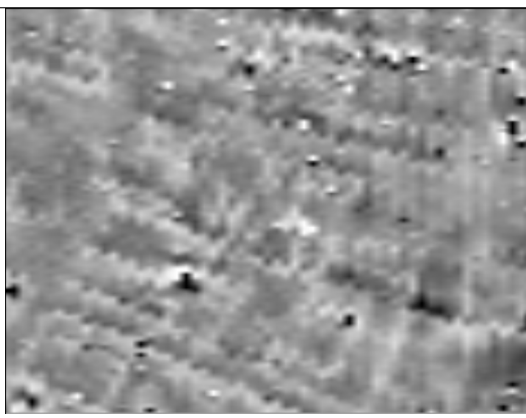
Fig. 4.10: Plot - Disabling the axis and color bar display.

Adding points and rectangles

Overlay some specific points or add rectangles:

```
>>> import geophpy.plotting.plot as gplt
```

```
>>> # Custom rectangles display
>>> xmin, xmax, ymin, ymax = dataset.get_gridextent()
>>> area_extents = [[15, 25, 15, 20], [35, xmax-0.5, 5, ymax-0.5]]
>>> rectangles = gplt.extents2rectangles(area_extents)
>>> dataset.plot(plottype='2D-SURFACE', rects=rectangles)
```

```
>>> # Custom points display
>>> points = [[22, 17], [45, 25]]
>>> dataset.plot(plottype='2D-SURFACE', rects=rectangles, points=points)
```
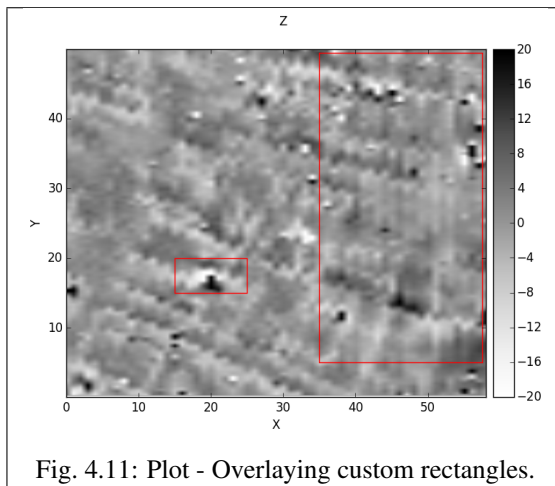

Fig. 4.11: Plot - Overlaying custom rectangles.
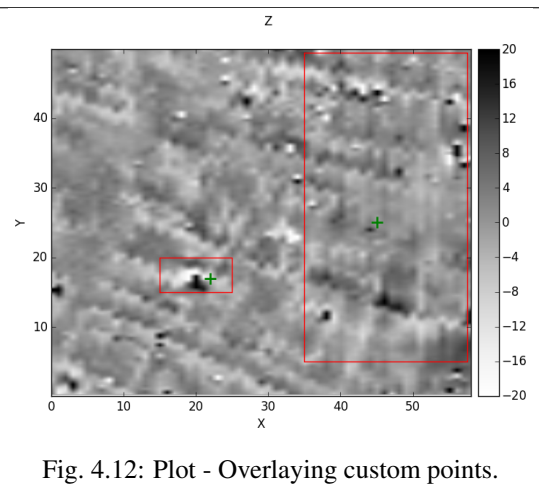
Fig. 4.12: Plot - Overlaying custom points.

Note: You can used the *extents2rectangles()* method of the *plot* module to convert area extent (xmin, xmax, ymin, ymax) to rectangle (x, y, width, height) to be displayed:

```
>>> import geophpy.plotting.plot as gplt
>>> area_extents = [ [xmin, xmax ymin, ymax], ...]
>>> rectangles = gplt.extents2rectangles(area_extents)
>>> dataset.plot(plottype='2D-SURFACE', rects=rectangles)
```

4.2 Histogram

To adjust the limits of color map you must view the limits of the data set:

```
>>> zmin, zmax = dataset.histo_getlimits()
```

You can plot the histogram curve in black, or respecting the dataset color map and with or without a color bar:

```
>>> # Black histogram
>>> valmin= -20, valmax = 20
>>> dataset.histo_plot(zmin=valmin, zmax=valmax, cmapdisplay=False,
→coloredhisto=False)
```

```
>>> # Colored histogram
>>> dataset.histo_plot(zmin=valmin, zmax=valmax, cmapdisplay=True, coloredhisto=True)
```

Fig. 4.13: Plot - Dataset Histogram.

Fig. 4.14: Plot - Dataset colored histogram.
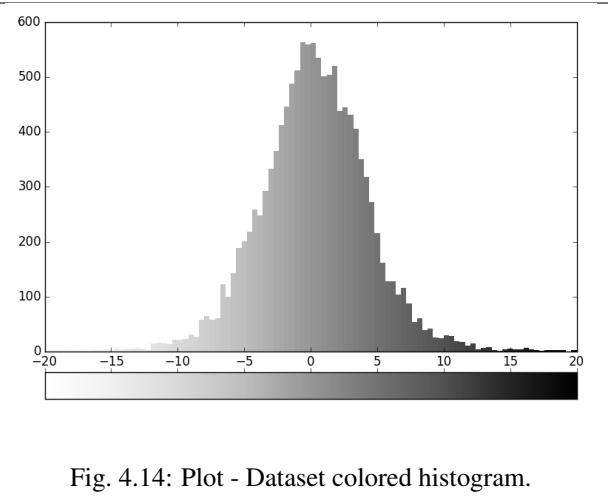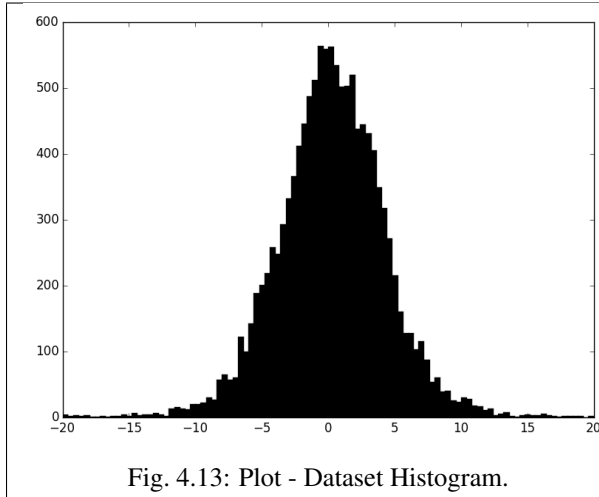
```
>>> # Saving histogram to a file
>>> dataset.histo_plot(filename='histogram.png')
```

4.3 Correlation

You can plot the correlation map between a profile and mean of its surrounding profiles (see *Festoon filtering*):

```
>>> dataset.correlation_plotmap(method="Crosscorr")
```

or the mean correlation profile is used as correlation map (see see *Festoon filtering*):

```
>>> dataset.correlation_plotsum(method="Crosscorr")
```


Fig. 4.15: Plot - Dataset correlation map.

Fig. 4.16: Plot - Dataset mean correlation profile.
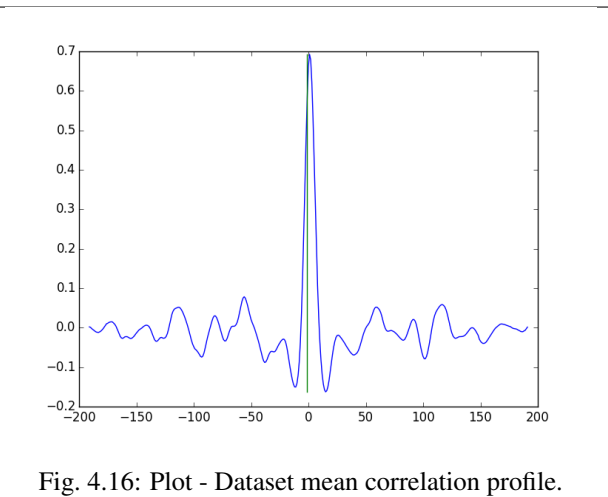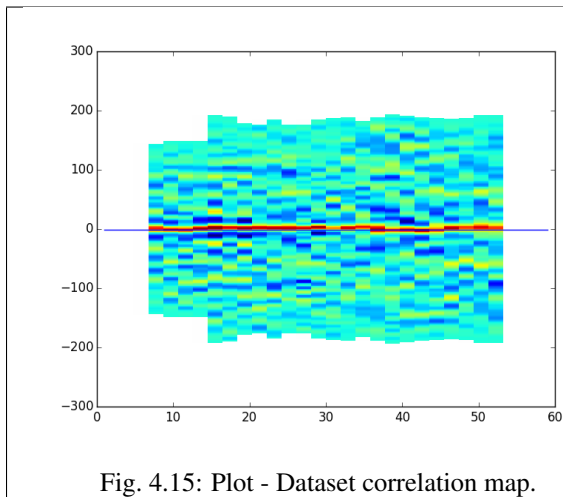
4.4 Mean cross-track profile

Before and after destriping mean cross-track profiles can be displayed with the following commands:

```
>>> dataset.meantrack_plot(Nprof=4, method='additive', Ndeg=None, plotflag='raw')
```

```
>>> dataset.meantrack_plot(Nprof=4, method='additive', Ndeg=None, plotflag='both')
```
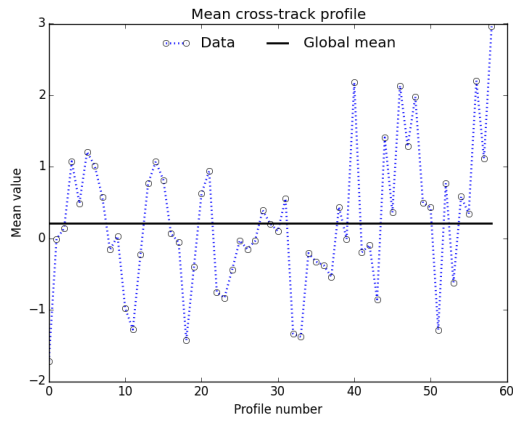

Fig. 4.17: Plot - Dataset raw mean cross-track profile.
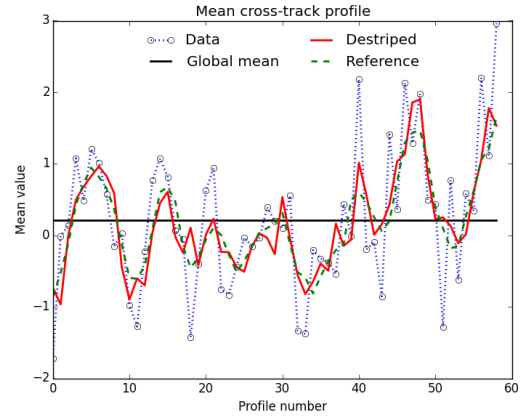
Fig. 4.18: Plot - Dataset destripped mean cross-track profile..

GENERAL PROCESSING

All the available processing techniques can be apply to a dataset through a simple command line of the form:

```
>>> dataset.ProcessingTechnique(option1=10, option2=True, option3='relative',...)
```

The available *general processing* techniques (i.e. not specific to a particular geophysical method) are listed in the sections below.

5.1 Peak filtering

Replace peaks in the dataset.

Peaks are detected using an *Hampel filter* or a *decision-theoretic median filter* and replaced by either NaNs or the local median value.

5.1.1 Examples

- replacing peaks using an *Hampel filter* to detect outliers:

```
>>> # Peaks are replaced by local median
>>> dataset.peakfilt(method='hampel', halfwidth=5, threshold=3)
```

```
>>> # or by NaNs
>>> dataset.peakfilt(method='hampel', halfwidth=5, threshold=3, setnan=True)
```

- replacing peaks using *decision-theoretic median filter* to detect outliers:

```
>>> # The threshold is a percentage of the local median value
>>> dataset.peakfilt(method='median', halfwidth=5, threshold=.05, mode='relative')
```

```
>>> # or a in raw unit
>>> dataset.peakfilt(method='median', halfwidth=5, threshold=15, mode='absolute')
```

5.1.2 Principle

A centered 1-D window is slided along a flattened version of the dataset and determines if the central element of the moving window is an outlier (peak) and has to be replaced.

To dertermine outliers, the median of the elements in the moving window is computed and used as a reference value. If the deviation of the central element of the window to the local median is above a threshold, the central value is replaced

by the local median (or NaNs depending on the filter configuration `setnan={True|False}`). Two different filters are implemented to dertermine outliers (a median filter and an Hampel filter).

Threshold value

For the median filter, the *threshold* can be defined as a percentage of the local median (`mode='relative'`) or directly in raw units (`mode='absolute'`).

For the Hampel filter, the higher the value of the *threshold* is, the less the filter will be selective. A threshold value of 0 is equivalent to a *standard median filter* (each element replaced by the value of the local median).

Filters definition

For f_k the central element of a moving window $\{f_{k-K}, ..., f_k, ..., f_{k+K}\}$ of half-with K and of local median $f^{\dagger} = $ median$\{f_{k-K}, ..., f_k, ..., f_{k+K}\}$

The Hampel filter is defined as [PeGa16] :

$$\mathcal{H}_{K,t}\{f_k\} = \begin{cases} f^{\dagger} & \text{if } |f_k - f^{\dagger}| > t \cdot Sk, \\ f_k & \text{otherwise}, \end{cases}$$

$$Sk = 1.4826 \cdot \text{median}\{|f_{k-K} - f^{\dagger}|, ..., |f_k - f^{\dagger}|, ..., |f_{k+K} - f^{\dagger}|\}$$

The (decision-theoretic) median filter is defined as:

$$\mathcal{M}_{K,t}\{f_k\} = \begin{cases} f^{\dagger} & \text{if } |f_k - f^{\dagger}| > t, \\ f_k & \text{otherwise}, \end{cases}$$

where is t the filter threshold.

5.1.3 Parameters

Name	Description	Type	Value
method	Type of the *decision-theoretic filter* used to determine outliers.	str	'median', 'hampel'
halfwidth	Filter half-width	int	5, 10 20, …
threshold	Filter threshold parameter. If t=0 and method='hampel', it is equal to a *standard median filter*.	int	0, 1, 2, 3, …
mode	Median filter mode. If 'relative', the threshold is a percentage of the local median value. If 'absolute', the threshold is a value.	str	'relative', 'absolute''
setnan	Flag to replace outliers by NaNs instead of the local median.	bool	`True` or `False`
valfilt	[For future implementation] Flag to apply filter on the ungridded data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.2 Thresholding

Dataset thresholding in a given interval.

5.2.1 Examples

```
>>> # Replacing out of range values by lower and upper bounds
>>> dataset.threshold(setmin=-10, setmax=10)
```
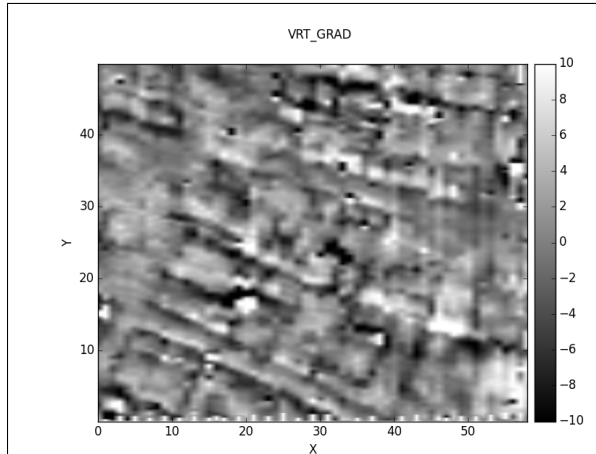

Fig. 5.1: Peak filter - Min, max thresholding - dataset.
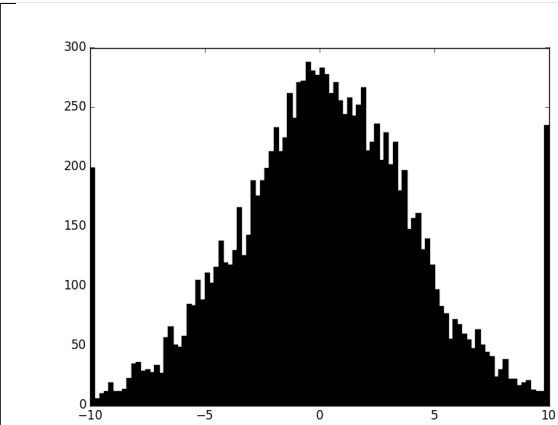
Fig. 5.2: Peak filter - Min, max thresholding - histogram.

```
>>> # by NaNs
>>> dataset.threshold(setmin=-10, setmax=10, setnan=True)
```
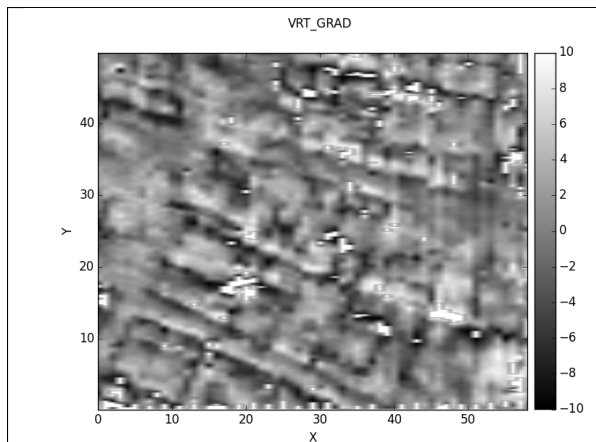

Fig. 5.3: Thresholding - NaN thresholding - dataset.
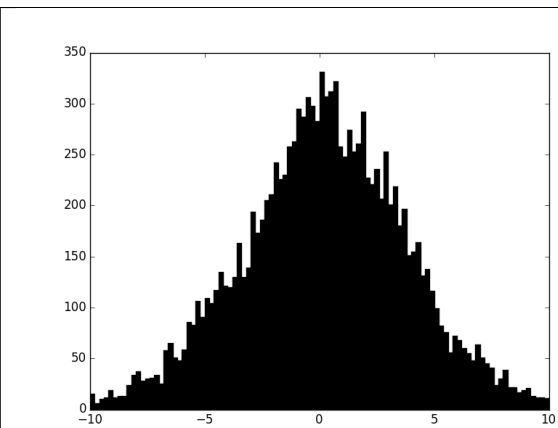
Fig. 5.4: Thresholding - NaN thresholding - histogram.

```
>>> # or by each profile's median
>>> dataset.threshold(setmin=-10, setmax=10, setmed=True)
```

Fig. 5.5: Thresholding - Median thresholding - dataset.

Fig. 5.6: Thresholding - Median thresholding - histogram.
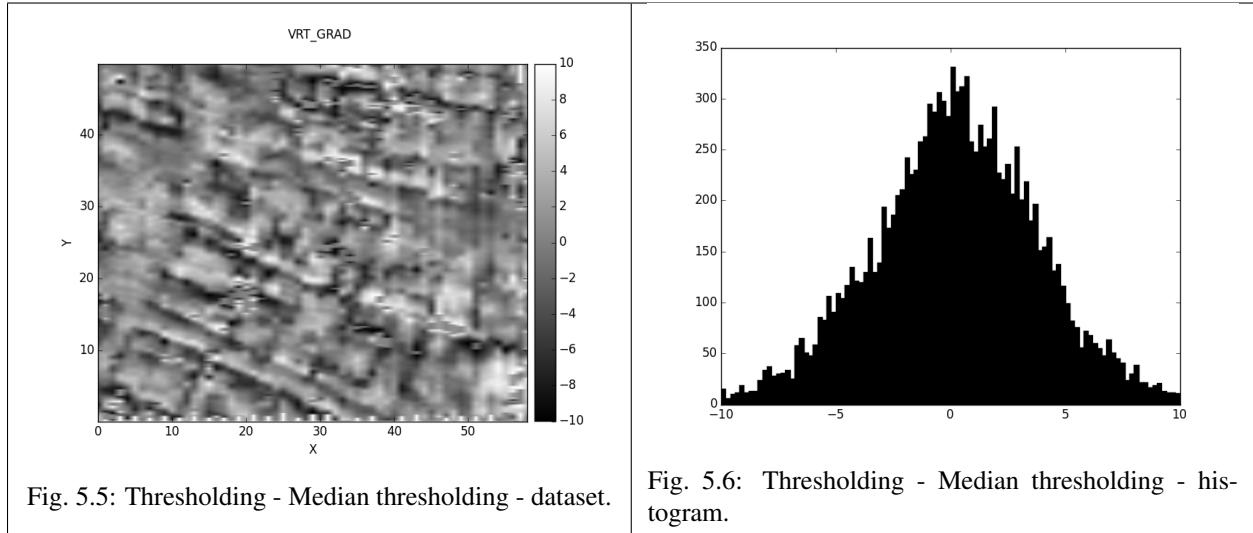
5.2.2 Principle

Each value of the dataset are compared to the given interval bounds. Depending on the filter configuration, values outside of the interval will be replaced by the interval bounds (`setmin=valmin`, `setmax=valmax`), NaNs (`setnan=True`), or the profile's median (`setmed=True`).

5.2.3 Parameters

Name	Description	Type	Value
set-min	Minimal interval value. All values lower than `setmin` will be replaced by `setmin` (if both `setmed` and `setnan` are `False`).	float	-5, 10, 42.5, . . .
set-max	Maximal interval value. All values lower than `setmax` will be replaced by `setmax` (if both `setmed` and `setnan` are `False`).	float	-5, 10, 42.5, . . .
setmed	Flag to replace out of bound data by the profile's median.	bool	`True` or `False`
set-nan	Flag to replace out of bound data by NaNs.	bool	`True` or `False`
val-filt	Flag to apply filter on the ungridded data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.3 Median filtering

Apply a median filter (*decision-theoretic* or *standard*) to the dataset.

5.3.1 Examples

```
>>> # No threshold : standard median filter
>>> dataset.medianfilt(nx=3, ny=3)
```
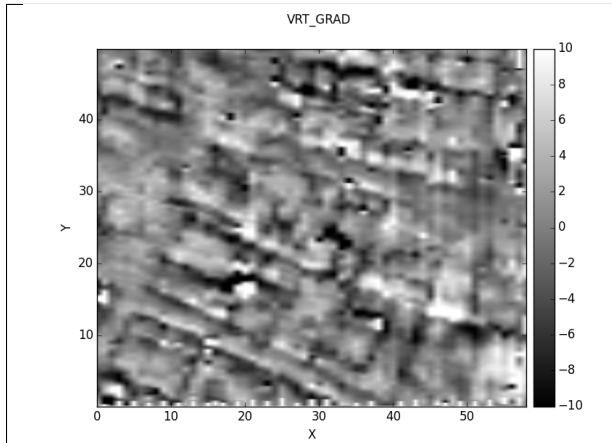

Fig. 5.7: Median filter - Raw dataset (no threshold).
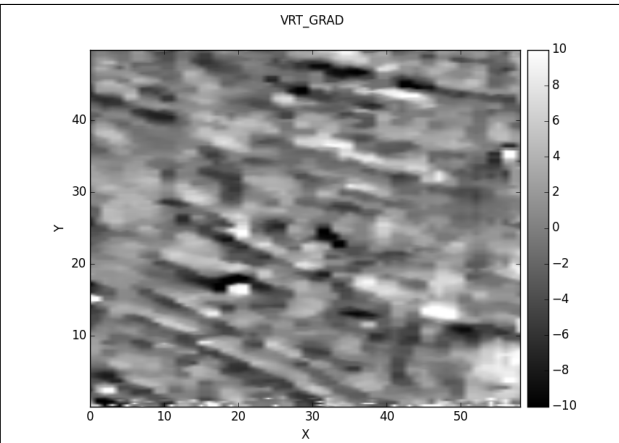
Fig. 5.8: Median filter - Filtered dataset (no threshold).

```
>>> # Threshold in raw unit : decision-theoretic median filter
>>> dataset.medianfilt(nx=3, ny=3, gap=5)
```
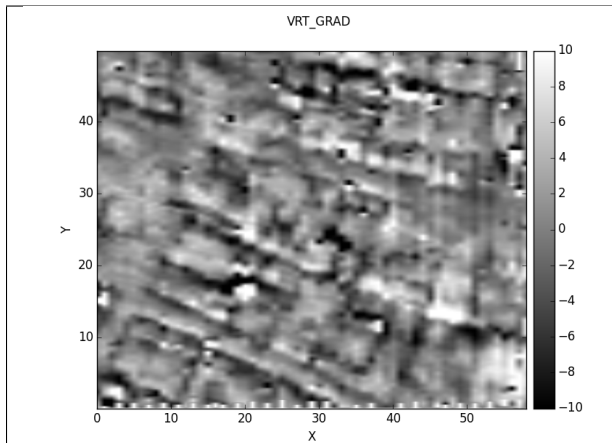

Fig. 5.9: Median filter - Raw dataset (threshold in raw unit).
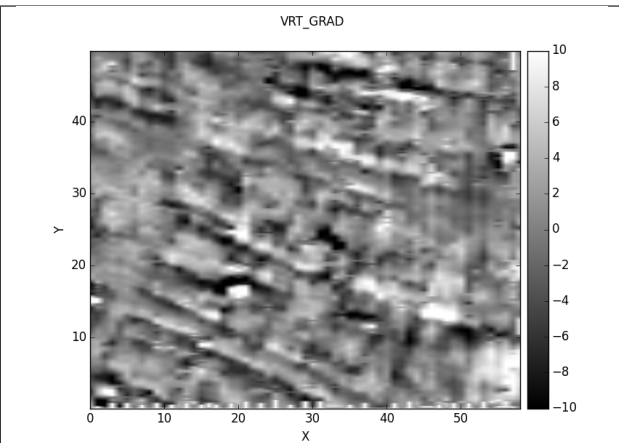
Fig. 5.10: Median filter - Filtered dataset (threshold in raw unit).

5.3.2 Principle

"Median filtering is a non linear process useful in reducing impulsive, or salt-and-pepper noise" [LimJ90]. It is capable of smoothing a few out of bounds pixels while preserving image's discontinuities without affecting the other pixels.

For each pixel in the dataset, the local median of the (**nx** x **ny**) neighboring points is calculated.
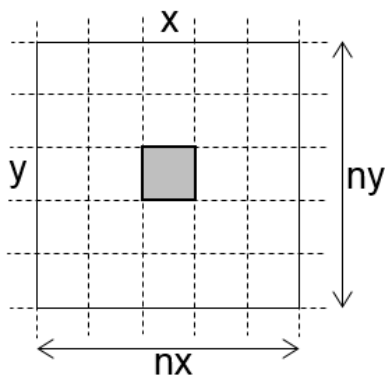
A threshold value is defined and if the deviation from the local median is higher than this threshold, then the center pixel value is replaced by the local median value. The threshold deviation from the local median can be defined:

- in percentage (`percent=10`) or raw units (`gap=5`),

- if no threshold is given, all pixels are replaced by their local medians (*standard median filter*).

5.3.3 Parameters

Name	Description	Type	Value
nx	Size, in number of sample, of the filer in the x-direction.	int	5, 10, 25, . . .
ny	Size, in number of sample, of the filer in the y-direction.	int	5, 10, 25, . . .
per-cent	Threshold deviation (in percents) to the local median value (for absolute field measurements).	float	
gap	Threshold deviation (in raw units) to the median value (for relative anomaly measurements).	float	
valfilt	[For future implementation] Flag to apply filter on the ungridded data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.4 Festoon filtering

Dataset destaggering.

The festoon filter is a destaggering filter that reduces the positioning error along the survey profiles that result in a festoon-like effect. An *optimum shift* is estimated based on the correlation of a particular profile and the mean of its two neighboring profiles.

This filter needs to be done as an early step in the processing flow as it needs to be done preferably before interpolation (or with no interpolation in the x-axis).

Filter applications: *data destaggering*

> **Warning:** The correlation map computation needs a regular grid WITHOUT interpolation between profiles (i.e. the x-axis direction). The correlation is done between each even profiles number and the mean of its two nearest neighbors. In the case of an interpolation in the x-axis, an interpolated (even) profile will be equal (or quasi-equal) to the mean of its two nearest neighbors. The computed map will be close to an auto-correlation map, resulting in a null shift optimum shift.

5.4.1 Examples

```
>>> # Gridding data without interpolation
>>> dataset.interpolate(interpolation='none')
>>>
>>> # Uniform shift
>>> dataset.festoonfilt(method='Crosscorr', uniformshift=True)
```


Fig. 5.11: Destaggering - Raw dataset (uniform shift).

Fig. 5.12: Destaggering - Filtered dataset (uniform shift).
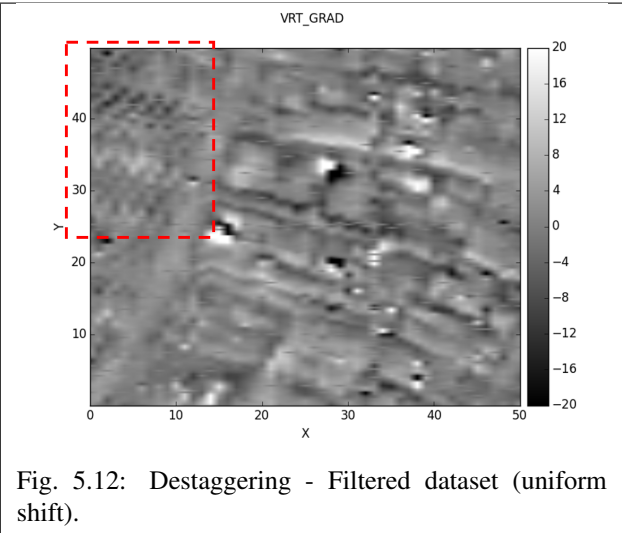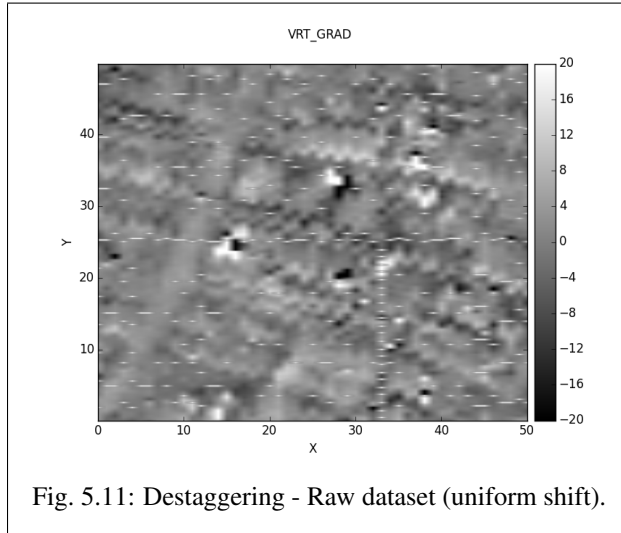
```
>>> # Non uniform shift
>>> dataset.festoonfilt(method='Crosscorr', uniformshift=False)
```


Fig. 5.13: Destaggering - Raw dataset (non uniform shift).

Fig. 5.14: Destaggering - Filtered dataset (non uniform shift).
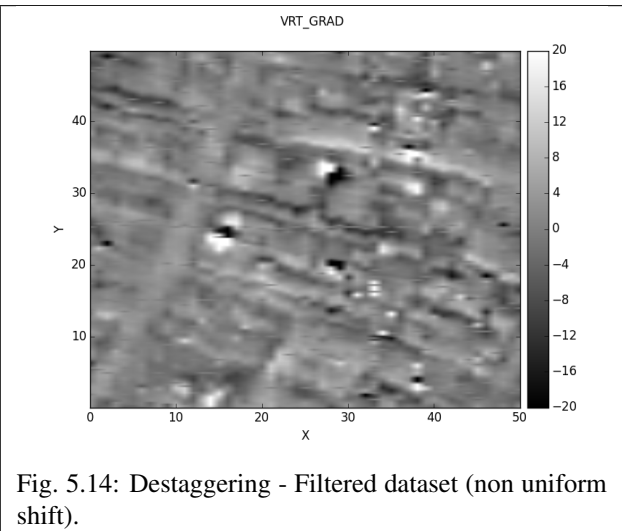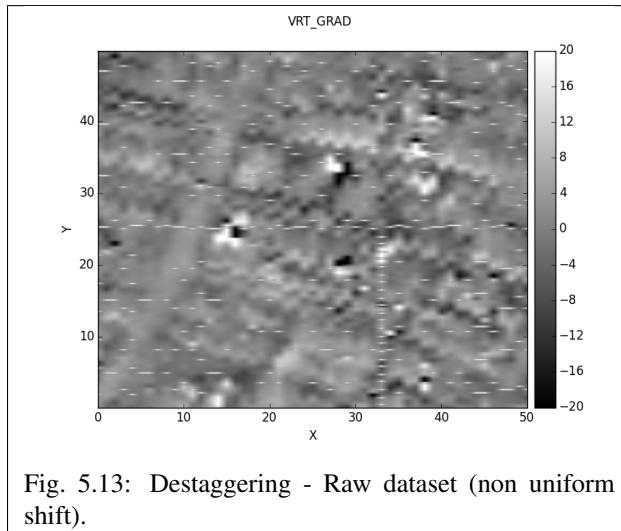
5.4.2 Principle

For every even profiles (columns) in the dataset, an optimum shift is estimated based on the correlation of the profile and the mean of its two nearest neighbors. For each possible shift, a correlation map is hence computed and used to estimate the optimum shift (max of correlation).

The optimum shift can be set to be uniform throughout the map (`uniformshift=True`) or different for each profile (`uniformshift=False`). If the shift is set uniform, the mean correlation profile is used as correlation map.

At the top and bottom edges of the correlation map (high shift values), high correlation values can arrise from low sample correlation calculation. To prevent those high correlation values to drag the best shift estimation, a limitation is set to only consider correlation with at least 50% overlap between profiles. Similarly, a minimum correlation value (`corrmin`) can be defined to prevent profile's shift if the correlation is too low.

```
>>> # Correaltion map
>>> dataset.correlation_plotmap()
```

```
>>> # Correaltion profile
>>> dataset.correlation_plotsum()
```
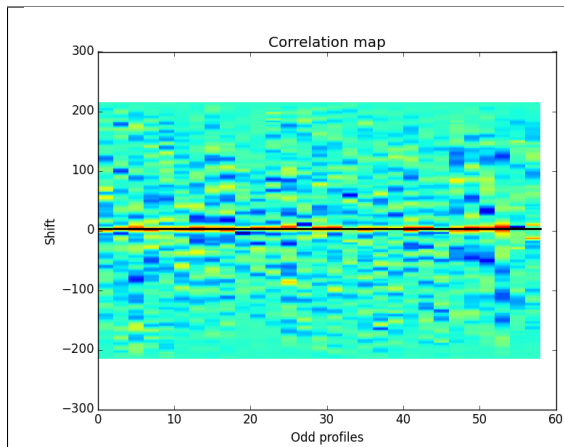

Fig. 5.15: Destaggering - Correlation map.
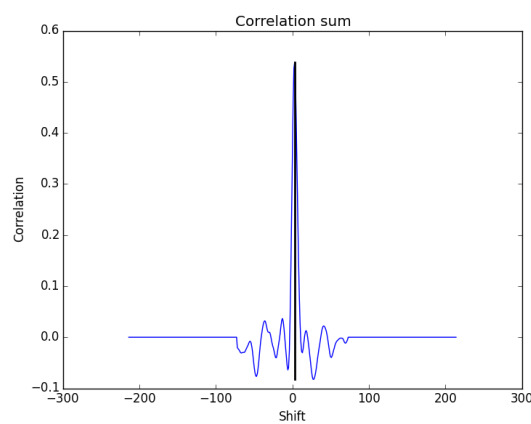
Fig. 5.16: Destaggering - Mean correlation profile.

5.4.3 Parameters

Name	Description	Type	Value
method	Correlation method to use to compute the correlation coefficient in the correlation map.	str	'Crosscorr', 'Pearson', 'Spearman' or 'Kendall'
shift	Optional shift value (in pixels) to apply to the dataset profiles. If shit=0, the shift will be determined for each profile by correlation with neighbors. If shift is a vector each value in shift will be applied to its corresponding even profile. In that case shift must have the same size as the number of even profiles.	int or array of int	3, 5, 10, [2, 3, 4, …, 3, 4] or 0
cor-rmin	Minimum correlation coefficient value to allow shifting.	float (in the range [0-1])	0.6, 0.8
uni-formshift	Flag to use a uniform shift on the map or a different one for each profile.	bool	`True` or `False`
set-min	Data values lower than `setmin` are ignored.	float	12, 44.5, …, `None`
set-max	Data values higher than `setmin` are ignored.	float	12, 44.5, …, `None`
val-filt	[For future implementation] Flag to apply filter on the ungridded data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.5 Detrending

… To Be Developed …

Subtracting a polynomial fit for each profile in the dataset.

5.6 Regional trend filtering

… To Be Developed …

Remove the background (or regional response) from a dataset to highlight the sub-surface features of interest.

5.6.1 Example

5.6.2 Principle

5.6.3 Parameters

See *High level API* for calling details.

5.7 Wallis filtering

The Wallis filter is a locally adaptative contrast enhancement filter.

Based on the local statistical properties of sub-window in the image, it adjusts brightness values (grayscale image) in the local window so that the local mean and standard deviation match target values.

Filter applications: *contrast enhancement*

5.7.1 Examples

```
>>> dataset.wallisfilt()
```
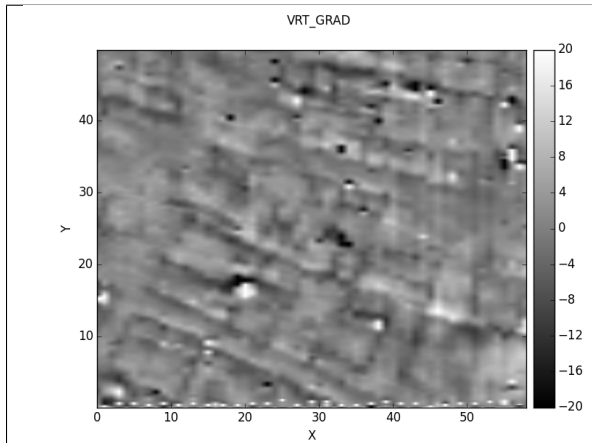

Fig. 5.17: Wallis Filter - Raw dataset.
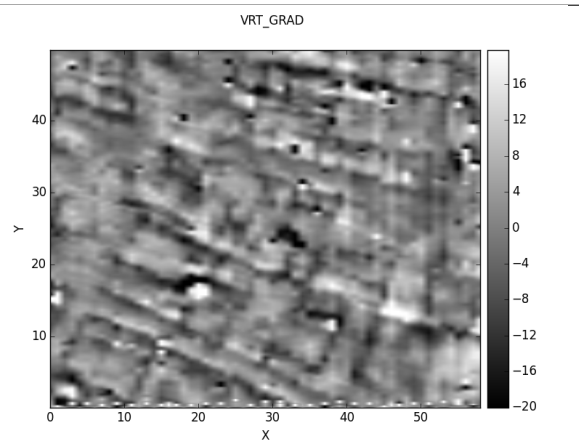
Fig. 5.18: Wallis Filter - Filtered dataset.

5.7.2 Principle

A window of size (**nx**, **ny**) is slided along the image and at each pixel the Wallis operator is calculated. The Wallis operator is defined as [STHH90]:

$$\frac{A\sigma_d}{A\sigma_{(x,y)} + \sigma_d}[f_{(x,y)} - m_{(x,y)}] + \alpha m_d + (1 - \alpha)m_{(x,y)}$$

where:

- A is the amplification factor for contrast,
- σ_d is the target standard deviation,
- $\sigma_{(x,y)}$ is the standard deviation in the current window,
- $f_{(x,y)}$ is the center pixel of the current window,
- $m_{(x,y)}$ is the mean of the current window,
- α is the edge factor (controlling portion of the observed mean, and brightness locally to reduce or increase the total range),
- m_d is the target mean.

As the Wallis filter is design for grayscale image, the data are internally converted to brightness level before applying the filter. The conversion is based on the minimum and maximum value in the dataset and uses 256 levels (from 0 to 255). The filtered brightness level are converted back to data afterwards.

A quite large window is recommended to ensure algorithm stability.

5.7.3 Parameters

Name	Description	Type	Value
nx	Size, in number of sample, of the filer in the x-direction.	int	5, 10, 25, …
ny	Size, in number of sample, of the filer in the y-direction.	int	5, 10, 25, …
targmean	The target standard deviation (in gray level).	int	125
set-gain	Amplification factor for contrast.	int	8
limitst-dev	imitation on the window standard deviation to prevent too high gain value if data are dispersed.	int	25
edge-factor	Brightness forcing factor (α), controls ratio of edge to background intensities.	float (in the range of [0,1])	
valfilt	[For future implementation] Flag to apply filter on the scattered data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.8 Ploughing filtering

Directional filter.

Apply a directional filter to reduce agricultural ploughing effect in the dataset (or any other directional feature).

5.8.1 Examples

```
>>> dataset.ploughfilt(apod=0, azimuth=90, cutoff=100, width=3)
```
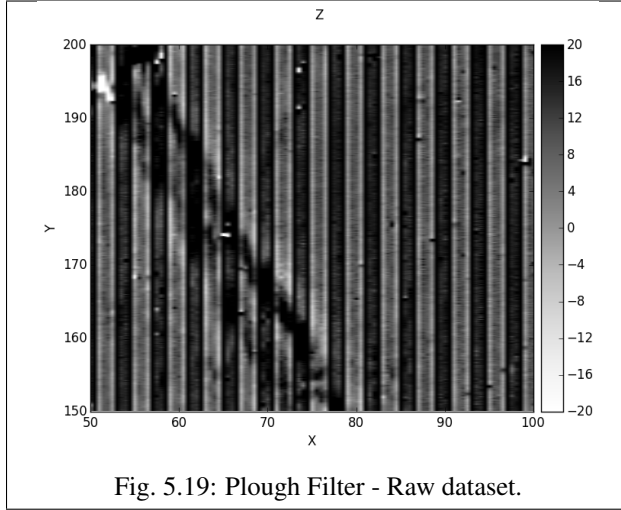
Fig. 5.19: Plough Filter - Raw dataset.
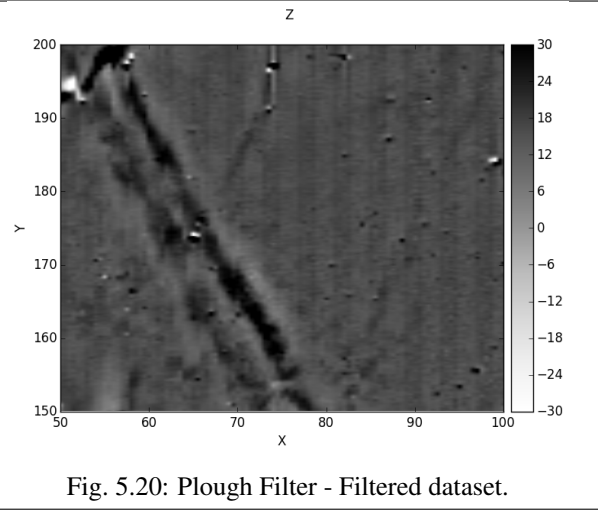
Fig. 5.20: Plough Filter - Filtered dataset.

```
>>> # Raw dataset spectral plot
>>> dataset.spectral_plotmap(plottype='magnitude', logscale=True)
>>> dataset.plot_directional_filter(apod=0, azimuth=90, cutoff=100, width=3)
```
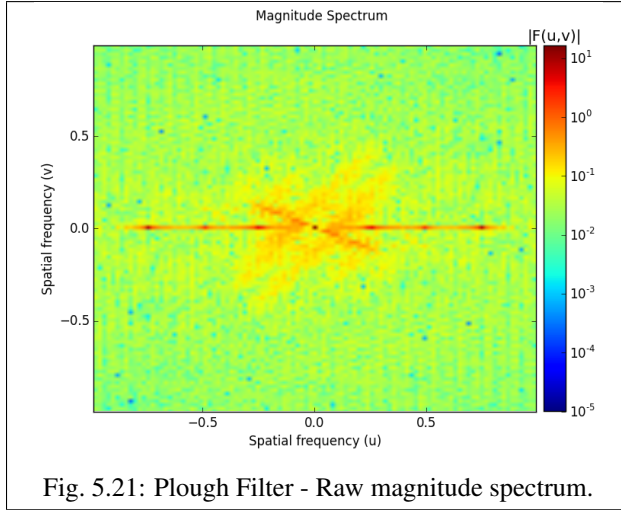

Fig. 5.21: Plough Filter - Raw magnitude spectrum.
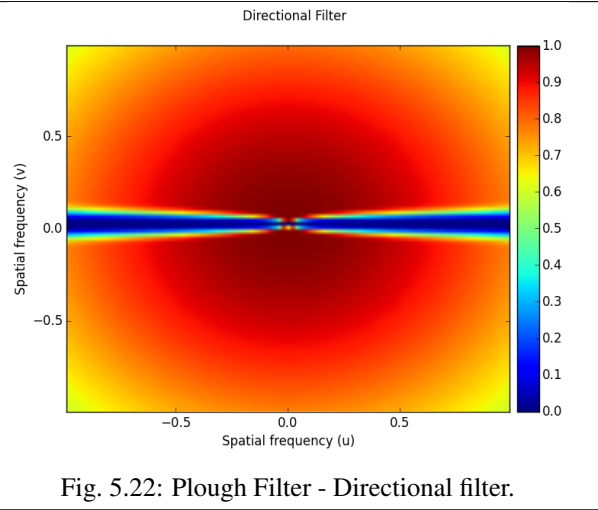
Fig. 5.22: Plough Filter - Directional filter.

5.8.2 Principle

The directional feature in the dataset is filtered in the spectral domain using the combination ($\mathcal{F}$) of a *gaussian low-pass filter* of order 2 ($\mathcal{F}_{GLP}$) and a *gaussian directional filter* ($\mathcal{F}_{DIR}$) defined as [TABB01] :

$$\mathcal{F}(\rho, \theta, f_c) = \mathcal{F}_{GLP}(\rho, f_c) * \mathcal{F}_{DIR}(\rho, \theta)$$

$$= e^{-(\rho/f_c)^2} * \left(1 - e^{-\rho^2/\tan(\theta - \theta_0)^n}\right)$$

where:

- ρ and θ are the current point polar coordinates,

- f_c is the gaussian low-pass filter cutoff frequency,

- θ_0 is the directional filter's azimuth,

- n is the parameter that controls the filter width.

The filter's width is determined by the value of n (Fig. 5.23, Fig. 5.24 and Fig. 5.25) and the *gaussian low-pass filter* component is neglected if no cutoff frequency is defined (cutoff=None, see Fig. 5.26)

```
>>> dataset.plot_directional_filter(azimuth=30, cutoff=50, width=2)
>>> dataset.plot_directional_filter(azimuth=30, cutoff=50, width=3)
```


Fig. 5.23: Plough Filter - Directional Filter (**n=2**)

Fig. 5.24: Plough Filter - Directional Filter (**n=3**)
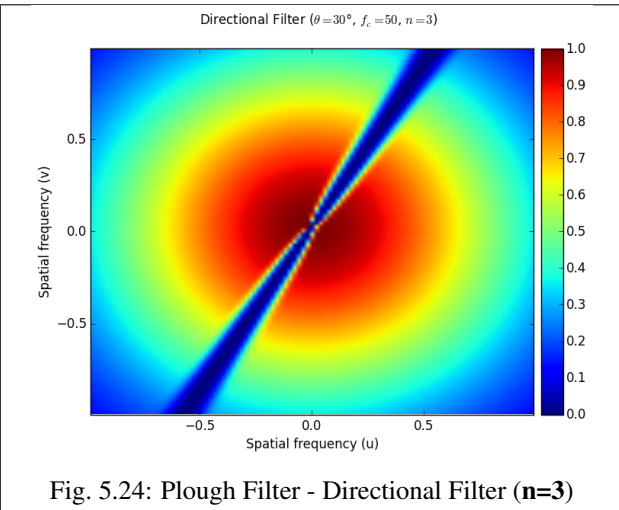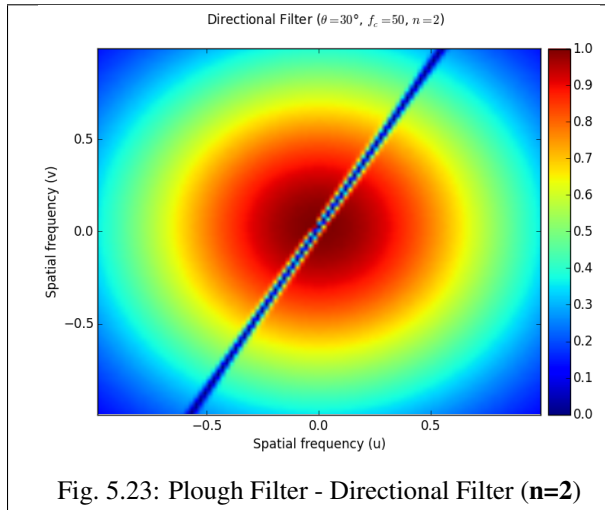
```
>>> dataset.plot_directional_filter(azimuth=30, cutoff=50, width=4)
>>> dataset.plot_directional_filter(azimuth=30, cutoff=None, width=4)
```


Fig. 5.25: Plough Filter - Directional Filter (**n=4**)

Fig. 5.26: Plough Filter - Pure directional filter (**fc=None**)
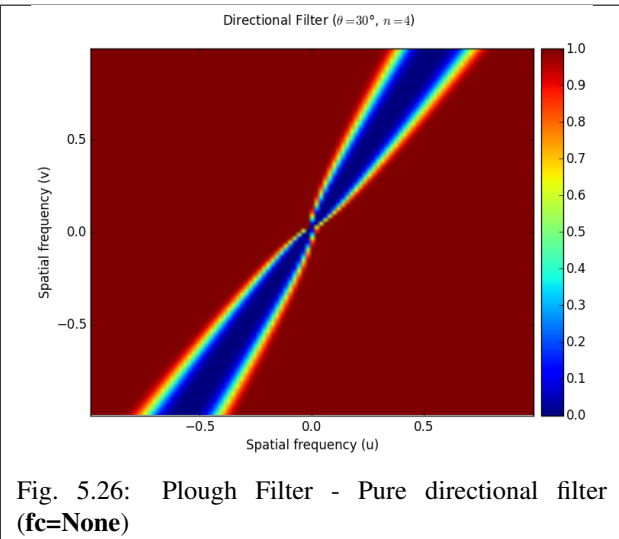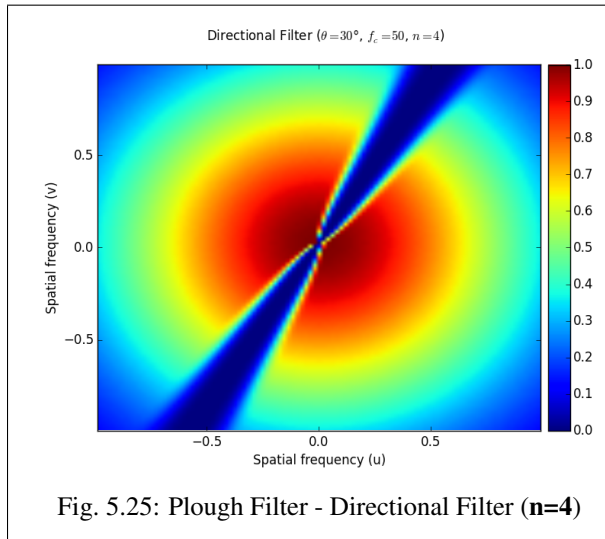
5.8.3 Parameters

Name	Description	Type	Value
apod	Apodization factor (%), to limit Gibbs phenomenon at jump discontinuities.	float	0, 5, 10, 20, 25, . . .
az-imuth	Filter azimuth in degree.	float	0, 10, 33.25, . . .
cutoff	cutoff spatial frequency (in number of sample).	int	5, 10, 100, . . . or `None`
width	ilter width parameter.	int	2, 3, 4, . . .
valfilt	[For future implementation] Flag to apply filter on the scattered data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.9 Zero-Mean Traversing

Subtracts the mean (or median) of each traverse (profile) in the dataset.

Filter applications: *(magnetic) probe compensation, data destriping, edge matching. . .*

5.9.1 Examples

```
>>> # Raw data display
>>> dataset.plot(plottype='2D-SURFACE')[0].show()
>>> dataset.histo_plot(cmapdisplay=False, coloredhisto=True).show()
```
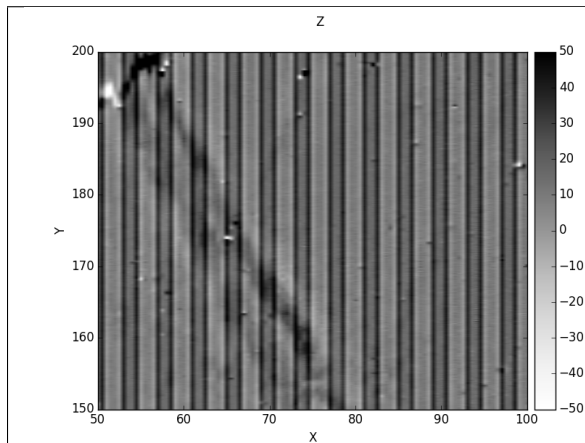

Fig. 5.27: Zero-mean traverse - Raw dataset.
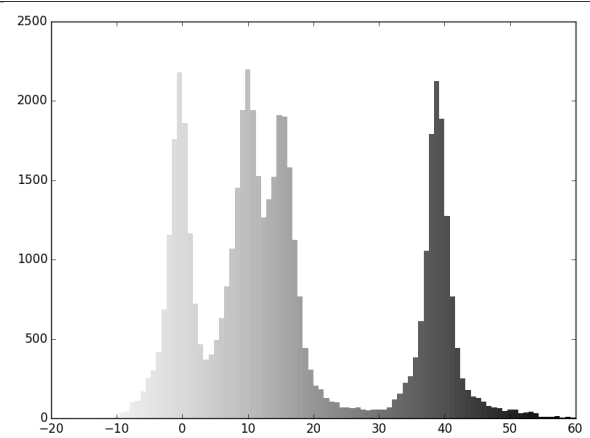
Fig. 5.28: Zero-mean traverse - Raw histogram.

```
>>> # Zero-Mean Traverse data
>>> dataset.zeromeanprofile(setvar='mean')
```
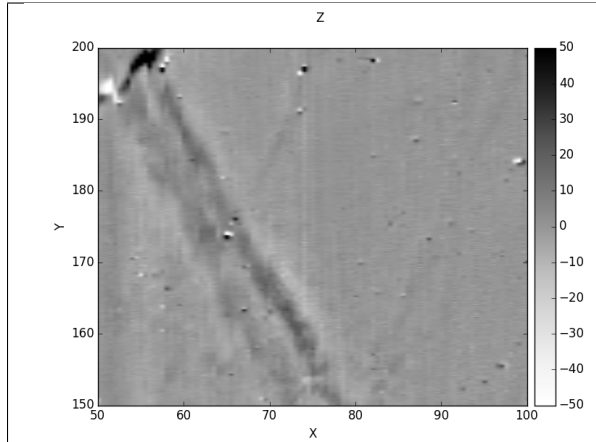
Fig. 5.29: Zero-mean traverse - Filtered dataset (zero-mean).
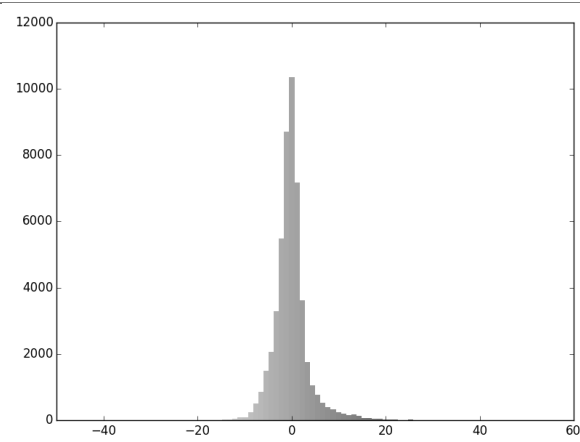
Fig. 5.30: Zero-mean traverse - Filtered histogram (zero-mean).

```
>>> # Zero-Median Traverse data
>>> dataset.zeromeanprofile(setvar='median')
```
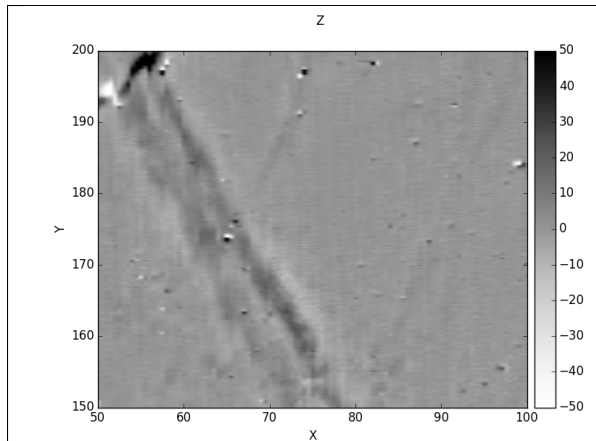

Fig. 5.31: Zero-mean traverse - Filtered dataset (zero-median).
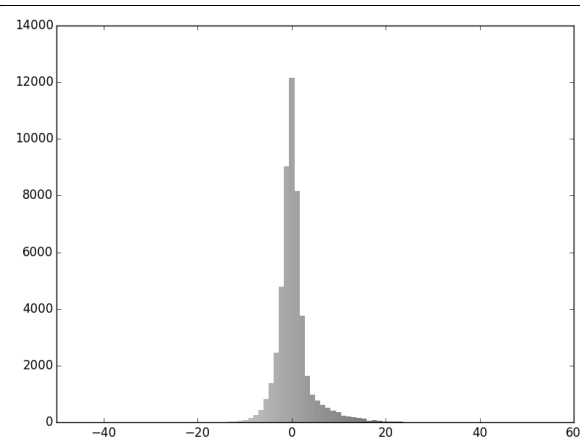
Fig. 5.32: Zero-mean traverse - Filtered histogram (zero-median).

5.9.2 Principle

For each traverse (profile) in the dataset, the mean (or median) is calculated and subtracted, leading to a zero-mean (or zero-median) survey [AsGA08].

Note: This filter is strictly equivalent to the constant destriping filter in configuration *'mono'* sensor, using *'additive'* destriping method and *Nprof=0*:

```
>>> dataset.zeromeanprofile(setvar='median')
>>> dataset.destripecon(Nprof=0, method='additive', config='mono', reference='median')
```

5.9.3 Parameters

Name	Description	Type	Value
set-var	Profile's statistical property be subtracted from each profile.	str	'mean' or 'median'
set-min	Data values lower than `setmin` are ignored.	float	12, 44.5, ..., `None`
set-max	Data values higher than `setmin` are ignored.	float	12, 44.5, ..., `None`
valfilt	[For future implementation] Flag to apply filter on the scattered data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.10 Constant destriping

Remove the strip noise effect from the dataset.

The strip noise effect arises from profile-to-profile differences in sensor height, orientation, drift or sensitivity (multi-sensors array). Constant destriping is done using Moment Matching method [GaCs00].

Filter applications: *(magnetic) probe compensation, data destriping, edge matching...*

5.10.1 Examples

```
>>> dataset.destripecon(Nprof=4, method='additive')
```
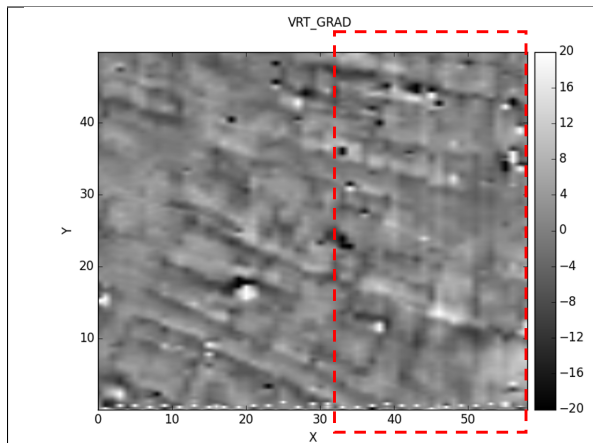

Fig. 5.33: Destriping - Raw dataset.
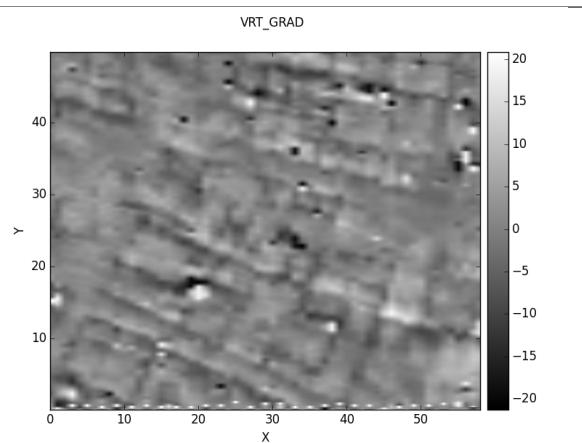
Fig. 5.34: Destriping - Filtered dataset.

5.10.2 Principle

In constant destriping, a linear relationship is assumed between profile-to-profile offset (means) and gain (standard deviation). The statistical moments (mean m_i and standard deviation σ_i) of each profile in the dataset are computed and matched to reference values.

Reference values

Typical reference values are:

- the mean (m_d) and standard deviation (σ_d) of the N neighboring profiles (`Nprof=4`),

- the mean and standard deviation of the global dataset (`Nprof='all'`),

- the mean and standard deviation of each profiles (`Nprof=0`, zero-mean traverse),

- alternatively, one can use the median and interquartile range instead of mean and standard deviation (`reference='median'`).

Additive vs multiplicative destriping

The corrected value can be calculated by

- an additive relation (`method='additive'`) [RiJi06], [Scho07]:

$$f_{corr} = (f - m_i)\frac{\sigma_d}{\sigma_i} + m_d$$

- or a multiplicative relation (`method='multiplicative'`):

$$f_{corr} = f\frac{\sigma_d}{\sigma_i}\frac{m_d}{m_i}$$

where f_{corr} is the corrected value, σ_d is the reference standard deviation, σ_i is the current profile standard deviation, f is the current value, m_i is the current profile and m_d is the reference mean.

Note: Ground surveys (unlike remote sensing) often demonstrate profile-to-profile offsets but rarely gain changes so that only matching profiles mean (or median) is usually appropriate (`configuration='mono'`):

$$f_{corr} = f - m_i + m_d$$

This is similar to the zero-mean filter, with the difference that the profile mean (or median) is set to a reference value instead of zero.

If `Nprof` is set to zero for the reference computation, this is strictly equivalent to the zero-mean filter:

```
>>> dataset.destripecon(Nprof=0, method='additive', config='mono', reference=
↪'mean')
>>> # Strictly equals to
>>> dataset.zeromeanprofile(setvar='mean')
```

Summary of the destiping configurations

Configuration / Method	`'additive'`	`'multiplicative'`
`'mono'`	$f_{corr} = f - m_i + m_d$	$f_{corr} = f\frac{m_d}{m_i}$
`'multi'`	$f_{corr} = (f - m_i)\frac{\sigma_d}{\sigma_i} + m_d$	$f_{corr} = f\frac{\sigma_d}{\sigma_i}\frac{m_d}{m_i}$

Mean cross-track profile display

The data mean cross-track profile before and after destriping can be displayed as follow:

```
>>> dataset.meantrack_plot(Nprof=4, method='additive', reference='median',␣
↪plotflag='raw')
>>> dataset.meantrack_plot(Nprof=4, method='additive', reference='median',␣
↪plotflag='both')
```
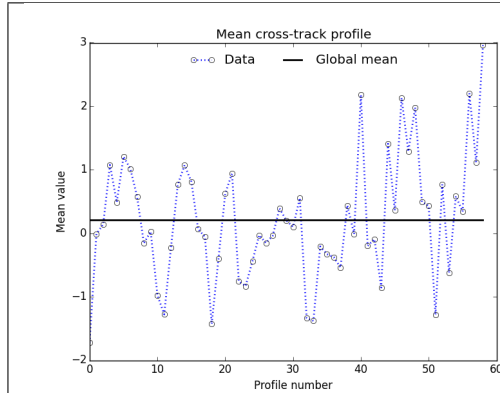

Fig. 5.35: Destriping - mean cross-track profile (before).
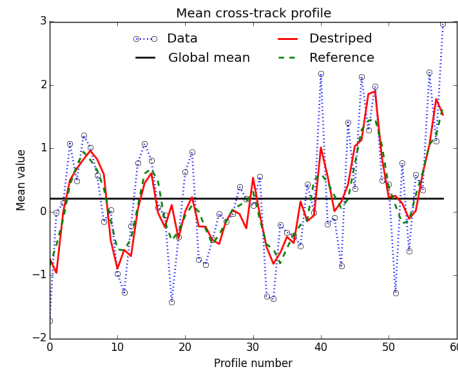
Fig. 5.36: Destriping - mean cross-track (after).

5.10.3 Parameters

Name	Description	Type	Value
Nprof	Number of neighboring profiles used to compute the the reference values.	int or 'all'	0, 4 ,… or 'all'
set-min	Data values lower than `setmin` are ignored.	float	12, 44.5, …, `None`
set-max	Data values higher than `setmin` are ignored.	float	12, 44.5, …, `None`
method	Destriping methode.	str	'additive' or 'multiplicative'
reference	References used for destriping.	str	'mean' or 'median'
config	Sensors configuration.	str	'mono' or 'multi'
valfilt	[For future implementation] Flag to apply filter on the scattered data values rather than on the gridded data.	bool	`True` or `False`

See *High level API* for calling details.

5.11 Curve destriping

… To Be Developed …

Remove from the dataset the strip noise effect by fitting and subtracting a MEAN polynomial curve to each profile on the dataset.

MAGNETIC PROCESSING

All the available processing techniques specific to magnetic surveys can be apply to a dataset through a simple command line of the form:

```
>>> dataset.ProcessingTechnique(option1=10, option2=True, option3='relative',...)
```

This section describes the different available processing specific to magnetic survey data.

6.1 General consideration

6.1.1 Spectral domain

As most of the computation are done in the spectral domain, classic pre-processing and post-processing operations are automatically performed:

Filling gaps

To use the Fast Fourier Transform algorithm, the dataset must not contain gaps or 'NaN' (Not A Number) values. If the data set grid is not interpolated and contains NaNs, the blanks will be automatically filled using profile by profile linear interpolation prior to the Fourier Transform. After the data filtering, the gaps will be automatically unfilled using NaNs.

Apodization

To limit Gibbs phenomenon at jump discontinuities, the dataset edges are extended and then smoothed using a cosine window. The factor of apodization (0, 5, 10, 15, 20 or 25%) gives the dataset extension size. Values in the extension will be attenuated using a cosine:

$na = N * \left(\frac{fap}{100}\right)$, with N = Nx or Ny, and fap the apodisation factor (in %)

$Zone\ 1 : value = full\ value$

$Zone\ 2 : value = full\ value * \cos\left(\frac{\pi}{2} * \frac{na - n}{na}\right)$, with $na = nax$ or nay, and $0 \le n \le na$

$Zone\ 2' : value = full\ value * \cos\left(\frac{\pi}{2} * \frac{n}{na}\right)$, with $na = nax$ or nay, and $0 \le n \le na$
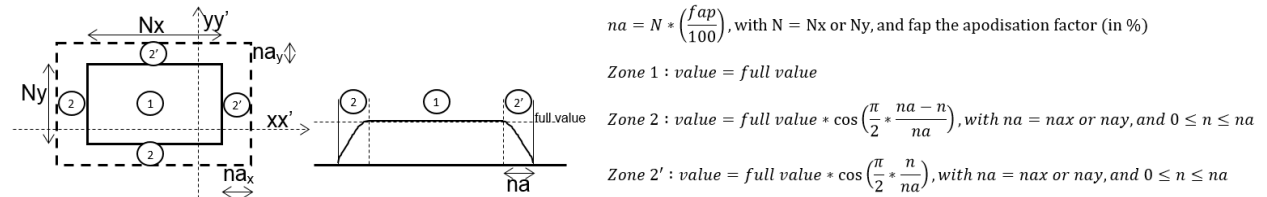
After processing, only the filtered values in the original dataset extension are kept so that the original data size is respected.

Directional derivatives computation

The nth directional derivatives of the a potential field are computed in the spectral domain using [BLAK96]:

$$\mathcal{F}\left[\frac{\partial^n T}{\partial x^n}\right] = (ik_x)^n \mathcal{F}[T], \mathcal{F}\left[\frac{\partial^n T}{\partial y^n}\right] = (ik_y)^n \mathcal{F}[T], \mathcal{F}\left[\frac{\partial^n T}{\partial z^n}\right] = |k|^n \mathcal{F}[T].$$

6.1.2 Magnetic field

System of coordinates
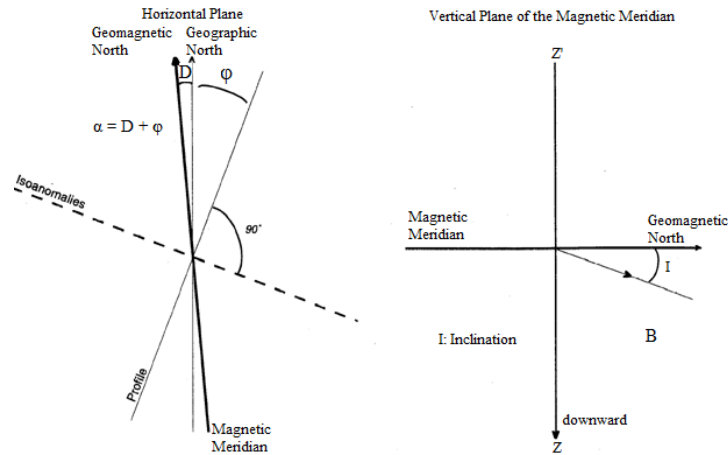
Fig. 6.1: Reduction to the Pole - Magnetic inclination (I), declination (D) and azimuth (θ) definition.

Ambient field vector components

Vector components

$$B_x = |B| \cdot f_x$$
$$B_y = |B| \cdot f_y$$
$$B_z = |B| \cdot f_z$$

where are $f = (f_x, f_y, f_z)$ is the unit-vector in the direction of the ambient field defined as:

$$f_x = \cos(I)\cos(\phi)$$
$$f_y = \cos(I)\sin(\phi)$$
$$f_z = \sin(I)$$

6.2 Logarithmic transformation

The logarithmic transformation is contrast enhancement filter.

Originally used for geological magnetic data, it enhances information present in the data at low-amplitude values while preserving the relative amplitude information via logarithmic transformation procedure [MPBL01].

Filter applications: *(magnetic) contrast enhancement*

6.2.1 Examples

```
>>> dataset.logtransform(multfactor=1e3, setnan=False, valfilt=False)
```

Fig. 6.2: Log Transform - Raw dataset.

Fig. 6.3: Log Transform - Filtered dataset.

Fig. 6.4: Log Transform - Raw dataset histogram.

Fig. 6.5: Log Transform - Filtered dataset histogram.
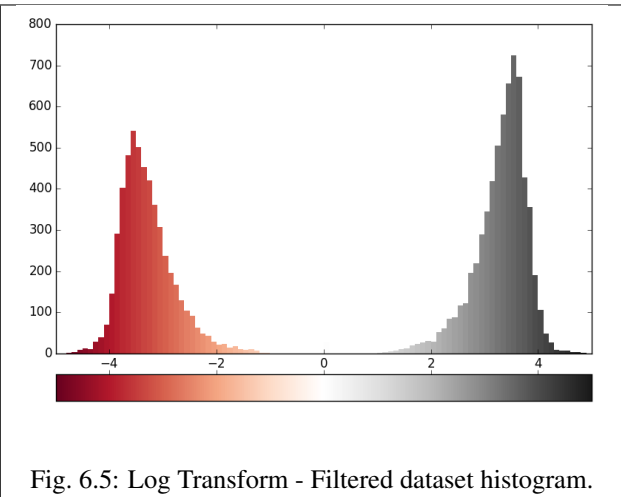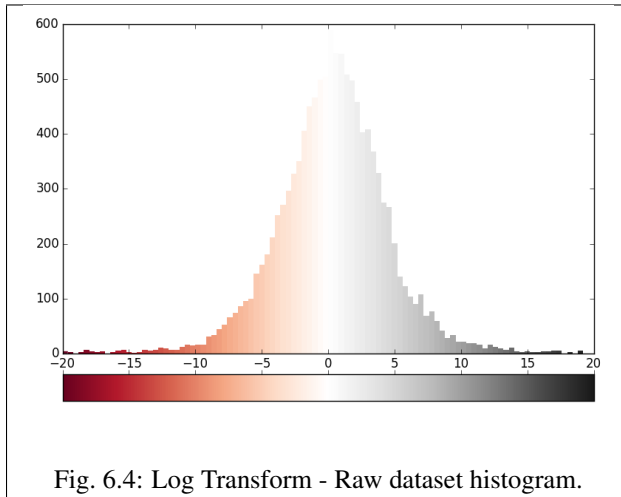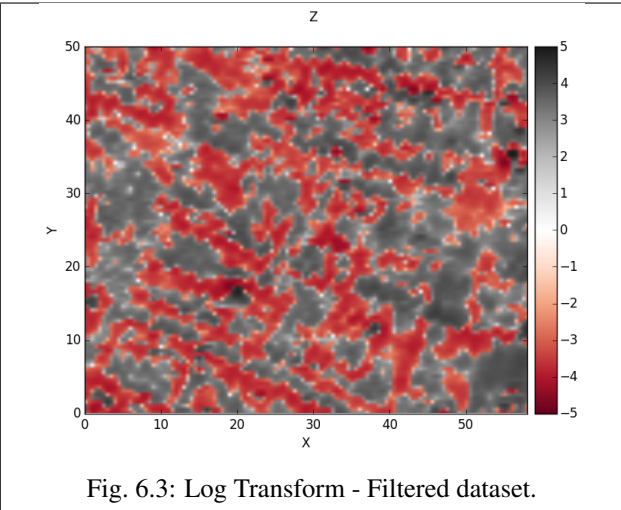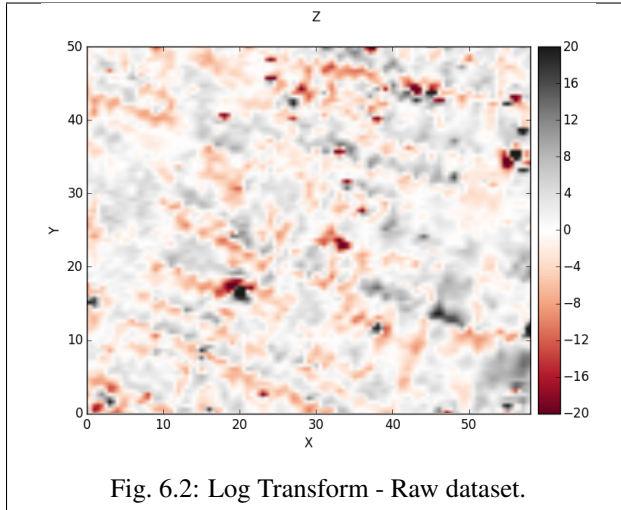
6.2.2 Principle

Contrast in magnetic susceptibility and magnetic remanence are the physical rock properties that controls magnetic anomalies, as well as the geometry and the position of the source body.

Magnetic mineral content variation and borehole magnetic susceptibility are know to be best represented by a log-normal distribution. Assuming a that a similar distribution can represent lithologies on magnetic anomaly maps, then log transformation of the magnetic data should serve to normalize the distribution and highlight features having common amplitude.

The log-normal transformation of magnetic data f is defined as [MPBL01]:

$$\mathcal{F}\{f\} = \begin{cases} -\log_{10}(-f) & \text{for} & f < -1 \\ \log_{10}(f) & \text{for} & f > 1 \\ 0 & \text{for} & -1 < f > 1 \end{cases}$$

A multiplying factor (`multfactor`) can be used to increase/decrease the number of data that falls into the condition $-1 < f > 1$, i.e. the number of data that are nulled.

6.2.3 Parameters

Name	Description	Type	Value
mult-factor	Multiplying factor that can be used to increase/decrease the number of data that falls into the condition [-1< f >1].	float	x5, x10, x20, x100...
setnan	Flag to replace values by NaNs instead of zeros.	bool	`True` or `False`
valfilt	Flag to apply filter on the ungridded data values rather than on the gridded data.	bool	`True` or `False`

6.3 Pole reduction

Classic reduction to the pole.

The reduction to the magnetic pole is a way to facilitate magnetic data interpretation and comparison. It simulates the anomaly that would be measured at the north magnetic pole (inclination of the magnetic field is maximum, i.e. vertical) [BLAK96].

Filter applications: *(magnetic) ease anomaly interpretation*

6.3.1 Examples

```
>>> dataset.polereduction(apod=5, inclination=65, declination=0, azimuth=10)
```

6.3.2 Principle

Due to the dipolar nature of the geomagnetic field, magnetic anomalies (if not located at the magnetic poles) are asymmetric with a geometry that depends on the ambient magnetic field inclination (I).

The filter symmetrize the anomalies and place them directly above the source. In other words, the reduction to the pole simulates the anomaly that would be measured at the north magnetic pole. A similar processing (reduction to the equator) is used when data are recored at low magnetic inclinations.

The computation is done in the spectral (frequency) domain using Fast Fourier Transform. For magnetization and ambient field uniform throughout the study area the transformation is given in the spectral domain by [BLAK96]:

$$\mathcal{F}_{RTP} = \mathcal{F}_{TF} \cdot \mathcal{F}$$

where $\mathcal{F}_{RTP}$ is the Fourier Transform of the anomaly reduced to the pole, $\mathcal{F}_{TF}$ is the Fourier Transform of the measured *total field* anomaly and $\mathcal{F}$ is the pole reduction operator defined as:

$$\mathcal{F}_{m,f}\{k_x, k_y\} = \frac{|k|^2}{a_1 k_x^2 + a_2 k_y^2 + a_3 k_x k_y + i|k|(b1 k_x + b2 k_y)}, |k| \neq 0$$

with

$$a_1 = m_z f_z - m_x f_x,$$
$$a_2 = m_z f_z - m_y f_y,$$
$$a_3 = -m_y f_x - m_x f_y,$$
$$b_1 = m_x f_z + m_z f_x,$$
$$b_2 = m_y f_z + m_z f_y,$$

where $m = (m_x, m_y, m_z)$ is the unit-vector in the direction of the total magnetization (remanant + induced) of the source; $f = (f_x, f_y, f_z)$ is the unit-vector in the direction of the ambient field; $|k| = \sqrt{k_x^2 + k_y^2}$ is the radial wavenumber and k_x and k_y are the wavenumber in the x and y-direction respectively.

The remanent magnetization of the source is usually small compared to the induced magnetization and can be neglected. The total magnetization is hence purely inductive so $m = (m_x, m_y, m_z) = f = (f_x, f_y, f_z)$ and the previous equation simplifies in:

$$\mathcal{F}_f\{k_x, k_y\} = \frac{|k|^2}{[|k|f_z + i(k_x f_x + k_y f_y)]^2}, |k| \neq 0$$

By default, the remanent magnetization is neglected in the filter (`incl_mag=None`, `decl_mag=None`).

6.3.3 Algorithm
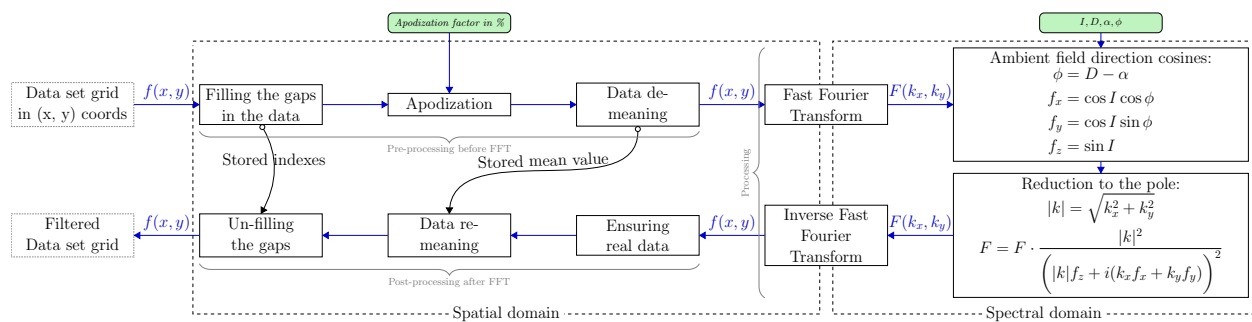
Fig. 6.6: Reduction to the Pole - Reduction to the Pole Algorithm.
Reduction to the Pole algorithm where k_x and k_y are the wavenumbers corresponding to the x and y-direction respectively; I is the ambient magnetic field inclination; D is the ambient magnetic field declination; ϕ is the survey x-axis azimuth.

See *General consideration* for the definition of filling gaps and apodization methods.

6.3.4 Parameters

Name	Description	Type	Value
apod	Apodization factor (in %), to limit Gibbs phenomenon at jump discontinuities.	float	0, 5, 10, 20, 25, ...
inclination	Magnetic inclination (I, in degree). The angle between the horizontal plane and the ambient magnetic field vector (positive downward).	float	65, 45 ...
declination	Magnetic declination (D, in degree). The angle between the geographic ('true') North and Magnetic North (the horizontal projection of the ambient magnetic field vector, positive eastward).	float	
azimuth	Survey profiles azimuth (ϕ, in degree). The angle between the profile direction and the Geographic North (positive east of north).	float	
mag-azimuth	Survey profiles magnetic azimuth ($\alpha = D - \phi$, in degree). The angle between the profile direction and the Magnetic North (positive east of north).	float	0, 10, 30..., None

6.4 Continuation

Upward or downward continuation of potential field data (magnetic or gravimetric).

The filter computes the data that would be measured at an upper (*upward continuation*) or lower (*downward continuation*) survey altitude. The computation is done in the spectral (frequency) domain using the Fast Fourier Transform [BLAK96].

Filter applications: *(magnetic) ease anomaly interpretation, background removal, data smoothing, merging surveys at different altitudes*

6.4.1 Examples

```
>>> continuation(apod=0, configuration='TotalField', distance=+0.5)
```
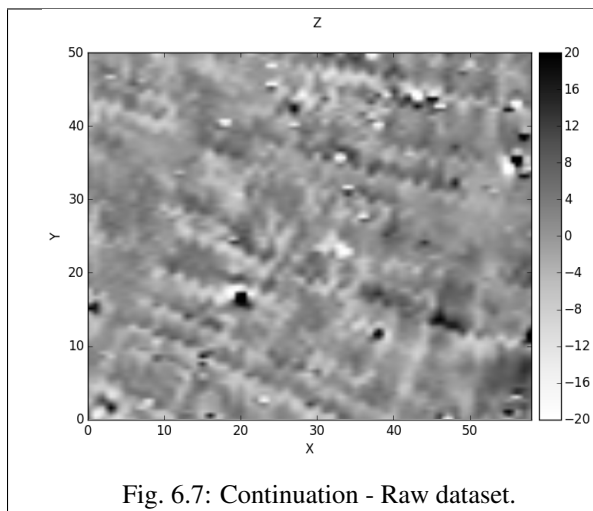

Fig. 6.7: Continuation - Raw dataset.
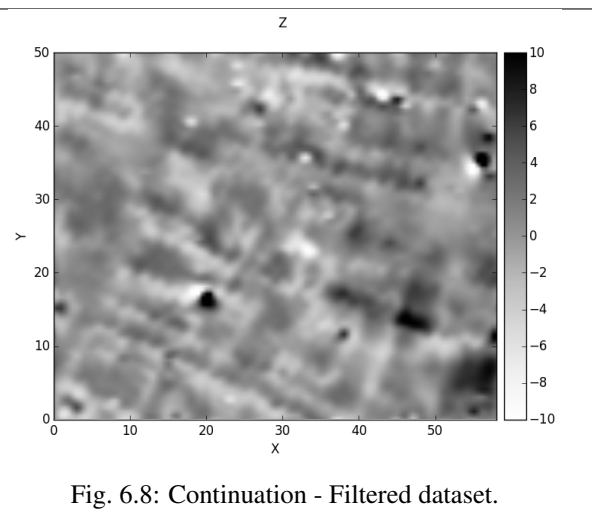
Fig. 6.8: Continuation - Filtered dataset.
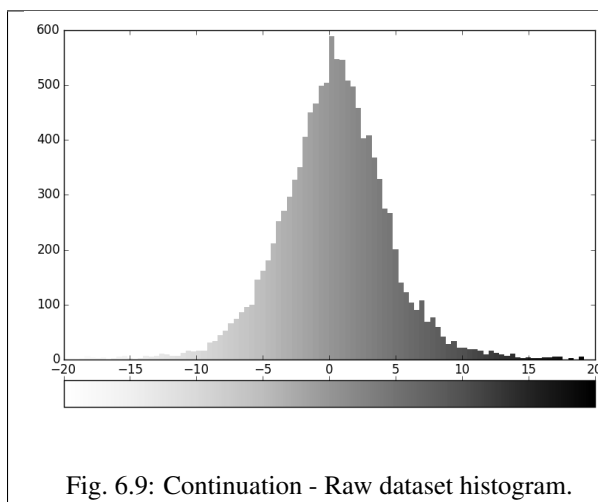
Fig. 6.9: Continuation - Raw dataset histogram.
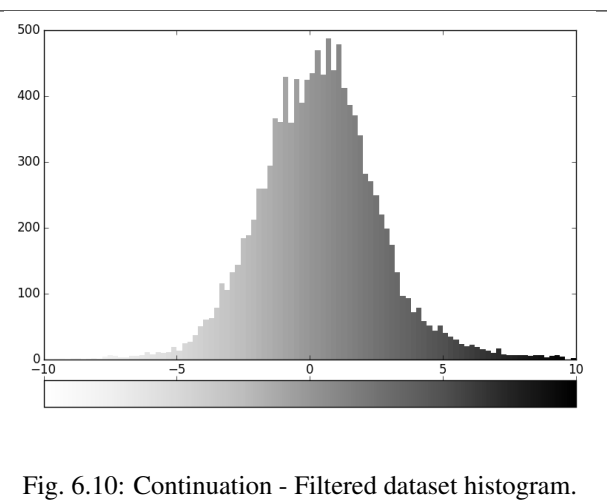
Fig. 6.10: Continuation - Filtered dataset histogram.

6.4.2 Principle

The (upward or downward) continuation transforms the potential field measured at one altitude to the field that would be measured at another altitude, farther (*upward continuation*) or closer (*downward continuation*) from the magnetic sources.

Assuming that all the magnetic sources are located below the observation surface, the continuation at a new observation altitude z of a survey acquired at an original altitude z_0 is given in the spectral domain by [BLAK96]:

$$\mathcal{F}_{\Delta z,k} = \mathcal{F}_{TF} \cdot e^{-\Delta z|k|}$$

where:

- $\mathcal{F}_{TF}$ is the Fourier Transform of the measured data at the original altitude of observation z_0;

- $\mathcal{F}_{\Delta z,k}$ is the Fourier Transform of the anomaly at the new altitude of observation $z = z_0 - \Delta z$;

- $\Delta z = z_0 - z$ is the altitude increase between the original and new altitude of observation;

- $|k| = \sqrt{k_x^2 + k_y^2}$ is the radial wavenumber where k_x and k_y are the wavenumber in the x and y-direction respectively.

The given altitude increase (Δz) is an algebraic value:

- If $\Delta z > 0$, the new altitude of observation is above the original altitude: the operation is an *upward continuation*;

- if $\Delta z < 0$, the new altitude of observation is below the original altitude: the operation is a *downward continuation*.

If the sensor configuration is in total-field vertical gradient, the data can be transformed to total-field data (`totalfieldconversionflag=True`) using $\left(1 - e^{\Delta s|k|}\right)^{-1}$ (see *Sensor configuration conversion*).

The *upward continuation* attenuates anomalies with respect to the wavelength in way that accentuates anomalies caused by deep sources and attenuates at the anomalies caused by shallow sources. It is hence a smoothing operator.

The *downward continuation* accentuates the shallowest components. It reduces spread of anomalies and corrects anomalies coalescences. It is useful to discriminates the number of body source at the origin of a one big anomaly. It is an unsmoothing operator that is instable as small changes in the data can cause large and unrealistic variations so it is to be used with caution. Low-pass filtering before the *downward continuation* can be a solution to increase the filter stability.

Note: Algorithm unstability

Unlike the *upward continuation*, the *downward continuation* is unstable process as it is an unsmoothing process that tends to accentuate small changes in the shallow components. So "Any errors present and perhaps undetected in the measured data may appear in the calculated field as large and unrealistic variations." [BLAK96]

6.4.3 Algorithm

See *General consideration* for the definition of filling gaps and apodization methods.
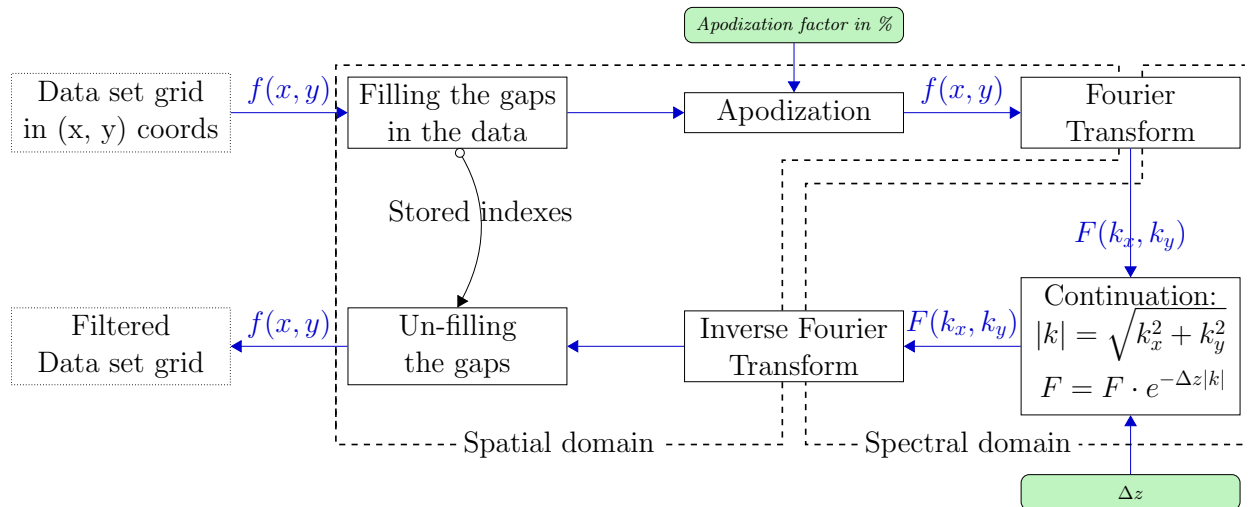
Fig. 6.11: Continuation - Continuation Algorithm.

6.4.4 Parameters

Name	Description	Type	Value
apod	Apodization factor (%), to limit Gibbs phenomenon at jump discontinuities.	float	0, 5, 10, 20, 25, ...
distance	Continuation distance (in m). Positive for upward continuation and negative for downward continuation.	float	0.5, 2, 10, ...
totalfieldconversionflag	Flag to proceed to the conversion to total-field data after continuation	bool	`True`, `False`
separation	Sensor separation if `totalfieldconversionflag` is `True`.	float	0.7, 1, ...

6.5 Analytic signal

Computes the 3-D Analytic Signal.

The Analytic Signal (also known as the total gradient magnitude or energy envelope) is a way to ease magnetic source characterization independently from the direction of its magnetization.

Filter applications: *magnetic source depth estimation*

6.5.1 Examples

```
>>> dataset.analyticsignal(apod=0)
```
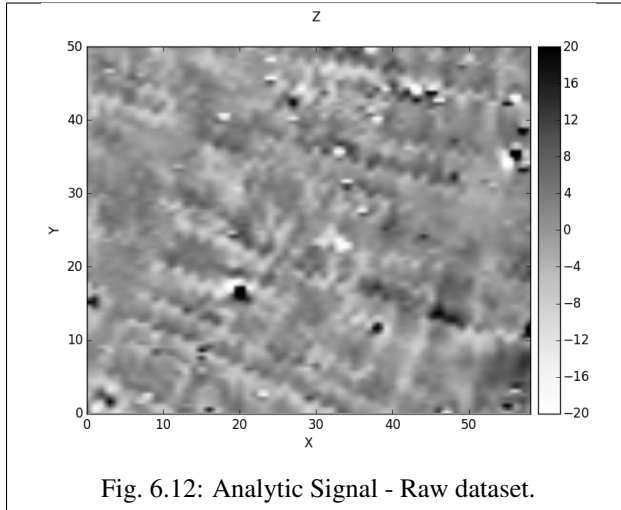
Fig. 6.12: Analytic Signal - Raw dataset.
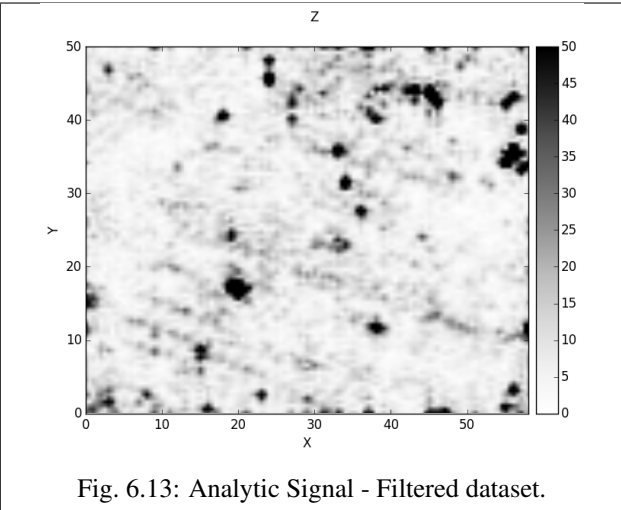
Fig. 6.13: Analytic Signal - Filtered dataset.
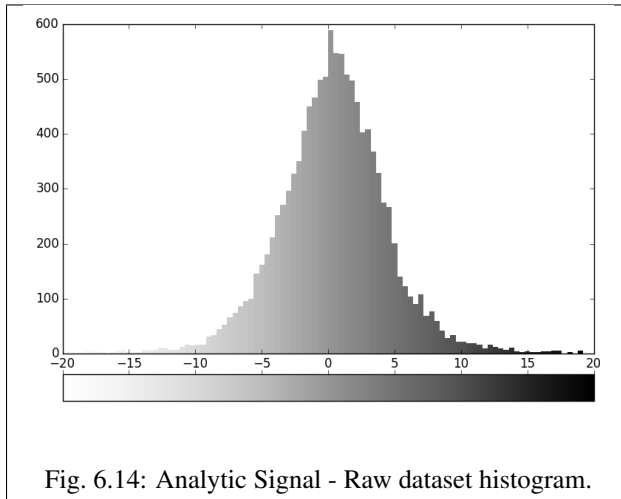
Fig. 6.14: Analytic Signal - Raw dataset histogram.
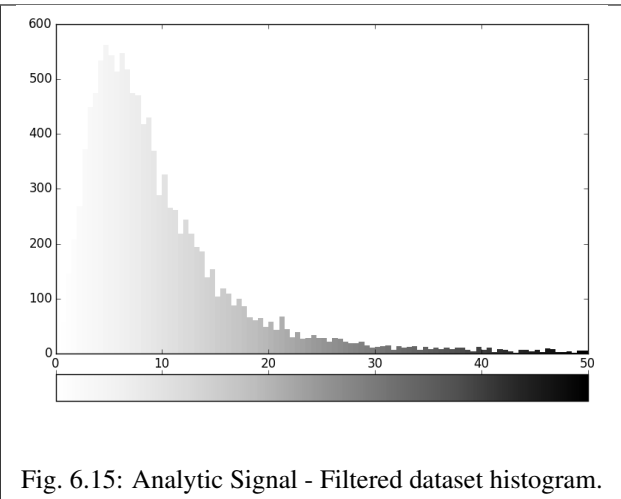
Fig. 6.15: Analytic Signal - Filtered dataset histogram.

6.5.2 Principle

For a magnetization and an ambient field uniform throughout the study area, the amplitude of the analytic signal (or total gradient magnitude or energy envelope) of a **potential field** anomaly T is given by [RoVP92]:

$$|A(x, y, z)| = \sqrt{\left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial y}\right)^2 + \left(\frac{\partial T}{\partial z}\right)^2}$$

The directional derivatives are computes in the spectral domain (see *General consideration*) and transformed back to the spatial domain using an inverse Fourier Transform to computed the analytic signal amplitude.

6.5.3 Algorithm

See *General consideration* for the definition of filling gaps and apodization methods.
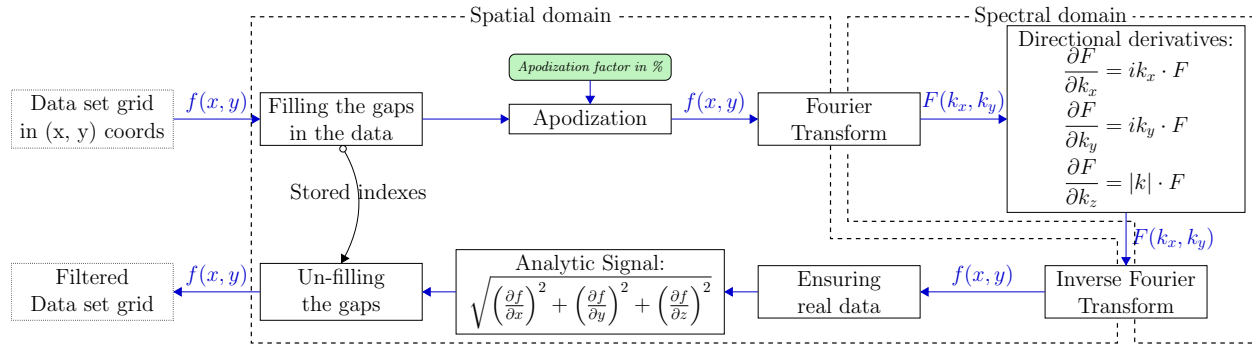
Fig. 6.16: Analytic Signal - Analytic Signal Algorithm.

6.5.4 Parameters

Name	Description	Type	Value
apod	Apodization factor (%), to limit Gibbs phenomenon at jump discontinuities.	float	0, 5, 10, 20, 25, . . .

6.6 Euler deconvolution

Classic Euler deconvolution.

Euler deconvolution is a method to estimate the depth of magnetic sources that do not required reduced-to-the-pole data.

6.6.1 Examples

Manual target picking

```
>>> target_extents = [[15,25,10,25], [35,42,8,15]]  # [[xmin, xmax, ymin,
↪xmax],...]
>>> results = dataset.eulerdeconvolution(apod=0, structind=3, window=target_
↪extents)
```
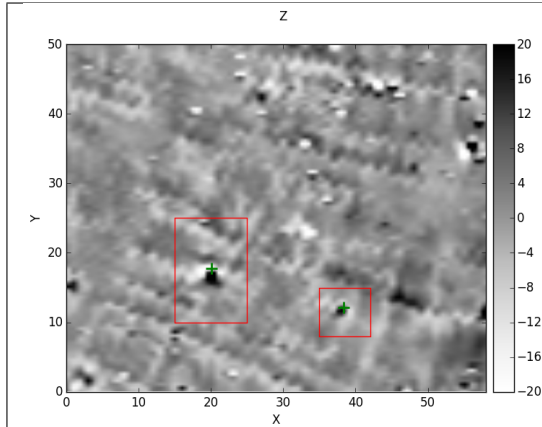
Fig. 6.17: Euler Deconvolution - Manual target picking.
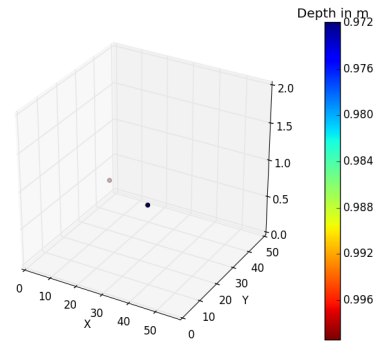
Fig. 6.18: Euler Deconvolution - Manual target picking

Sliding window target picking

```
>>> # Sliding window of xstep*xstep samples
>>> results = dataset.eulerdeconvolution(apod=0, structind=3, xstep=10)
```
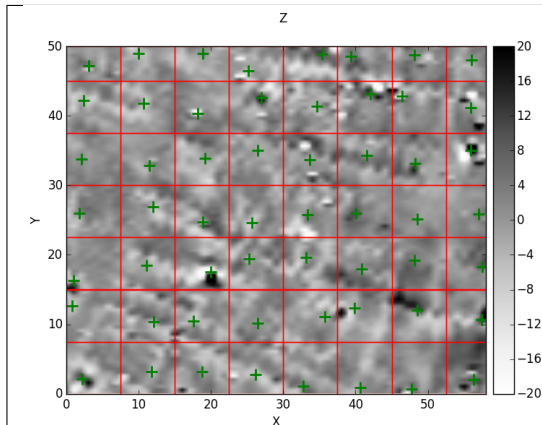

Fig. 6.19: Euler Deconvolution - Manual target picking.
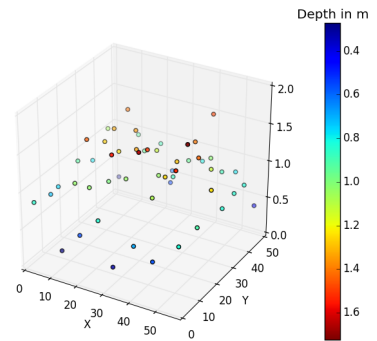
Fig. 6.20: Euler Deconvolution - Manual target picking

Displaying results

The eulerdeconvolution() method returns the results of the Euler deconvolution as a list containing the source position, the associated structural index, the residual of the least-square estimation and the used dataset sub-window extent:

```
>>> print(results[0])
[20.078, 17.80, 0.99, 3, 0.94343, 15.0, 25.0, 10.0, 25.0]
```

The sub-window extents and source positions can be display onto the dataset using the following commands sequence:

```
>>> # Importing the sub-window-to-rectangle conversion tool
>>> import geophpy.plotting.plot as gplt
```

```
>>> # Extracting source positions and sub-windows extents
>>> sources = np.asarray(results)[:,:2].tolist()
>>> window_extents = np.asarray(results)[:,5:].tolist()
```

```
>>> # Converting sub-windows extents to rectangles
>>> rects = gplt.extents2rectangles(window_extents)
```

```
>>> # Plotting results
>>> dataset.plot(plottype='2D-SURFACE', rects=rects, points=sources)
```

6.6.2 Principle

Euler deconvolution is a method to estimate position and depth of magnetic sources. It is based on the assumption that magnetic field *is homogeneous of degree N*, so that the total magnetic field and its originating magnetic source are related by the following equation [RAGM90]:

$$N(B - T) = (x - x_0)\frac{\partial T}{\partial x} + (y - y_0)\frac{\partial T}{\partial y} + (z - z_0)\frac{\partial T}{\partial z}$$

where (x_0, y_0, z_0) is the position of a source; T is the total-field detected at (x, y, z); B is the regional value of the field and N is a non-negative integer known as the structural index (or degree of homogeneity).

For a set of n observation points of the total-field, the equation can be reformulated as a linear system of equations and expressed in matrix form as:

$$\begin{pmatrix} \frac{\partial T_1}{\partial x} & \frac{\partial T_1}{\partial y} & \frac{\partial T_1}{\partial z} & N \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial T_i}{\partial x} & \frac{\partial T_i}{\partial y} & \frac{\partial T_i}{\partial z} & N \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial T_n}{\partial x} & \frac{\partial T_n}{\partial y} & \frac{\partial T_n}{\partial z} & N \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ B \end{pmatrix} = \begin{pmatrix} x_1 \frac{\partial T_1}{\partial x} + y_1 \frac{\partial T_1}{\partial y} + z_1 \frac{\partial T_1}{\partial z} + NT_1 \\ \vdots \\ x_n \frac{\partial T_i}{\partial x} + y_i \frac{\partial T_i}{\partial y} + z_i \frac{\partial T_i}{\partial z} + NT_i \\ \vdots \\ x_n \frac{\partial T_n}{\partial x} + y_n \frac{\partial T_n}{\partial y} + z_n \frac{\partial T_n}{\partial z} + NT_n \end{pmatrix}$$

$$i = 1, ..., n$$

where $T_i = T(x_i, y_i, z_i)$ is the *ith* observation of the total-field anomaly at the coordinates (x_i, y_i, z_i).

This system of has the form $\mathbf{Ax} = \mathbf{b}$ and can be solved in least-square sense in the overdetermined case, leading to least-square estimates of (x_0, y_0, z_0, B).

Directional derivatives computation

The directional derivatives are computes in the spectral domain (see *General consideration*) and transformed back to the spatial domain using an inverse Fourier Transform.

Structural Index (SI)

The structural index is a measure of the rate of change with distance of a field. It is an integer value that depends on the source model geometry and is "not a tuning parameter" [ReEW14]. Valid structural index values are referenced in the following table.

Source Model		SI
Point, sphere		3
Line, cylinder, thin bed fault		2
Thin sheet edge, thin sill, thin dike		1
Contact of infinite depth extent		0

A wrong value of the SI will lead to errors in the depth estimation of the source: a too high value will yield over-estimated depths and vice versa.

Note: Structural Index estimation

Although it is not recommended, the structural index can be estimated in least-square sense as a parameter of the linear system by setting `structind=None`.

For now, this estimation may results in non-realistic (non integer) values for the structural index. In the future it will be constrained to select only allowed (integer) values.

Set of observation points

The set of observation points can be:

- specified manually by giving a set of windows extent (xmin, xmax, ymin and xmax for each window of interest),

- or automatically by using a sliding windows of a given size (`xstep * ystep`).

6.6.3 Algorithm

. . . TBD . . .

See *General consideration* for the definition of filling gaps and apodization methods.

6.6.4 Parameters

Name	Description	Type	Value
apod	Apodization factor (%), to limit Gibbs phenomenon at jump discontinuities.	float	0, 5, 10, 20, 25, . . .
struc-tuind	Structural Index, depends on the source geometry.	int	0, 1, 2, 3 or `None`
win-dows	List containing the satial extent of each sub-window to be considered for the deconvolution	list (of float)	[15,25,10,25], [35,42,8,15], . . .] or `None`
xstep	Size, in number of sample, of the sub-windows in the x-direction. Only if `windows=None`.	float	10, 33, . . . or `None`
ystep	Size, in number of sample, of the sub-windows in the y-direction. Only if `windows=None`.	float	10, 33, . . . or `None`

6.7 Sensor configuration conversion

Conversion between the different sensor's configurations.

Magnetic data are all derived from the same potential and thereby contains in the same information, making theoretical conversion from one sensor configuration to another is possible.

6.7.1 Examples

6.7.2 Principle

Du to the "potential field nature" of the magnetic field, the same information is theoretically contained in the measure of the total-field anomaly, the total-field gradient and the vertical component of the anomaly. Conversion from one sensor configuration to another is theoretically possible even in practice the (inevitable) measurement noise can make it difficult.

Sensor's configurations

Different sensor's configurations exist to conduct magnetic surveys. Three most commonly used are the *total-field*, the *total-field vertical gradient* and the *fluxgate* sensor's configurations :

- **Total-field magnetometers** measure the magnitude of the total magnetic field regardless of the magnetic vector direction. They give a *scalar measure* of the total strength of an ambient magnetic field at any given point and are hence sometimes referred to as *scalar magnetometers* [AsGA08].

- **Total-field gradiometers** simply refers to *total-field magnetometers* in *vertical gradient configuration*, *i.e* two total-field sensors one on top of the other.

- **Fluxgate magnetometers** measure the component of the ambient magnetic field in a particular direction (the sensor's axis only) and are hence sometimes referred to as *vector magnetometers* [AsGA08]. *Fluxgate gradiometers* are composed of two sensors one on top of the other and measure directly the difference of the vertical component of the ambient magnetic field.

Conversion between configurations

A simple linear transformation combining a *Continuation* and a subtraction allow the conversion between the different sensor's configuration. For an ambient field uniform throughout the study area the transformation of the different sensor's configurations is defined in the spectral domain by [TaDD97]:

Total-field and total-field vertical gradient Conversion of a measure obtained using a *total-field magnetometer* to the measure that would be obtained using a *total-field magnetometer in vertical gradient configuration* (i.e. with two sensors, one on top of the ohers) is defined in the spectral domain as:

$$e^{-\Delta h|k|} - e^{-(\Delta h + \Delta s)|k|}$$

The reverse transformation (from total-field vertical gradient to total-field) can simply be computed using the inverse expression:

$$\left(e^{-\Delta h|k|} - e^{-(\Delta h + \Delta s)|k|}\right)^{-1}$$

Total-field vertical gradient and fluxgate Conversion of a measure obtained using a *total-field magnetometer in vertical gradient configuration* to the measure that would be obtained using a *fluxgate gradiometer* is defined in the spectral domain as:

$$\frac{|k|}{|k|f_y + i(k_x f_x + k_y f_y)}$$

The reverse transformation (from fluxgate vertical gradient to total-field vertical gradient) can simply be computed using the inverse expression:

$$\left(\frac{|k|}{|k|f_y + i(k_x f_x + k_y f_y)}\right)^{-1}$$

Total-field and fluxgate Conversion of a measure obtained using a *total-field magnetometer* to the measure that would be obtained using a *fluxgate gradiometer* is defined in the spectral domain as the combination of the previous transformation :

$$\left(e^{-\Delta h|k|} - e^{-(\Delta h+\Delta s)|k|}\right) \cdot \frac{|k|}{|k|f_y + i(k_x f_x + k_y f_y)}$$

The reverse transformation (from fluxgate vertical gradient to total-field) can simply be computed using the inverse expression:

$$\left(\left(e^{-\Delta h|k|} - e^{-(\Delta h+\Delta s)|k|}\right) \cdot \frac{|k|}{|k|f_y + i(k_x f_x + k_y f_y)}\right)^{-1}$$

Where

- Δh is the difference of altitude between the bottom sensors in the initial and final configuration;

- Δs is the difference of altitude between top and bottom sensors in gradient configuration;

- $|k| = \sqrt{k_x^2 + k_y^2}$ is the radial wavenumber with k_x and k_y are the wavenumbers in the x and y-direction respectively;

- I is the ambient magnetic field inclination;

- D is the ambient magnetic field declination;

- ϕ is the survey x-axis azimuth;

- $f_x = \cos(I) \cdot cos(D - \phi)$ is the unit-vector in the x-direction of the ambient field;

- $f_y = \cos(I) \cdot sin(D - \phi)$ is the unit-vector in the y-direction of the ambient field;

- $f_z = \sin(I)$ is the unit-vector in the z-direction of the ambient field.

6.7.3 Algorithm

See *General consideration* for the definition of filling gaps and apodization methods.

6.7.4 Parameters

6.8 Equivalent Susceptibility

6.8.1 Examples

... TBD ...

6.8.2 Principle

... TBD ...

6.8.3 Algorithm

... TBD ...

See *General consideration* for the definition of filling gaps and apodization methods.

6.8.4 Parameters

... TBD ...

GEOGRAPHIC POSITIONNING

All geographic position information are manage via a *GeoPosSet* class.

You can open a file containing the geographic coordinates of the geophysical survey corresponding to a *DataSet* object from both :

- an ascii file** (*.csv* for instance)
- or a shapefile (*.shp*)

Reading positions from file

To open your a *GeoPosSet* from a file (ASCII or shapefile) use:

```python
>>> from geophpy.geoposset import GeoPosSet
```

```python
>>> # file type determined from extension
>>> gpos = GeoPosSet.from_file('GCPs.txt')
```

```python
>>> # file type forced manually
>>> gpos = GeoPosSet.from_file('GCPs.shp', filetype='shapefile', )
```

Retrieve position

Retrieve list of Ground Control Points via:

```python
>>> gcp_list = gpos.points_list
>>> # or
>>> gcp_list = gpos.GCPs()
```

```python
>>> print(gcp_list)
>>> [[0, 32.52, 34.70], [1, 32.52, 34.70]]  # with [num, x or lon, y or lat]
```

Plot GCPs

Plot the Ground Control Points:

```python
>>> fig = gpos.plot(long_label=False)
>>> fig.show()
```
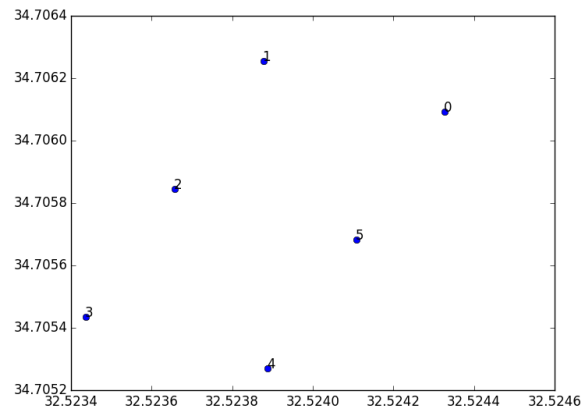
Saving GCPs

Ground Control Points can be saved as an ascii file or a kml file:

```
>>> # saving as an ascii file
>>> gpos.to_ascii("MyGCPs.dat", delimiter=',')
```

To be saved as a kml file, the coordinates must be converted into lat,long first. If the original geographic system is UTM you can directly do:

```
>>> # Converting UTM to WGS84
>>> import geophpy.geoposset as pset
>>> num, east, north, xlocal, ylocal = gpos.GCPs().T
>>> lt, lg = pset.utm_to_wgs84(east, north, 32,'T')
>>> gcp_wgs84 = np.stack((num, xlocal, ylocal, lt, lg)).T
```

```
>>> # saving as kml
>>> gpos_conv = pset.GeoPosSet(refsystem='WGS84', points_list=gcp_wgs84)
>>> gpos.to_kml("ConvGCPs.kml")
```


Dataset georeferencing

Dataset georeferencing is possible with at less 4 points.

```
>>> # Georeferencing ungridded values
>>> dataset.setgeoref('UTM', gpos.points_list, 'T', 32)
```

```
>>> # Georeferencing gridded values
>>> dataset.interpolate(x_step=0.5, y_step=0.25)
>>> dataset.setgeoref('UTM', gpos.points_list, 'T', 32)
```

Plotting georeferenced data position (example for ungridded values)

```
>>> import geophpy.dataset as dset
>>> import geophpy.geoposset as pset
```

```
>>> # Reading files
>>> success, dataset = dset.DataSet.from_file('Mag_ex1.dat')
>>> success, gpos = pset.GeoPosSet.from_file('GPS_ex1.csv')
```

```
>>> # Displaying GCps
>>> gpos.plot(long_label=True)
```
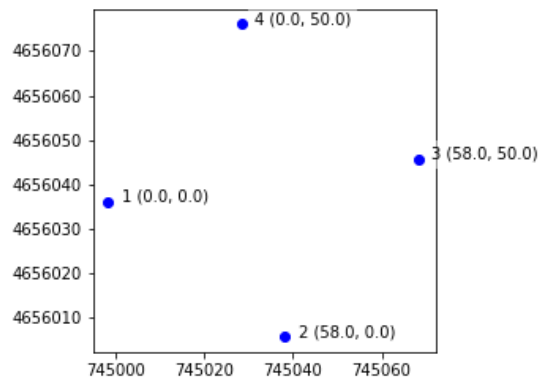

Fig. 7.1: Georeferencing - Ground Control Points.

```
>>> # Georeferencing
>>> dataset.setgeoref(gpos.refsystem, gpos.points_list, 'T', 32)
```

```
>>> # Displaying in local system
>>> box = np.stack(dataset.get_boundingbox())  # bounding box corners
>>> x, y = dataset.get_xyvalues()  # data sample position
```

```
>>> box_gcp = gpos.points_list.T[1:3].T  # georef bounding box corners
>>> east, north = dataset.get_georef_xyvalues() # data sample georef
→position
```

```
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(x, y, 'r.', markersize=0.25) # or east, north
>>> ax.plot(box.T[0], box.T[1], 'bo') # or box_gcp.T[0], box_gcp.T[1]
>>> ax.set_aspect('equal')
```

```
>>> # adding GCPs label
>>> for point in  box:
>>>     strg = '(%s, %s)' % (point[0], point[1])
>>>     dx, dy = 3, 0
>>>     plt.text(point[0]+dx, point[1]+dy, strg,
>>>             bbox=dict(fc='white',ec='none', alpha=0.5))
```
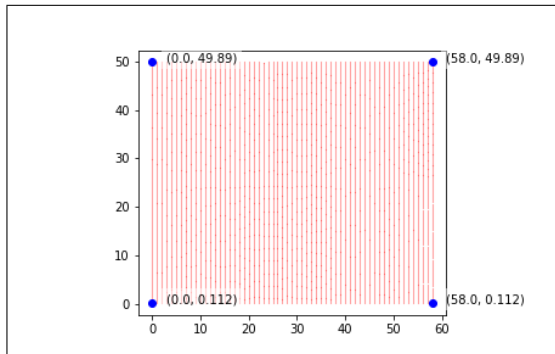

Fig. 7.2: Georeferencing - Dataset in local system.
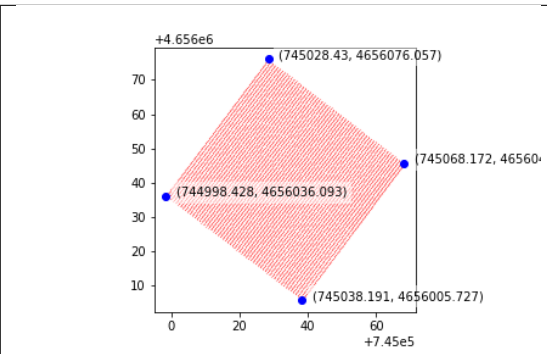
Fig. 7.3: Georeferencing - Dataset in geographic system.

With the data set georeferenced, it is possible to export the dataset as a kml file:

```
>>> dataset.to_kml('2D-SURFACE', 'gray_r', "prospection.kml",
    cmmin=-10, cmmax=10, dpi=600)
```


Exporting the data set as a raster in a SIG application (as ArcGis, QGis, Grass, ...) is possible with severals picture file format ('jpg', 'png', 'tiff'):

```
>>> dataset.to_raster('2D-SURFACE', 'gray_r', "prospection.png",
    cmmin=-10, cmmax=10, dpi=600)
```

A world file containing positioning informations of the raster is created ('jgw' for JPG, 'pgw' dor.png, and 'tfw' for TIFF picture format) with:

Line 1: A: pixel size in the x-direction in map units/pixel

Line 2: D: rotation about y-axis

Line 3: B: rotation about x-axis

Line 4: E: pixel size in the y-direction in map units, almost always negative[3]

Line 5: C: x-coordinate of the center of the upper left pixel

Line 6: F: y-coordinate of the center of the upper left pixel

Example:

0.0062202177595

-0.0190627320737

0.0131914192417

0.00860610262817

660197.8178

3599813.97056

ASCII GCPs file format

If the *geographic reference system* is known, it must be written on the first line. The other lines contain the *point number* followed by its *GPS coordinates* (*Longitude* and *Latitude* or *Easting* and *Northing*) and eventually the corresponding local *X*, *Y* -coordinates.

```
>>> # WGS84 file without local coordinates
>>> # (delimiter is tabulation)
WGS84
1    66.84617533    37.74956917
2    66.84649517    37.7489535
3    66.8472475     37.74972867
4    66.84689417    37.7491385
5    66.84691867    37.7491025
...
```

```
>>> # UTM file can also contain the corresponding local coordinates
>>> # (delimiter is tabulation)
UTM
1    745038.191      4656005.727      150     0
2    745068.172      4656045.663      150     50
3    745028.43       4656076.057      100     50
4    744988.466      4656105.978      50      50
5    744998.428      4656036.093      100     0
...
```

```
>>> # UTM file with zone letter and number
>>> # (delimiter is tabulation)
UTM         L       32
1    745038.191      4656005.727      150     0
2    745068.172      4656045.663      150     50
3    745028.43       4656076.057      100     50
4    744988.466      4656105.978      50      50
5    744998.428      4656036.093      100     0
...
```

```
>>> # Unknown geographic system with missing local position
>>> # (delimiter is ";")
1;745038.191;4656005.727;150;0
2;745068.172;4656045.663;150;50
3;745028.43;4656076.057;;            # missing local position
4;744988.466;4656105.978;50;50
5;744998.428;4656036.093;100;0
...
```

USING A GUI

All plots in **GeophPy** are actually made using Matplotlib.

To embed Matplotlib's plots in your own GUI, you have to select the proper backend:

```
>>> import matplotlib
>>> #for Qt4 and Pyside
>>> matplotlib.use('Qt4Agg')
```

```
>>> #for Qt5 and Pyside2
>>> matplotlib.use('Qt5Agg')
```

```
>>> #for Tkinter
>>> matplotlib.use('TkAgg')
```

As specified in the SciPy Cookbook on PySide and Matplotlib, "in case of problems for PySide, try to change the rcParam entry "backend.qt4" to "PySide" (e.g. by in the matplotlibrc file):

```
>>> matplotlib.rcParams['backend.qt4']='PySide'
```

Warning: `rcParams` has been depreciated in recent matplotlib versions

---

Note:

- in Windows environment, this file `matplotlibrc` is in the "C:\PythonXY\Lib\site-packages\matplotlib\mpl-data" directory.
- in Linux environment, it is in the "/etc" directory.

You can also create widgets to embed Matplotlib's plot : see Embedding Matplotlib in Qt and Embedding Matplotlib in Tk for complete examples.

You can also plot data in a windows with several color maps:

```
>>> from geophpy.dataset import *
>>>                 # to get the list of available color maps
>>> list = colormap_getlist()
>>> first = True                        # first plot
>>> fig = None                          # no previous figure
>>> cmap = None                         # no previous color map
>>> success, dataset = DataSet.from_file("DE11.dat", delimiter=' ',
```

(continues on next page)

```
    z_colnum=5)
>>> if (success == True):                 # if file opened
>>>                # for each color map name in the list
>>>     for colormapname in list :
>>>         fig, cmap = dataset.plot('2D-SURFACE', 'gist_rainbow',
            dpi=600, axisdisplay=True, cmapdisplay=True, cmmin=-10,
            cmmax=10)
>>>         if (first == True):           # if first plot
>>>             fig.show()                # displays figure windows
>>>             first = False             # one time only
>>>                # updates the plot in the figure windows
>>>         p.draw()
>>>                # removes it to display the next
>>>         cmap.remove()
>>>                # waits 3 seconds before display the plot
>>>                # with the next color map
>>>         time.sleep(3)
```

API

9.1 High level processing functions

The calling protocol of these functions is described in the end of this document (see *High level API*) but about the detailed source code of is available in this section.

9.1.1 geophpy.processing.general

DataSet Object general processing routines.

> **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.
>
> **license** GNU GPL v3.

`geophpy.processing.general.`**`threshold`**(*dataset*, *setmin=None*, *setmax=None*, *setmed=False*, *setnan=False*, *valfilt=False*)

> Dataset thresholding
>
> cf. *threshold()*

`geophpy.processing.general.`**`peakfilt`**(*dataset*, *method='hampel'*, *halfwidth=5*, *threshold=3*, *mode='relative'*, *setnan=False*, *valfilt=False*)

> Datset peak filtering
>
> cf. *peakfilt()*

`geophpy.processing.general.`**`medianfilt`**(*dataset*, *nx=3*, *ny=3*, *percent=0*, *gap=0*, *valfilt=False*)

> 2-D median filter
>
> cf. *medianfilt()*

`geophpy.processing.general.`**`festoonfilt`**(*dataset*, *method='Crosscorr'*, *shift=0*, *corrmin=0.4*, *uniformshift=False*, *setmin=None*, *setmax=None*, *valfilt=False*)

> Destaggering filter
>
> cf. *festoonfilt()*

`geophpy.processing.general.`**`detrend`**(*dataset*, *order=1*, *setmin=None*, *setmax=None*, *valfilt=False*)

> Dataset detreding using a constant value, a linear or polynomial fit.
>
> cf. `detrend()`

geophpy.processing.general.**regtrend**(*dataset, nx=3, ny=3, method='relative', component='local', loctrendout=None, regtrendout=None, valfilt=False*)

    cf. dataset.py

geophpy.processing.general.**wallisfilt**(*dataset, nx=11, ny=11, targmean=125, targstdev=50, setgain=8, limitstdev=25, edgefactor=0.1, valfilt=False*)

    cf. *wallisfilt()*

geophpy.processing.general.**ploughfilt**(*dataset, apod=0, azimuth=0, cutoff=100, width=2, valfilt=False*)

    cf. *ploughfilt()*

geophpy.processing.general.**zeromeanprofile**(*dataset, setvar='median', setmin=None, setmax=None, valfilt=False*)

    Zero-traverse filter

    cf. *zeromeanprofile()*

geophpy.processing.general.**destripecon**(*dataset, Nprof=0, setmin=None, setmax=None, method='additive', reference='mean', config='mono', valfilt=False*)

    Destripe dataset using a constant value.

    cf. *destripecon()*

geophpy.processing.general.**destripecub**(*dataset, Nprof=0, setmin=None, setmax=None, Ndeg=3, valfilt=False*)

    Destripe dataset using a polynomial fit.

    cf. *destripecub()*

9.1.2 geophpy.processing.magnetism

DataSet Object general magnetism processing routines.

    copyright Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

    license GNU GPL v3.

geophpy.processing.magnetism.**logtransform**(*dataset, multfactor=5, setnan=True, valfilt=False*)

    Apply a logarihtmic transformation to the dataset.

    cf. *logtransform()*

geophpy.processing.magnetism.**polereduction**(*dataset, apod=0, inclination=65, declination=0, azimuth=0, magazimuth=None, incl_magn=None, decl_magn=None*)

    Dataset Reduction to the magnetic Pole.

    cf. *polereduction()*

geophpy.processing.magnetism.**continuation**(*dataset, apod=0, distance=2, totalfieldconversionflag=False, separation=0.7*)

    Dataset continuation (upward/downward).

    cf. *continuation()*

geophpy.processing.magnetism.**eulerdeconvolution**(*dataset*, *apod=0*, *structind=None*, *windows=None*, *xstep=None*, *ystep=None*)

Classic Euler deconvolution.

cf. `eulerdeconvolution()`

geophpy.processing.magnetism.**analyticsignal**(*dataset*, *apod=0*)

Dataset Analytic Signal.

cf. *analyticsignal()*

geophpy.processing.magnetism.**magconfigconversion**(*dataset*, *fromconfig*, *toconfig*, *apod=0*, *FromBottomSensorAlt=0.3*, *FromTopSensorAlt=1.0*, *ToBottomSensorAlt=0.3*, *ToTopSensorAlt=1.0*, *inclination=65*, *declination=0*, *azimuth=0*, *magazimuth=None*)

Conversion between the different sensors configurations.

cf. *magconfigconversion()*

geophpy.processing.magnetism.**susceptibility**(*dataset*, *prosptech*, *apod=0*, *downsensoraltitude=0.3*, *upsensoraltitude=1.0*, *calculationdepth=0.0*, *stratumthickness=1.0*, *inclineangle=65*, *alphaangle=0*)

Dataset magnetic susceptibilty of the equivalent stratum.

cf. `susceptibility()`

9.1.3 geophpy.operation.general

DataSet Object general operations routines.

> **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

> **license** GNU GPL v3.

geophpy.operation.general.**apodisation2d**(*val*, *apodisation_factor*)

2D apodisation, to reduce side effects

Parameters :

> **Val** 2-Dimension array

> **Apodisation_factor** apodisation factor in percent (0-25)

Returns :

- apodisation pixels number in x direction

- apodisation pixels number in y direction

- enlarged array after apodisation

9.2 High level plotting functions

The calling protocol of these functions is described in the end of this document (see *High level API*) but about the detailed source code of is available in this section.

9.2.1 geophpy.plotting.histo

Histogram Plot Management.

> **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

> **license** GNU GPL v3.

`geophpy.plotting.histo.`**`getlimits`**(*self*, *valfilt=False*)
> Get limits values of histogram.

`geophpy.plotting.histo.`**`plot`**(*dataset*, ***kwargs*)
> Plot the dataset histogram.

> cf. *histo_plot()*

9.2.2 geophpy.plotting.correlation

Module regrouping dataset correlation plots functions.

> **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

> **license** GNU GPL v3.

`geophpy.plotting.correlation.`**`plotmap`**(*dataset*, *fig=None*, *filename=None*, *method='Crosscorr'*, *dpi=None*, *transparent=False*, *showenvelope=False*, *cmapname='jet'*, *cmapdisplay=True*)
> Plot data profile-to-profile correlation map.

`geophpy.plotting.correlation.`**`plotsum`**(*dataset*, *fig=None*, *filename=None*, *method='Crosscorr'*, *dpi=None*, *transparent=False*, *showenvelope=True*, *showfit=True*)
> Plot the dataset profile-to-profile correlation sum.

9.2.3 geophpy.plotting.destrip

Destriping Mean Cross-Track Plot Management module.

> **copyright** Copyright 2017-2019 Lionel Darras, Quentin Vitale and contributors, see AUTHORS.

> **license** GNU GPL v3.

`geophpy.plotting.destriping.`**`plot_mean_track`**(*dataset*, *fig=None*, *filename=None*, *Nprof='all'*, *setmin=None*, *setmax=None*, *method='additive'*, *reference='mean'*, *config='mono'*, *Ndeg=None*, *plotflag='raw'*, *dpi=None*, *transparent=False*)
> Plotting the mean cross-track (mean of each profile).

> cf. *meantrack_plot()*

9.2.4 geophpy.plotting.spectral

Module regrouping Fourier transform plotting functions.

> **copyright** Copyright 2018-2019 Lionel Darras, Quentin Vitale and contributors, see AUTHORS.

> **license** GNU GPL v3.

`geophpy.plotting.spectral.`**`getspectrum_plottype_list`**`()`
>   Return the list of available spectrum plot types.

`geophpy.plotting.spectral.`**`plot_directional_filter`**`(dataset, cmapname=None, cre-`
versed=False, fig=None, file-
name=None, dpi=None, cmapdis-
play=True, axisdisplay=True,
paramdisplay=False, cutoff=100,
azimuth=0, width=2)
>   Plot 2-D directional filter.

`geophpy.plotting.spectral.`**`plotmap`**`(dataset, fig=None, plottype='magnitude', filename=None,`
dpi=None, transparent=False, logscale=False, cmapdis-
play=True, axisdisplay=True, rects=None, lines=None)
>   Plot Dataset 2-D Fourier Transform.

9.2.5 geophpy.plotting.plot

Module regrouping map plotting functions.

>   **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

>   **license** GNU GPL v3.

`geophpy.plotting.plot.`**`plot`**`(dataset, plottype, cmapname, creversed=False, fig=None, file-`
name=None, cmmin=None, cmmax=None, interpolation='bilinear',
levels=None, cmapdisplay=True, axisdisplay=True, labeldis-
play=False, pointsdisplay=False, dpi=None, transparent=False,
logscale=False, rects=None, points=None, marker='+', marker-
size=None)
>   Dataset display.

>   cf. *:meth:'~geophpy.dataset.DataSet.plot*

`geophpy.plotting.plot.`**`extents2rectangles`**`(extentlist)`
>   Convert a list of rectangle extent coordinates to a list of bottom left corner width and height coordinates.

9.3 High level API

9.3.1 GeophPy.dataset

DataSet Object constructor and methods.

>   **copyright** Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

>   **license** GNU GPL v3.

`geophpy.dataset.`**`getlinesfrom_file`**`(filename, fileformat=None, delimiter='\t', skipini-`
tialspace=True, skiprowsnb=0, rowsnb=1)
>   Reads lines in a file.

>   Parameters :

>>   **Fileformat** file format

> **Filename** file name with extension to read, "test.dat" for example.
>
> **Delimiter** delimiter between fields, tabulation by default.
>
> **Skipinitialspace** if True, considers severals delimiters as only one : " " as ' '.
>
> **Skiprowsnb** number of rows to skip to get lines.
>
> **Rowsnb** number of the rows to read, 1 by default.

> Returns:
>
> > **Colsnb** number of columns in all rows, 0 if rows have different number of columns
> >
> > **Rows** rows.

geophpy.dataset.**fileformat_getlist**()
> Get list of format files availables

> Returns: list of file formats availables, ['ascii', ...]

geophpy.dataset.**plottype_getlist**()
> Get list of plot type availables

> Returns : list of plot type availables, ['2D_SURFACE', '2D_CONTOUR', ...]

geophpy.dataset.**interpolation_getlist**()
> Get list of interpolation methods availables

> Returns : list of interpolation methods availables, ['bilinear', 'bicubic', ...]

geophpy.dataset.**colormap_getlist**(*sort=True*)
> Get the list of available colormaps.

> If sort is True, colormaps are sorted alphabetically ingoring the case.

geophpy.dataset.**colormap_plot**(*cmname*, *creversed=False*, *fig=None*, *filename=None*, *dpi=None*, *transparent=False*)
> Plots the colormap.

> Parameters :
>
> > **Cmname** Name of the colormap, 'gray_r' for example.
> >
> > **Creversed** True to add '_r' at the cmname to reverse the color map
> >
> > **Fig** figure to plot, None by default to create a new figure.
> >
> > **Filename** Name of the color map file to save, None if no file to save.
> >
> > **Dpi** 'dot per inch' definition of the picture file if filename != None
> >
> > **Transparent** True to manage the transparency.

> Returns:
>
> > **Fig** Figure Object

geophpy.dataset.**pictureformat_getlist**()
> Get list of pictures format availables

> Returns: list of picture formats availables, ['jpg', 'png', ...]

geophpy.dataset.**rasterformat_getlist**()
> Get list of raster format files availables

> Returns : list of raster file formats availables, ['jpg', 'png', ...]

---

geophpy.dataset.**correlmap**(*dataset*, *method='Crosscor'*)
    Profile-to-profile correlation map.

    Profile-to-profile correlation map:

    - each odd profile in the dataset is incrementally shifted and the correlation coefficient computed against neighbouring profiles for each shift value

    - profiles are considered to be vertical, and the shift performed along the profile (hence vertically)

    - the correlation map size is then "twice the image vertical size" (shift may vary from -ymax to +ymax) by "number of profiles" (correlation is computed for each column listed in input)

    Parameters method (*str, {'Crosscor', 'Pearson', 'Spearman', 'Kendall'}*) – Correlation method use.

    Returns

    - **cormap** (*2-D array_like*) – Profile-to-profile correlation map.

    - **pva1** (*2-D array_like*) – Correlation weight map.

geophpy.dataset.**griddinginterpolation_getlist**()
    To get the list of available gridding interpolation methods.

geophpy.dataset.**festooncorrelation_getlist**()
    To get the list of available festoon correlation methods.

geophpy.dataset.**sensorconfig_getlist**()
    Returns the list of available magnetic sensor configurations.

class geophpy.dataset.**Info**
    Class to sotre grid informations.

    x_min
        Grid minimum x values.

        Type float

    x_max
        Grid maximum x values.

        Type float

    y_min
        Grid minimum y values.

        Type float

    y_max
        Grid maximum y values.

        Type float

    z_min
        Grid minimum z values.

        Type float

    z_max
        Grid maximum z values.

        Type float

x_gridding_delta
Grid stepsize in the x_direction.

> **Type** float

y_gridding_delta
Grid stepsize in the y_direction.

> **Type** float

gridding_interpolation
Interpolation method used for gridding.

> **Type** str

plottype
Grid plot type.

> **Type** str

cmapname
Grid plot colormap name.

> **Type** str

class geophpy.dataset.**Data**
Class to sotre data.

fields
Field names corresponding to the data values ('x', 'y', 'vgrad').

> **Type** list of str

values
Ungridded data values.

> **Type** array-like

z_image
Gridded data values.

> **Type** 2-D array-like.

easting_image
???

northing_image
???

class geophpy.dataset.**GeoRefSystem**

class geophpy.dataset.**DataSet**
Creates a DataSet Object to process and display data.

info = Info() data = Data() georef = GeoRefSystem()

analyticsignal(*apod=0*)
Conversion from potential field to *analytic signal*.

> **Parameters** **apod** (`float`) – Apodization factor, to limit Gibbs phenomenon at jump discontinuities.

Notes

The amplitude of the *analytical signal* (or the *amplitude of the total gradient*) of a potential field T is defined as[1]:

$$|A(x, y, z)| = \sqrt{\left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial y}\right)^2 + \left(\frac{\partial T}{\partial z}\right)^2}$$

The directional derivative are computed in the spectral domain using[2]:

$$\mathcal{F}\left[\frac{\partial^2 T}{\partial x^2}\right] = (ik_x)^2 \mathcal{F}\left[T\right], \mathcal{F}\left[\frac{\partial^2 T}{\partial y^2}\right] = (ik_y)^2 \mathcal{F}\left[T\right], \mathcal{F}\left[\frac{\partial^2 T}{\partial z^2}\right] = |k|^2 \mathcal{F}\left[T\right].$$

and transformed back in the spatial domain for the total gradient amplitude calculation.

References

continuation (*apod=0*, *distance=2*, *continuationflag=True*, *totalfieldconversionflag=False*, *separation=0.7*)
Upward or downwad continuation of the magnetic field.

The continuation computes the data that would be measured at an upper (*upward continuation*) or lower survey altitude (*downward continuation*). The computation is done in the spectral (frequency) domain using Fast Fourier Transform.

Returns the continued *DataSet()* object.

> **Parameters**
>
> - **apod** (`float`) – Apodization factor (in %), to limit Gibbs phenomenon at jump discontinuities.
>
> - **distance** (`float`) – Continuation distance. Positive for an *upward continuation* (above ground level, away from the source) and negative for a *downward continuation* (under ground level, toward the source).
>
> - **totalfieldconversionflag** (`bool,`) – If True, the data are considered as gradient data (Total-field gradient or Fluxgate) and will be converted to total-field data after the continuation using the provided separation.
>
> - **separation** (`folat`) – Sensor separation for the conversion to Total-field data.

Notes

Assuming that all the magnetic sources are located below the observation surface, the continuation at a new observation altitude z of a survey acquired at an original altitude z_0 is given in the spectral domain by[3]:

$$\mathcal{F}_{\Delta z,k} = \mathcal{F}_{TF} \cdot e^{-\Delta z|k|}$$

where $\mathcal{F}_{TF}$ is the Fourier Transform of the measured data at the original altitude of observation z_0; $\mathcal{F}_{\Delta z,k}$ is the Fourier Transform of the anomaly at the new altitude of observation $z = z_0 - \Delta z$; $\Delta z = z_0 - z$

---

[1] Blakely R. J. 1996. Potential Theory in Gravity and Magnetic Applications. Chapter 12.1, p313-320. Cambridge University Press.
[2] Roest Walter R. , Verhoef Jacob and Pilkington Mark 1992. Magnetic interpretation using the 3-D analytic signal. Geophysics, vol. 57, no. 1, p116-125.
[3] Blakely R. J. 1996. Potential Theory in Gravity and Magnetic Applications. Chapter 12.1, p313-320. Cambridge University Press.

is the altitude increase between the original and new altitude of observation and $|k| = \sqrt{k_x^2 + k_y^2}$ is the radial wavenumber where k_x and k_y are the wavenumber in the x and y-direction respectively.

The given altitude increase (Δz) is an algebraic value:

- If $\Delta z > 0$, the new altitude of observation is above the original altitude: the operation is an *upward continuation*;

- if $\Delta z < 0$, the new altitude of observation is below the original altitude: the operation is a *downward continuation*.

The *upward continuation* attenuates anomalies with respect to the wavelength in way that accentuates anomalies caused by deep sources and attenuates at the anomalies caused by shallow sources. It is hence a smoothing operator.

The *downward continuation* accentuates the shallowest components. It reduces spread of anomalies and corrects anomalies coalescences. It is usefull to discriminates the number of body source at the origin of a one big anomaly. It is an unsmoothing operator that is instable as small changes in the data can cause large and unrealistic variations so it is to be used with caution. Low-pass filtering before the *downward continuation* can be a solution to increase the filter stability.

References

copy ( )
> To duplicate a DataSet Object.
>
> Parameters:
>
>> **Dataset** DataSet Object to duplicate
>
> Returns:
>
>> **Newdataset** duplicated DataSet Object

correlmap (*method='Crosscor'*)
> Profile-to-profile correlation map.
>
> Each even profile in the dataset is incrementally shifted and the correlation coefficient is computed against the (standardized) mean of the two neighbouring profiles for each shift value.
>
> Profiles are considered to be vertical and the shift is performed along the profile direction (y-direction). For a dataset of size ny*nx , the correlation map size is then 2*ny*nx (shift may vary from -ymax to +ymax).
>
>> **Parameters method** (`str, {'Crosscor', 'Pearson', 'Spearman', 'Kendall'}`) – Correlation method use.
>>
>> **Returns**
>>
>> - **cormap** (*2-D array_like*) – Profile-to-profile correlation map.
>>
>> - **pva1** (*2-D array_like*) – Correlation weight map.

destripecon (*Nprof='all'*, *setmin=None*, *setmax=None*, *method='additive'*, *reference='mean'*, *config='mono'*, *valfilt=False*)
> Destriping dataset using profiles' statistical moments (Moment Matching method).
>
> Moment Matching method: the statistical moments (mean and standard deviation) of each profile in the dataset are computed and matched to reference values.
>
>> **Parameters**

- **Nprof** (*int or str ('all')*) – Number of neighboring profiles used to compute the the reference values. Set to `'all'` by default) to compute the mean over the whole dataset. If set to `0`, it is the zero-mean (or zero-median) traverse filter.

- **setmin** (*float or None*) – While computing the mean, do not take into account data values lower than `setmin`. If `None`, all data are considered.

- **setmax** (*float or None*) – While computing the mean, do not take into account data values greater than `setmax`. If `None`, all data are considered.

- **method** (*str {'additive','multiplicative'}*) – Destriping methode. If set to `'additive'` (default), destriping is done additively. If set to `'multiplicative'`, it is done multiplicatively.

- **reference** (*str {'mean' 'median'}*) – References used for destriping. If set to `'mean'` (default), destriping is done using mean and standard deviation. If set to `'median'`, destriping is done using median and interquartile range.

- **config** (*str {'mono','multi'}*) – Sensors configuration. If set to `'mono'` (default), destriping is done using only offset matching (mean or median). If set to `'multi'`, destriping is done using both offset and gain (mean and standard deviation or median and interquartile range).

- **valfilt** (*bool*) – If set to True, the `values()` are filtered instead of the `z_image()`.

destripecub (*Nprof=0*, *setmin=None*, *setmax=None*, *Ndeg=3*, *valfilt=False*)
  To destripe a DataSet Object by a cubic curvilinear regression (chi squared)

  Parameters:

  Dataset DataSet Object to be destriped

  Nprof number of profiles over which to compute the polynomial reference ; if set to 0 (default), compute the mean over the whole data

  Setmin while fitting the polynomial curve, do not take into account data values lower than setmin

  Setmax while fitting the polynomail curve, do not take into account data values greater than setmax

  Ndeg polynomial degree of the curve to fit

  Valfilt If True, the dataset *values* are filtered instead of the dataset *z_image*.

  See also:

  destripecon(), *zeromeanprofile()*

festoonfilt (*method='Crosscorr'*, *shift=0*, *corrmin=0.4*, *uniformshift=False*, *setmin=None*, *setmax=None*, *valfilt=False*)
  Filters festoon-like artefacts out of in the dataset.

  Returns the destaggered *DataSet()* object and the shift used for each profile.

  Parameters

  - **method** (str, {'Crosscorr', 'Pearson', 'Spearman' or 'Kendall'} (from *festooncorrelation_getlist()*)) – Correlation method to use to compute the correlation coefficient in the correlation map.

  - **shift** (*scalar or array of float*) – Shift value (in pixels) to apply to the dataset profile. If shit=0, the shift will be determined for each profile by correlation with neighbours. If shift is a vector each value in shift will be applied to its corresponding odd profile. In that case shift must have the same size as the number of odd profiles.

- **corrmin** (*scalar in the range [0-1]*) – Minimum correlation coefficient value to allow shifting.

- **uniformshift** (*bool*) – If True, the shift is uniform on the map. If False the shift depends on each profile.

- **setmin** (*float*) – Data values lower than setmin are ignored.

- **setmax** (*float*) – Data values higher than setmax are ignored.

- **valfilt** (*bool*) – If set to True, the values() are filtered instead of the z_image().

Returns shift – Shift values used to destagger the dataset (unmodified if provided as an input parameter).

Return type array of float

See also:

correlmap(), correlshift()

Notes

The festoon-like artefacts are filtered based on the correlation between neighboring profiles.

For each odd profile index in the dataset, the correlation with its neighboring profile is calculated using the provided correlation method. Then, the profile is shifted from a sample and the correlation is computed anew and so forth to build a correlation map fo revery possible shift. For each profile, the shift with the maximum correlation coefficient is chosen as the best shift and used to destagger the dataset. It the shift is set to be uniform on the map, the correlation map is summed up and the shit correspoding to the global maximum correlation coefficient is used for each odd profile.

Alternatively, if a custom set of shift is provided, it will be used to destagger the dataset. It must have the same size as the number of odd profile in the dataset.

Example

```
>>> dataset.festoonfilt(method='Crosscorr', shift=0, corrmin=0.6,
↪uniformshift=False)
```

static from_file(*filenameslist*, *fileformat=None*, *delimiter=None*, *x_colnum=1*, *y_colnum=2*, *z_colnum=3*, *skipinitialspace=True*, *skip_rows=0*, *fields_row=1*)
Build a DataSet Object from a file.

Parameters:

Filenameslist list of files to open ['file1.xyz', 'file2.xyz' …] or ['file*.xyz'] to open all files with filename beginning by "file" and ending by ".xyz"

Fileformat format of files to open (None by default implies automatic determination from filename extension)

Note: all files must have the same format

Delimiter pattern delimitting fields within one line (e.g. ' ', ',', ';' …)

X_colnum column number of the X coordinate of the profile (1 by default)

Y_colnum column number of the Y coordinate inside the profile (2 by default)

Z_colnum column number of the measurement profile (3 by default)

> > **Skipinitialspace** if True, several contiguous delimiters are equivalent to one
> >
> > **Skip_rows** number of rows to skip at the beginning of the file, i.e. total number of header rows (1 by default)
> >
> > **Fields_row** row number where to read the field names ; if -1 then default field names will be "X", "Y" and "Z"
>
> Returns:
>
> > **Success** true if DataSet Object built, false if not
> >
> > **Dataset** DataSet Object build from file(s) (empty if any error)
>
> Example:
>
> success, dataset = DataSet.from_file("file.csv")

get_grid_values()
    Return dataset Z_image.

get_gridcorners()
    Return dataset grid corners coordinates.

    Returns `None` if no interpolation was made. Use :meth:~geophpy.dataset.DataSet.get_boundingbox' to get the ungridded data values' bounding box.

get_gridextent()
    Return dataset grid extent.

get_median_xstep(*prec=2*)
    Return the median step between two distinct x values rounded to the given precision.

get_median_xystep(*x_prec=2*, *y_prec=2*)
    Return the median steps between two distinct x and y values rounded to the given precisions.

get_median_ystep(*prec=2*)
    Return the median step between two distinct y values rounded to the given precision.

get_values()
    Return the dataset values.

get_xgrid()
    Return dataset x-coordinate matrix of the Z_image.

get_xvalues()
    Return the x-coordinates from the dataset values.

get_xvect()
    Return dataset x-coordinate vector of the Z_image.

get_xygrid()
    Return dataset x- and y-coordinate matrices of the Z_image.

get_xyvalues()
    Return the x- and y-coordinates from the dataset values.

get_xyvect()
    Return dataset x- and y-coordinate vectors of the Z_image.

get_xyzvalues()
    Return both the x, y-coordinates and dataset ungridded values.

get_ygrid()
    Return dataset y-coordinate matrix of the Z_image.

---

get_yvalues()
>   Return the y-coordinates from the dataset values.

get_yvect()
>   Return dataset y-coordinate vector of the Z_image.

histo_getlimits(*valfilt=False*)
>   Return the data min and max values.

>   > **Parameters valfilt** (`bool`) – If True, min and max are taken from data values. Otherxise, min and max are taken from data z_image.

>   > **Returns zmin, zmax** – Data min and max values.

>   > **Return type** float

histo_plot(*fig=None, filename=None, zmin=None, zmax=None, cmapname=None, creversed=False, cmapdisplay=False, coloredhisto=True, dpi=None, transparent=False, valfilt=False*)
>   Plot the dataset histogram.

>   > **Parameters**

>   > - **fig** (`Matplotlib figure object`) – Figure to use for the plot, None (by default) to create a new figure.

>   > - **filename** (`str or None`) – Name of file to save the figure, None (by default).

>   > - **zmax** (`zmin,`) – Minimal and maximal values to represent.

>   > - **cmapname** (`str`) – Name of the color map to be used 'gray' for example. To use a reversed colormap, add '_r' at the end of the colormap name ('gray_r') or set `creversed` to `True`.

>   > - **creversed** (`bool`) – Flag to add '_r' at the end of the color map name to reverse it.

>   > - **cmapdisplay** (`bool`) – Flag to display a color bar (True by default).

>   > - **coloredhisto** (`bool`) – Flag to color the histogram using the given colormap (True by default). If False, balck will be used.

>   > - **dpi** (`int or None`) – Definition ('dot per inch') to use when saving the figure into a file (None by default).

>   > - **transparent** (`bool`) – Flag to manage transparency when saving the figure into a file (False by default).

>   > - **valfilt** (`bool`) – If set to True, the `values()` are filtered instead of the `z_image()` (False By default).

>   > **Returns**

>   > - **fig** (*Matplotlib Figure Object*) – Dataset histogram figure.

>   > - **cmap** (*Matplotlib ColorBar Object*) – Colorbar associated with the figure if `cmapdisplay` is `True`, `None` otherwise.

interpolate(*interpolation='none', x_step=None, y_step=None, x_prec=2, y_prec=2, x_frame_factor=0.0, y_frame_factor=0.0*)
>   Dataset gridding.

>   > **Parameters**

>   > - **interpolation** (`str {'none', 'nearest', 'linear', 'cubic'}`) – Method used to grid the dataset. Can be:

none (by default) simply projects raw data on a regular grid without interpolation. Holes are filled with NaNs and values falling in the same grid cell are averaged.

nearest return the value at the data point closest to the point of interpolation.

linear tesselate the input point set to n-dimensional simplices, and interpolate linearly on each simplex.

cubic return the value determined from a piecewise cubic, continuously differentiable (C1), and approximately curvature-minimizing polynomial surface.

- **y_step** (*x_step,*) – Gridding step in the x and y direction. Use None (by default) to estimate the median x- and y-step from data values.

- **y_prec** (*x_prec,*) – Decimal precision to keep for the grid computation. None (by default) to use the maximal number of decimal present in the data values coordinates.

- **x_frame_factor** – Frame extension coefficient along x axis (e.g. 0.1 means xlength +10% on each side, i.e. xlength +20% in total) ; pixels within extended borders will be filled with "nan"

- **y_frame_factor** – Frame extension coefficient along y axis (e.g. 0.45 means yheight +45% top and bottom, i.e. yheight +90% in total) ; pixels within extended borders will be filled with "nan"

Returns

Return type Gridded DataSet object.

logtransform (*multfactor=1*, *setnan=True*, *valfilt=False*)
Apply a logarihtmic transformation to the dataset.

The logarihtmic transformation is a contrast enhancement filter that enhances information at low-amplitude values while preserving the relative amplitude information.

Returns the transformed *DataSet()* object.

Parameters

- **multfactor** (*float*) – Multiplying factor to apply to the data to increase/decrease the number of data that falls into the condition]-1,1[.

- **setnan** (*bool,*) – If True, the data value between]-1,1[will be replaced by nans. If False, they will be replaced by zero.

- **valfilt** (*bool,*) – If True, then filters *values* instead of z_image

Notes

The logarihtmic transformation is defined as[4]:

$$\mathcal{F}\{f\} = \begin{cases} -\log_{10}(-f) & for & f < -1 \\ \log_{10}(f) & for & f > 1 \\ 0 & for & -1 < f > 1 \end{cases}$$

where: f is the original data.

[4] Morris B., Pozza M., Boyce J. and Leblanc G. 2001. Enhancement of magnetic data by logarithmic transformation. The Leading Edge, vol. 20, no. 8, p882-885.

References

magconfigconversion(*fromconfig*, *toconfig*, *apod=0*, *FromBottomSensorAlt=0.3*, *FromTopSensorAlt=1.0*, *ToBottomSensorAlt=0.3*, *ToTopSensorAlt=1.0*, *inclination=65*, *declination=0*, *azimuth=0*, *magazimuth=None*)

Conversion between the different magnetic survey sensor's configurations.

Returns the transformed `DataSet()` object.

Parameters

- **fromconfig** (*str {"TotalField"|"TotalFieldGradient"|"Fluxgate"}, from sensorconfig_getlist()*) – Initial sensor's configuration from which to convert de data.

- **toconfig** (*str {"TotalField"|"TotalFieldGradient"|"Fluxgate"}, from sensorconfig_getlist()*) – Final sensor's configuration to which to convert de data.

- **apod** (*float*) – Apodization factor, to limit Gibbs phenomenon at jump discontinuities.

- **FromBottomSensorAlt** (*float*) – Bottom sensor altidue of the initial sensor's configuration.

- **FromTopSensorAlt** (*float*) – Top sensor altidue of the initial sensor's configuration.

- **ToBottomSensorAlt** (*float*) – Bottom sensor altidue of the final sensor's configuration.

- **ToTopSensorAlt** (*float*) – Top sensor altidue of the final sensor's configuration.

- **inclination** (*float*) – Ambient magnetic field inclination (I) in degrees positive below horizontal.

- **declination** (*float*) – Ambient magnetic field declination (D) in degrees positive east of geographic (true) north.

- **azimuth** (*float*) – Azimuth of the survey **x-axis** in degrees positive east of north (θ, the angle between the survey profile direction and the geographic north). The *magnetic azimuth* (ϕ) is computed from the `declination` (D) and the azimuth as $\phi = D - \theta$.

- **magazimuth** (*float*) – *Magnetic azimuth* survey **x-axis** in degrees positive east of north (ϕ, the angle between the survey profile direction and the magnetic north). `None` by default. If a value is given, the `declination` is ignored and the magnetic azimuth will derectly be used.

Notes

as[5]:

References

meantrack_plot(*fig=None*, *filename=None*, *Nprof=0*, *setmin=None*, *setmax=None*, *method='additive'*, *reference='mean'*, *config='mono'*, *Ndeg=None*, *plotflag='raw'*, *dpi=None*, *transparent=False*)

Plot the dataset mean cross-track profile before and after destriping.

Parameters :

---

[5] Tabbagh A., Desvignes G. and Dabas M. 1997. Processing of Z Gradiometer Magnetic data using Linear Transforms and Analytical Signal. Archaelogical Prospection, vol. 4, issue 1, p1-13.

Fig figure to plot, None by default to create a new figure. *matplotlib.figure.Figure*

Filename Name of the histogram file to save, None if no file to save.

Nprof number of profiles to compute the reference mean

Setmin float, While computing the mean, do not take into account data values lower than *setmin*.

Setmax float, While computing the mean, do not take into account data values greater than *setmax*.

Method destriping method (additive or multiplicative)

Reference destriping reference (mean and standard deviation or median and interquartile range)

Config destriping configuration ('mono' sensor: only offset matching (mean / median), 'multi' sensor: both offset and gain (standard deviation/interquartile range))

Plotflag str, {'raw', 'destriped', 'both'} to plot raw, destriped or both data

Ndeg polynomial degree of the curve to fit

Transparent True to manage the transparency.

Returns :

Fig Figure Object

Note: The mean cross-track profile is the profile composed the mean of each profile in the dataset.

---

medianfilt (*nx=3*, *ny=3*, *percent=0*, *gap=0*, *valfilt=False*)
Apply a median filter (*decision-theoretic* or *standard*) to the dataset.

Returns the filtered [*DataSet()*](#) object.

> **Parameters**
>
> - **nx** (*int*) – Size, in number of sample, of the filer in the x-direction.
>
> - **ny** (*int*) – Size, in number of sample, of the filter in the y-direction.
>
> - **percent** (*float*) – Threshold deviation (in percents) to the local median value (for absolute field measurements).
>
> - **gap** (*float*) – Threshold deviation (in raw units) to the median value (for relative anomaly measurements).
>
> - **valfilt** (*bool*) – If set to True, the `values()` are filtered instead of the `z_image()`.

peakfilt (*method='hampel'*, *halfwidth=5*, *threshold=3*, *mode='relative'*, *setnan=False*, *valfilt=False*)
Eliminate peaks from the dataset.

Returns the de-peaked [*DataSet()*](#) object.

> **Parameters**
>
> - **method** (*str {'median', 'hampel'}*) – Type of the `decision-theoric filter` used to determine outliers.
>
> - **halfwidth** (*scalar*) – Filter half-width.
>
> - **threshold** (*scalar (positive)*) – Filter threshold parameter. If t=0 and method='hampel', it is equal to a `standard median filter`.

- **mode** (*str {'relative', 'absolute'}*) – Median filter mode. If 'relative', the threshold is a percentage of the local median value. If 'absolute', the threshold is a value.

- **setnan** (*bool*) – If True, the outliers are replaced by nan instead of the local median.

- **valfilt** (*bool*) – If set to True, the `values()` are filtered instead of the `z_image()`.

plot(*plottype='2D-SCATTER'*, *cmapname=None*, *creversed=False*, *fig=None*, *filename=None*, *cmmin=None*, *cmmax=None*, *interpolation='bilinear'*, *levels=100*, *cmapdisplay=True*, *axisdisplay=True*, *labeldisplay=True*, *pointsdisplay=False*, *dpi=None*, *transparent=False*, *logscale=False*, *rects=None*, *points=None*, *marker='+'*, *markersize=None*)

2D representation of the dataset.

Parameters

- **plottype** (*str, {'2D-SCATTER', '2D-SURFACE', '2D-CONTOUR', '2D-CONTOURF', '2D-POSTMAP'}*) – Plot type for the data representation. Plot type can be

  2D-SCATTER (by default) ungridded data values are dispay in a scatter plot. Plot type 'SCATTER', 'SCAT' and 'SC' are also recognised.

  2D-SURFACE Gridded data values are dispay in a surface (image) plot. Plot type 'SURFACE', 'SURF' and 'SF' are also recognised.

  2D-CONTOUR Gridded data values are dispay in a UNFILLED contour plot. If this plot type is used, you can specify the number of contours used with the `levels` keyword. Plot type 'CONTOUR', 'CONT' and 'CT' are also recognised.

  2D-CONTOURF Gridded data values are dispay in a FILLED contour plot. If this plot type is used, you can specify the number of contours used with the `levels` keyword. Plot type 'CONTOURF', 'CONTF' and 'CF' are also recognised.

  2D-POSTMAP Ungridded data position are dispay in a scatter plot. It is different from the '2D-SCATTER' plot because the data value is not represented (all point have the same color). Plot type 'POSTMAP', 'POST' and 'PM' are also recognised.

- **cmapname** (*str*) – Name of the color map to be used 'gray' for example. To use a reversed colormap, add '_r' at the end of the colormap name ('gray_r') or set `creversed` to `True`.

- **creversed** (*bool*) – Flag to add '_r' at the end of the color map name to reverse it.

- **fig** (*Matplotlib figure object*) – Figure to plot, None by default to create a new figure. If a figure object is provided, it wiil be cleared before displaying the current data.

- **filename** (*str or None*) – Name of file to save the figure, None (by default).

- **cmmin** (*scalar*) – Minimal value to display in the color map range.

- **cmmax** (*scalar*) – Maximal value to display in the color map range.

- **interpolation** (*str,*) – Interpolation method used for the pixel interpolation ('bilinear' by default). If interpolation is 'none', then no interpolation is performed for the display. Supported values are 'none', 'nearest', 'bilinear', 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos'.

- **levels** (*(int or array-like)*) – Determines the number and positions of the contour lines / regions. If an int n, use n data intervals; i.e. draw n+1 contour lines. The level heights are automatically chosen. If array-like, draw contour lines at the specified levels. The values must be in increasing order.

- **cmapdisplay** (`bool`) – True to display colorbar, False to hide the colorbar.
- **axisdisplay** (`bool`) – True to display axis (and their labels), False to hide axis (and labels).
- **labeldisplay** (`bool`) – True to display labels on axis, False to hide labels on axis.
- **pointsdisplay** (`bool`) – True to display grid points.
- **dpi** (`int or None`) – Definition ('dot per inch') to use when saving the figure into a file (None by default).
- **transparent** (`bool`) – Flag to manage transparency when saving the figure into a file (False by default).
- **logscale** (`bool`) – Flag to use a logarithmic color scale.
- **rects** (`None or array_like`) – Coordinates of additional rectanngles to display: [[x0, y0, w0, h0], [x1, y1, w1, h1], . . .]. None by default.
- **points** (`None or array_like`) – Coordinates of additional points to display: [[x0, y0], [x1, y1], . . .]. None by default.
- **marker** (`str`) – Matplotlib marker style for data point display.
- **markersize** (`scalar`) – Size for the marker for data point display.

  Returns

- **fig** (*Matplotlib Figure Object*) – Dataset map figure.
- **cmap** (*Matplotlib ColorBar Object*) – Colorbar associated with the figure if `cmapdisplay` is `True`, `None` otherwise.

ploughfilt (*apod=0*, *azimuth=0*, *cutoff=100*, *width=2*, *valfilt=False*)
Apply a directionnal ("anti-ploughing") filter to the dataset.

Returns the filtered [*DataSet()*](#) object.

  Parameters

- **apod** (`float`) – Apodization factor in percent [0,1].
- **azimuth** (`scalar`) – Filter azimuth in degree.
- **cutoff** (`scalar`) – Cutoff spatial frequency (in number of sample).
- **width** (`int`) – Filter width parameter.
- **valfilt** (`bool`) – If set to True, the `values()` are filtered instead of the `z_image()`.

Notes

The filter used is a combination of a classic gaussian low-pass filter of order 2 with a directional filter. This gaussian low-pass directional filter is defined as[6]:

$$\mathcal{F}(\rho, \theta, f_c) = e^{-(\rho/f_c)^2} \cdot \left(1 - e^{-r^2/\tan(\theta - \theta_0)^n}\right)$$

where: ρ and θ are the current point polar coordinates; f_c is the gaussian low-pass filter cutoff frequency; θ_0 is the directional filter's azimuth and n is the parameter that controls the filter width.

---

[6] Tabbagh J. 2001. Filtre directionel permettant d'eliminer les anomalies crees par le labour, in "Filtering, Optimisation and Modelling of Geophysical Data in Archaeological Prospecting", Fondazione Ing. Carlo Maurillo Lerici, Politecnico di Milano, M. Cucarzi and P. Conti (eds.), Roma 2001, 202p, p161-166.

References

Examples

```
>>> dataset.ploughfilt()
>>> dataset.ploughfilt(apod=0, azimuth=30, cutoff=100, width=2)
>>> dataset.ploughfilt(azimuth=45, cutoff=None)
```

polereduction(*apod=0*, *inclination=65*, *declination=0*, *azimuth=0*, *magazimuth=None*, *incl_magn=None*, *decl_magn=None*)
Reduction to the pole.

The reduction to the pole is a phase transformation in the spectral domain applied to the **total magnetic field** to computes the "anomaly that would be measured at the north magnetic pole, where induced magnetization and ambient field both would be directed vertically down."[7]

Returns the transformed *DataSet()* object.

> **Parameters**
>
> - **apod** (*float*) – Apodization factor, to limit Gibbs phenomenon at jump discontinuities.
>
> - **inclination** (*float*) – Ambient magnetic field inclination (I) in degrees positive below horizontal.
>
> - **declination** (*float*) – Ambient magnetic field declination (D) in degrees positive east of geographic (true) north.
>
> - **azimuth** (*float*) – Azimuth of the survey **x-axis** in degrees positive east of north (θ, the angle between the survey profile direction and the geographic north). The *magnetic azimuth* (ϕ) is computed from the declination (D) and the azimuth as $\phi = D - \theta$.
>
> - **magazimuth** (*float*) – *Magnetic azimuth* survey **x-axis** in degrees positive east of north (ϕ, the angle between the survey profile direction and the magnetic north). None by default. If a value is given, the declination is ignored and the magnetic azimuth will derectly be used.
>
> - **decl_magn** (*incl_magn,*) – The source remanent magnetization inclination and declination in degrees. By default they are set to None, the remanent magnetization is neglected.

Notes

If the magnetization and ambient field are not vertical, a uniform magnetic source will produce a skewed anomaly. The reduction to the pole aims to eliminate this effect by transforming the "measured **total field** anomaly into the vertical component of the field caused by the same source distribution magnetized in the vertical direction".

This transformation is given in the spectral domain by:

$$\mathcal{F}_{RTP} = \mathcal{F}_{TF} \cdot \mathcal{F}$$

where $\mathcal{F}_{TF}$ is the Fourier Transform of measured *total field* anomaly and $\mathcal{F}$ is defined as:

$$\mathcal{F}_{m,f}\{k_x, k_x\} = \frac{|k|^2}{a_1 k_x^2 + a_2 k_y^2 + a_3 k_x k_y + i|k|(b1k_x + b2k_y)}, |k| \neq 0$$

---

[7] Blakely R. J. 1996. Potential Theory in Gravity and Magnetic Applications. Chapter 12.3.1, p330. Cambridge University Press.

with

$$a_1 = m_z f_z - m_x f_x,$$
$$a_2 = m_z f_z - m_y f_y,$$
$$a_3 = -m_y f_x - m_x f_y,$$
$$b_1 = m_x f_z + m_z f_x,$$
$$b_2 = m_y f_z + m_z f_y,$$

where $m = (m_x, m_y, m_z)$ is the unit-vector in the direction of the magnetization of the source; $f = (f_x, f_y, f_z)$ is the unit-vector in the direction of the ambiant field; $|k| = \sqrt{k_x^2 + k_y^2}$ is the radial wavenumber and k_x and k_y are the wavenumber in the x and y-diection respectively.

References

regrid (*x_step=None*, *y_step=None*, *method='cubic'*)
> Dataset re-gridding.
>
> During regridding, a copy of the dataset is re-sampled (see `sample()`) and interpolated with the given x and y-steps.
>
> > **Parameters**
> >
> > - **y_step** (`x_step,`) – Gridding step in the x and y direction. Use None (by default) to resample current grid by a factor 2 (step_new = step_old*0.5).
> >
> > - **method** (`str {'cubic', 'nearest', 'linear'}`) – Method used to re-grid the dataset. see `interpolate()`
> >
> > **Returns**
> >
> > **Return type** Re-gridded DataSet object.

Note: Only the grid is affected and the ungriddedd data values are kept unchanged. to propagate a change in the ungriddedd data values, you may want to use the filters `valfilt` keyword or the association of `sample()` and `sample()`.

> **See also:**
>
> `sample()`, `interpolate()`

regtrend (*nx=3*, *ny=3*, *method='relative'*, *component='local'*, *valfilt=False*)
> To filter a DataSet Object from its regional trend
>
> Parameters:
>
> > **Dataset** DataSet Object to be filtered
> >
> > **Nx** filter size in x coordinate
> >
> > **Ny** filter size in y coordinate
> >
> > **Method** set to "relative" to filter by relative value (resistivity) or to "absolute" to filter by absolute value (magnetic field)
> >
> > **Component** set to "local" to keep the local variations or to "regional" to keep regional variations
> >
> > **Valfilt** if set to True, then filters data.values instead of data.zimage

sample()
    Re-sample data at ungridded sample position from gridded Z_image.

    Re-sample ungridded values from gridded Z_image using a cubic spline spline interpolation.

threshold(*setmin=None*, *setmax=None*, *setmed=False*, *setnan=False*, *valfilt=False*)
    Threshold a dataset in the given interval.

    Returns the thresholded *DataSet()* object.

    Parameters

- **setmin** (*float*) – Minimal interval value. All values lower than setmin will be replaced by setmin (if both setmed and setnan are False).

- **setmax** (*float*) – Maximal interval value. All values lower than setmax will be replaced by setmax (if both setmed and setnan are False).

- **setmed** (*bool*) – If set to True, out of range values are replaced by the profile's median.

- **setnan** (*bool*) – If set to True, out of range values are replaced by NaNs

- **valfilt** (*bool*) – If set to True, the values() are filtered instead of the z_image().

    Note: If both setnan and setmed are True at the same time, setnan prevails.

to_file(*filename*, *fileformat=None*, *delimiter='\t'*, *description=None*, *gridtype='surfer7bin'*)
    Save a DataSet Object to a file.

    Parameters :

    Filename name of file to save.

    Fileformat format of output file . . .

    Delimiter delimiter between fields in a line of the output file, ' ', ',', ';', . . .

    Gridtype str {'surfer7bin', 'surfer6bin', 'surfer6ascii'}, optional Format of the grid file to write. By default 'surfer7bin'.

    Returns :

    Success boolean

wallisfilt(*nx=11*, *ny=11*, *targmean=125*, *targstdev=50*, *setgain=8*, *limitstdev=25*, *edgefactor=0.1*, *valfilt=False*)
    Apply the Wallis contrast enhancement filter to the dataset.

    Returns the contrast-enhanced *DataSet()* object.

    Parameters

- **nx** (*int*) – filter window size in x-direction

- **ny** (*int*) – filter window size in y-direction

- **targmean** (*float*) – The target mean brigthness level (m_d)

- **targstdev** (*float*) – The target standard deviation (σ_d)

- **setgain** (*float*) – Amplification factor for contrast (A)

- **limitstdev** (*float*) – Limitation on the window standard deviation to prevent too high gain value if data are dispersed

- **edgefactor** (*float in the range of [0,1]*) – Brightness forcing factor (α), controls ratio of edge to background intensities.
- **valfilt** (*bool*) – If set to True, the `values()` are filtered instead of the `z_image()`.

Notes

The Wallis filter is a locally adaptative contrast enhancement filter based on the local statistical properties of sub-windows in the image. It adjusts brightness values (grayscale image) in the local window so that the local mean and standard deviation match target values.

The Wallis operator is defined as[8]:

$$\frac{A\sigma_d}{A\sigma_{(x,y)} + \sigma_d}[f_{(x,y)} - m_{(x,y)}] + \alpha m_d + (1 - \alpha)m_{(x,y)}$$

where: A is the amplification factor for contrast; σ_d is the target standard deviation; $\sigma_{(x,y)}$ is the standard deviation in the current window; $f_{(x,y)}$ is the center pixel of the current window; $m_{(x,y)}$ is the mean of the current window; α is the edge factor (controlling portion of the observed mean, and brightness locally to reduce or increase the total range) and m_d is the target mean.

As the Wallis filter is design for grayscale image, the data are internally converted to brightness level before applying the filter. The conversion is based on the minimum and maximum value in the dataset and uses 256 levels (from 0 to 255).

References

Examples

```
>>> dataset.wallisfilt()
>>> dataset.wallisfilt(nx=21, ny=21, targmean=125, targstdev=50)
```

zeromeanprofile (*setvar='median'*, *setmin=None*, *setmax=None*, *valfilt=False*)
Subtract the mean (or median) of each profile in the dataset.

Returns the zero-mean (or zero-median) *DataSet()* object.

> **Parameters**
>
> - **setvar** (*str, {'mean', 'median'}*) – Profile's statistical property be subtracted from each profile.
> - **setmin** (*float*) – While computing the mean, do not take into account data values lower than *setmin*.
> - **setmax** (*float*) – While computing the mean, do not take into account data values greater than *setmax*.
> - **valfilt** (*bool*) – If set to True, the `values()` are filtered instead of the `z_image()`.
>
> **See also:**
>
> *destripecon()*, *destripecub()*

---

[8] Scollar I., Tabbagh A., Hesse A. and Herzog I. 1990. Archaeological Prospecting and Remote Sensing (Topics in Remote Sensing 2). 647p, chapter 4.5 p174. Cambridge University Press.

Notes

For each profile in the dataset, the mean (or median depending on `setvar`) is calculated and subtracted from the profile.

This is equivalent to the [`destripecon()`](#) method in configuration `mono sensor` using the `additive` destriping `method` and a `number of profile` for the calculation equals to `zero`.

Examples

```
>>> dataset.zeromeanprofile(setvar='median')
    equivalent to
>>> dataset.destripecon(Nprof=0, method='additive', config='mono', reference=
↪'median')
```

9.3.2 geophpy.geoposset

Module for the management of Geographic Positioning Sets.

This module provides a number of tools, including the [`GeoPosSet`](#) class, to dealing with Geographic Positioning Sets (or Ground Control Points).

copyright Copyright 2014-2019 Lionel Darras, Philippe Marty, Quentin Vitale and contributors, see AUTHORS.

license GNU GPL v3.

9.3.3 Conversion

- *utm_to_wgs84* – Convert UTM to lat, long coordinates.

- *wgs84_to_utm* – Convert lat, long to UTM coordinates.

9.3.4 Saving

- *save* – Save GCPs to an ascii file.

- *to_kml* – Save GCPs to a kml file.

geophpy.geoposset.**refsys_getlist**()
    List of available geographic reference system.

geophpy.geoposset.**filetype_getlist**()
    List read file types, 'ascii', 'shapefile', ...

geophpy.geoposset.**utm_to_wgs84**(*easting*, *northing*, *zonenumber*, *zoneletter*)
    Conversion from UTM to WGS84 coordinates (lat, lon).

    Parameters

- **easting** (`scalar`) – Easting UTM coordinate.

- **northing** (`scalar`) – Northing UTM coordinate.

- **zonenumber** (`int`) – UTM zone number.

- **zoneletter** (`str`) – UTM zone letter.

> **Returns**
>
> > - **latitude** (`scalar`) – WGS84 latitude coordinate.
> >
> > - **longitude** (`scalar`) – WGS84 longitude coordinate.

`geophpy.geoposset.`**`wgs84_to_utm`**(*latitude*, *longitude*)

> Conversion from WGS84 to UTM coordinates.
>
> works on list
>
> > **Parameters**
> >
> > > - **latitude** (`scalar`) – WGS84 latitude coordinate.
> > >
> > > - **longitude** (`scalar`) – WGS84 longitude coordinate.
> >
> > **Returns**
> >
> > > - **easting** (`scalar`) – Easting UTM coordinate.
> > >
> > > - **northing** (`scalar`) – Northing UTM coordinate.
> > >
> > > - **zonenumber** (`int`) – UTM zone number.
> > >
> > > - **zoneletter** (`str`) – UTM zone letter.

`geophpy.geoposset.`**`utm_getzonelimits`**()

> UTM coordinates system min and max numbers and letters.
>
> > **Returns**
> >
> > > - **min_number** (`int`) – Minimal number of the UTM zone (1).
> > >
> > > - **min_letter** (`str`) – Minimal letter of the UTM zone (E).
> > >
> > > - **max_number** (`int`) – Maximal number of the UTM zone (60).
> > >
> > > - **max letter** (`str`) – Maximal letter of the UTM zone (X).

`geophpy.geoposset.`**`isvalid_utm_letter`**(*strg*)

> Check validity of an UTM zone letter.

`geophpy.geoposset.`**`isvalid_utm_number`**(*strg*)

> Check validity of an UTM zone number.

class `geophpy.geoposset.`**`GeoPosSet`**(*refsystem=None*, *utm_letter=None*, *utm_number=None*, *points_list=None*)

> Class to manage geographic positioning set.
>
> **`refsystem`**
> > Geographic reference system ('UTM', 'WGS84', ...).
> >
> > > **Type** `str` or `None`, opt
>
> **`utm_zoneletter`**
> > Utm zone letter for 'UTM' *refsystem* (E -> X).
> >
> > > **Type** str, opt
>
> **`utm_zonenumber`**
> > Utm zone number for 'UTM' *refsystem* (1 -> 60).
> >
> > > **Type** int, opt
>
> **`points_list`**
> > List of Ground Control Points: >>> [[lat1, lon1, x1, y1], [lat2, lon2, x2, y2], ...]
> >
> > > **Type** `list` of `scalar`, opt

classmethod from_ascii_file(*filenames*, *delimiter=None*)
    Build a geophpy.geoposdet.GeoPosSet object from one or several ascii files.

> **Parameters**
>
> - **filenames** (str or list of str) – Names of files to be read.
>
> - **delimiter** (str or None) – The ASCII file delimiter. If None (by default), the delimiter will be sniffed from the file itself.
>
> **Returns**
>
> - [*GeoPosSet*](#) object (possibly empty).
>
> - **succes** (bool) – True if build was successful, False otherwise.

classmethod from_file(*filenames*, *filetype=None*)
    Build a GeoPosSet object from one or several files.

> **Parameters**
>
> - **filenames** (str or list of str) – Names of files to be read.
>
> - **filetype** ({'ascii', 'shapefile', None}) – Type of the files to read. Id None (default), the file type will be determined from the file extension.
>
> **Returns**
>
> - [*GeoPosSet*](#) object (possibly empty).
>
> - **succes** (*bool*) – True if build was successful, False otherwise.

plot(*filename=None*, *dpi=None*, *transparent=False*, *i_xmin=None*, *i_xmax=None*, *i_ymin=None*, *i_ymax=None*, *long_label=False*)
    Display GCPs.

    Plots the GCPs using the point number as label. To save the plot, use the picturefilename option.

> **Parameters**
>
> - **filename** (str or None) – Name of the file to save the picture. If None, no picture is saved.
>
> - **dpi** (int) – 'dot per inch' definition for the picture file if filename is not None.
>
> - **transparent** (bool) – if True, picture display points not plotted as transparents
>
> - **i_xmin** (*x minimal value to display, None by default*) –
>
> - **i_xmax** (*x maximal value to display, None by default*) –
>
> - **i_ymin** (*y minimal value to display, None by default*) –
>
> - **i_ymax** (*y maximal value to display, None by default*) –
>
> - **long_label** (*bool*) – Flag to display both point number and local coordinates.
>
> **Returns**
>
> - **success** (*True if no error*)
>
> - **fig** (Figure object)

to_ascii(*filename*, *delimiter=';'*)
    Save [*GeoPosSet*](#) points list to an ascii file.

> **Parameters**
>
> - **filename** (str) – Name of the ascii file to save in.

> - **delimiter** (str, opt) – Delimiter to use.
>
>> **Returns success** – True if a file was saved.
>>
>> **Return type** bool

to_kml (*filename*)

Save *GeoPosSet* points list to a kml file.

> **Parameters filename** (str) – Name for the kml file to save in.
>
> **Returns success** – True if a file was saved.
>
> **Return type** bool

FEEDBACK & CONTRIBUTE

Your feedback is more than welcome.

Write email to lionel.darras@mom.fr, philippe.marty@upmc.fr or quentin.vitale@eveha.fr

To cite this software : "Marty, P., Darras, L. (2015). GeophPy. Tools for geophysical survey data processing (version x.y) [software]. Available at https://pypi.python.org/pypi/GeophPy."

CHANGELOG

11.1 Version 0.32

Released on 2019-08-01.

- HTML documentation theme changed to Read the Docs theme.

- Reading Georeferencing Ground Control Points from CSV as been extended to any delimiter-value file.

- Georeferencing now works ungridded dataset values.

- Forced equal aspect ratio for map plots.

- Added Dataset linear/polynomial detrending of dataset profile.

- Constant destriping now works on ungridded dataset values.

- Changed the euler deconvolution implementation (now allow automatic sub-windowing).

- Changed the magnetic continuation implementation.

- Changed the analytic signal implementation.

- Added import/export from/to Surfer grids.

- Changed the reduction to the pole implementation.

- Added directional filter (plough filter).

- Added 2D-Fourier spectral plot.

- Added setmin/setmax range possibility for correlation calculation in festoon filter.

- Uptaded Dataset mean cross-track plot.

- Added Dataset zero-mean/zero-median profile filter (works on both gridded and ungridded dataset values).

- Enhanced CSV opening file method for both dataset and geoposset.

- Added Datasets merging & edge matcing methods.

- Added Dataset translation & rotation.

- Added Label display option for 2-D plots.

- Added Scatter plot of raw values.

- Added number of levels or levels values specification for 2-D contour and filled contour plots.

- Fixed Filled 2-D contour plot bug.

11.2 Version 0.31

Released on 2018-01-02.

- Fixed GeophPy pip installation issues and updated documentation.

11.3 Version 0.30

Released on 2017-12-01.

- Updated GeophPy documentation.
- Added options for constant destriping filter.
- Added Mean cross-track profile plot for destripping filters.
- Implemented Wallis filter.
- Implemented replace by profile's median in peak filtering.
- Added automatic delimiter search in delimited files.
- Fixed reading delimited file issues.
- Added non uniform shift for Festoon.
- Fixed correlation and cross-correlation map calculation.
- Added color map for histogram plot.
- Fixed bug in histo.plot.

11.4 Version 0.21

Released on 2016-05-01

- Initial version.

TWELVE

REFERENCES

BIBLIOGRAPHY

[AsGA08] Aspinall A., Gaffney C. and Schmidt A. 2008. Magnetometry for Archaeologists. 224p. AltaMira Press.

[BLAK96] Blakely R. J. 1996. Potential Theory in Gravity and Magnetic Applications. Cambridge University Press.

[GaCs00] Gadallah F. L., Csillag F. 2000. Destriping multisensor imagery with moment matching. Int. J. Remote Sensing, vol. 21, no. 12, p2505-2511.

[LimJ90] Lim, J. S. 1990. Two-Dimensional Signal and Image Processing. p469-476. Prentice-Hall.

[MPBL01] Morris B., Pozza M., Boyce J. and Leblanc G. 2001. Enhancement of magnetic data by logarithmic transformation. The Leading Edge, vol. 20, no. 8, p882-885.

[PeGa16] Pearson R. K. and Gabbouj M. 2016. Nonlinear Gigital Filtering with Python: an Introduction, 1st edition. Chapter 4.7 p137. CRC Press.

[RAGM90] Reid A. B., Allsop J. M., Granser H., Millett A. J. and Somerton I. W. 1990. Magnetic interpretation in three dimensions using Euler deconvolution. Geophysics, vol. 55, no. 1, p80-91.

[ReEW14] Reid A. B., Ebbing J. and Webb S. J. 2014. Avoidable Euler Errors - the use and abuse of Euler deconvolution applied to potential fields. Geophysical Prospecting, vol. 62, issue 5, p1162-1168.

[RiJi06] Richards, J. A. and X. Jia 2006. Remote Sensing Digital Image Analysis - An Introduction, 4rth edition. Chapter 2.2.3 p37. Springer.

[RoVP92] Roest Walter R. , Verhoef Jacob and Pilkington Mark 1992. Magnetic interpretation using the 3-D analytic signal. Geophysics, vol. 57, no. 1, p116-125.

[Scho07] Schowengerdt R. A. .2007. Remote Sensing: Models and Methods for Image Processing, 3rd edition. Chapter 7.4 p325. Elsevier.

[STHH90] Scollar I., Tabbagh A., Hesse A. and Herzog I. 1990. Archaeological Prospecting and Remote Sensing (Topics in Remote Sensing 2). 647p. Cambridge University Press.

[TABB01] Tabbagh J. 2001. Filtre directionel permettant d'eliminer les anomalies crees par le labour, in "Filtering, Optimisation and Modelling of Geophysical Data in Archaeological Prospecting", Fondazione Ing. Carlo Maurillo Lerici, Politecnico di Milano, M. Cucarzi and P. Conti (eds.), Roma 2001, 202p, p161-166.

[TaDD97] Tabbagh A., Desvignes G. and Dabas M. 1997. Processing Z gradiometer magnetic data using linear transforms and analytical signal. Archaeological Prospection, vol. 4, issue 1, p1-13.