Welcome to WellDetective 1 documentation!

Contents:

Welcome to WellDetective documentation

What is WellDetective?

WellDetective is a Python module for magnetic, methane, and other dataset analysis for the identification and characterization of abandoned oil and gas wells.

What can WellDetective do?

WellDetective interprets and maps raw magnetic data, correcting for background magnetic noise, heading error, removing survey turns, performing reduction to pole, plotting, and detecting magnetic anamolies and comparing to known well datasets for potential unidentified well identification.

Obtaining and installing WellDetective

To obtain and install WellDetective, please use the following instructions

```
# clone into the git repo
$ git clone git@github.com:lanl/WellDetective.git
$ cd WellDetective/src
# install dependencies
$ python -m pip install --user -r requirements.txt
# make the packages available from any directory
$ python -m pip install -e .
```

Once the module in installed, example jupyter notebooks to test the installation are in the example/notebooks directory.

Contributors

- Eric Guiltinan
- Javier Santos
- Roman Colman

Contact

- Email: WellDetective@lanl.gov
- Please post issues on the github issues page

Copyright Information

WellDetective object - all functions

Complete list of functions on the WellDetective Class

---

class **WellDetective.src.general.WellDetective**(*data: DataFrame*)

A class for processing geophysical magnetic survey data.

Provides methods to read raw magnetic data, apply heading corrections, remove turning data, grid and filter data, identify magnetic anomalies (“hotspots”), and export results for visualization.

data

DataFrame containing processed magnetic survey data.

Type: pd.DataFrame

__init__(*data: DataFrame*)

Initialize GeoMagProcessor with survey data.

Parameters: **data** (*pd.DataFrame*) – Initial survey data with required columns (‘Lat’, ‘Long’, ‘Mag’).

__weakref__

list of weak references to the object (if defined)

add_heading_column(*lat_col='Lat'*, *lon_col='Long'*, *window=20*)

Compute and add a ‘Heading’ column based on coordinate differences.

Parameters:

- **lat_col** (*str*) – Column names for latitude and longitude.

- **lon_col** (*str*) – Column names for latitude and longitude.
- **window** (*int*) – Look-back window size for computing headings.

Return type: None

auto_equalize_heading_correction(*primary_range*, *secondary_range*, *mag_col='Mag'*)

Apply heading correction to equalize magnetic field measurements.

Parameters:
- **primary_range** (*tuple*) – Heading ranges for primary and secondary directions.
- **secondary_range** (*tuple*) – Heading ranges for primary and secondary directions.
- **mag_col** (*str*) – Column name of magnetic field measurements.

Return type: None

static **calculate_heading**(*lat1*, *lon1*, *lat2*, *lon2*)

Calculate compass heading between two geographical points.

Parameters:
- **lat1** (*float*) – Latitude and longitude coordinates in degrees.
- **lon1** (*float*) – Latitude and longitude coordinates in degrees.
- **lat2** (*float*) – Latitude and longitude coordinates in degrees.
- **lon2** (*float*) – Latitude and longitude coordinates in degrees.

Returns: Heading in degrees (0° is North).

Return type: float

detect_hotspots(*mag_grid*, *grid_x*, *grid_y*, *well_coords*, *distance_threshold=100*, *bandwidth=80*)

Detect hotspots and flag potential orphan wells.

Parameters:
- **mag_grid** (*np.array*) – 2D magnetic data grid.
- **grid_x** (*np.array*) – Coordinate grids.
- **grid_y** (*np.array*) – Coordinate grids.
- **well_coords** (*list of tuple*) – Known well coordinates in projected units.
- **distance_threshold** (*float*) – Minimum distance from a hotspot to known wells to be flagged as orphan.
- **bandwidth** (*float*) – MeanShift clustering bandwidth.

Returns:
- cluster centroids
- orphan wells

Return type: tuple of pd.DataFrame

find_primary_secondary_headings(*tolerance=10*)

Identify primary and secondary heading ranges using K-Means clustering.

Parameters: **tolerance** (*float*) – Degrees around the cluster centers for heading ranges.
Returns: Primary and secondary heading ranges as tuples (min, max).
Return type: tuple

grid_and_filter_data(*inclination*, *declination*, *grid_res=500*, *cutoff_wavelength=30*, *proximity_threshold=40*)

Interpolate, reduce-to-pole, and low-pass filter magnetic data. Subtracts mean magnetic field before processing and masks areas too far from the flight path.

Parameters:
- **inclination** (*float*) – IGRF inclination and declination values.
- **declination** (*float*) – IGRF inclination and declination values.
- **grid_res** (*int*) – Grid resolution for interpolation.
- **cutoff_wavelength** (*float*) – Wavelength for low-pass filter (meters).
- **proximity_threshold** (*float*) – Maximum distance (in meters) from flight line to keep grid points.

Returns: Processed and masked magnetic data grid.
Return type: xarray.DataArray

project_coordinates(*utm_zone=12*)

Project geographic coordinates to UTM.

Parameters: **utm_zone** (*int*) – UTM zone for projection.
Return type: None

remove_turning_data(*primary_range*, *secondary_range*)

Remove turning data based on heading ranges.

Parameters:
- **primary_range** (*tuple*) – Tuples representing primary and secondary heading ranges.
- **secondary_range** (*tuple*) – Tuples representing primary and secondary heading ranges.

Return type: None

segment_and_filter_data(*max_gap_distance=20*, *min_segment_length=100*, *lat_col='Lat'*, *lon_col='Long'*)

Segment data into continuous lines and remove segments shorter than a specified minimum length.

Parameters:
- **max_gap_distance** (*float*, *optional*) – Maximum allowed gap distance between consecutive points (in meters). A gap larger than this starts a new segment.
- **min_segment_length** (*float*, *optional*) – Minimum length of segments to retain (in meters). Segments shorter than this will be discarded.
- **lat_col** (*str*, *optional*) – Name of the latitude column.
- **lon_col** (*str*, *optional*) – Name of the longitude column.

Returns: Updates self.data to include only segments meeting length criteria.

Return type: None

Example Usage

The *WellDetective* class provides a modular and flexible pipeline for magnetic survey data preprocessing, correction, gridding, and anomaly detection. It supports heading-based corrections, distance-aware masking, reduction to the pole, and filtering of interpolated magnetic data using widely adopted Python scientific libraries like NumPy, SciPy, Pandas, and Xarray.

Flight Line Correction & Gridding

WellDetective enables heading correction and gridding for geophysical flight line data. The goal is to prepare the dataset for anomaly analysis by: 1. Calculating headings and identifying primary/secondary directions. 2. Removing noisy turn data. 3. Performing heading-based equalization of the magnetic signal. 4. Projecting to UTM and interpolating to a regular grid. 5. Reducing the magnetic field to the pole. 6. Applying a Gaussian low-pass filter.

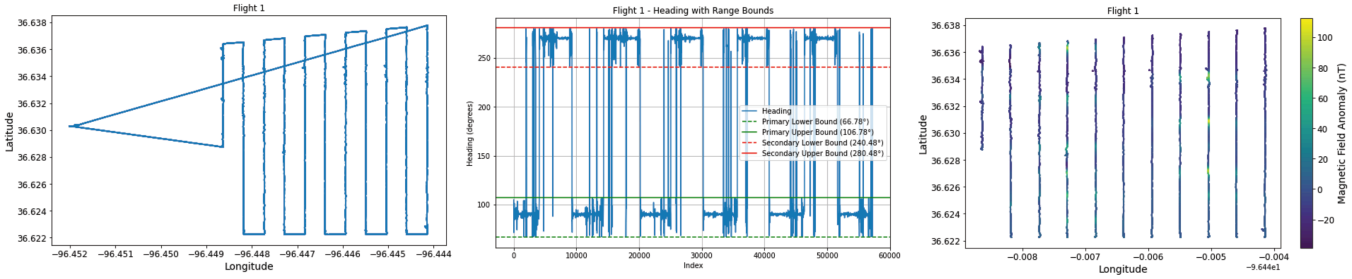
These steps are bundled into clean function calls, with smart masking and analysis-ready outputs.

Example

The example below demonstrates the full use of the class to clean, grid, and filter magnetic survey data.

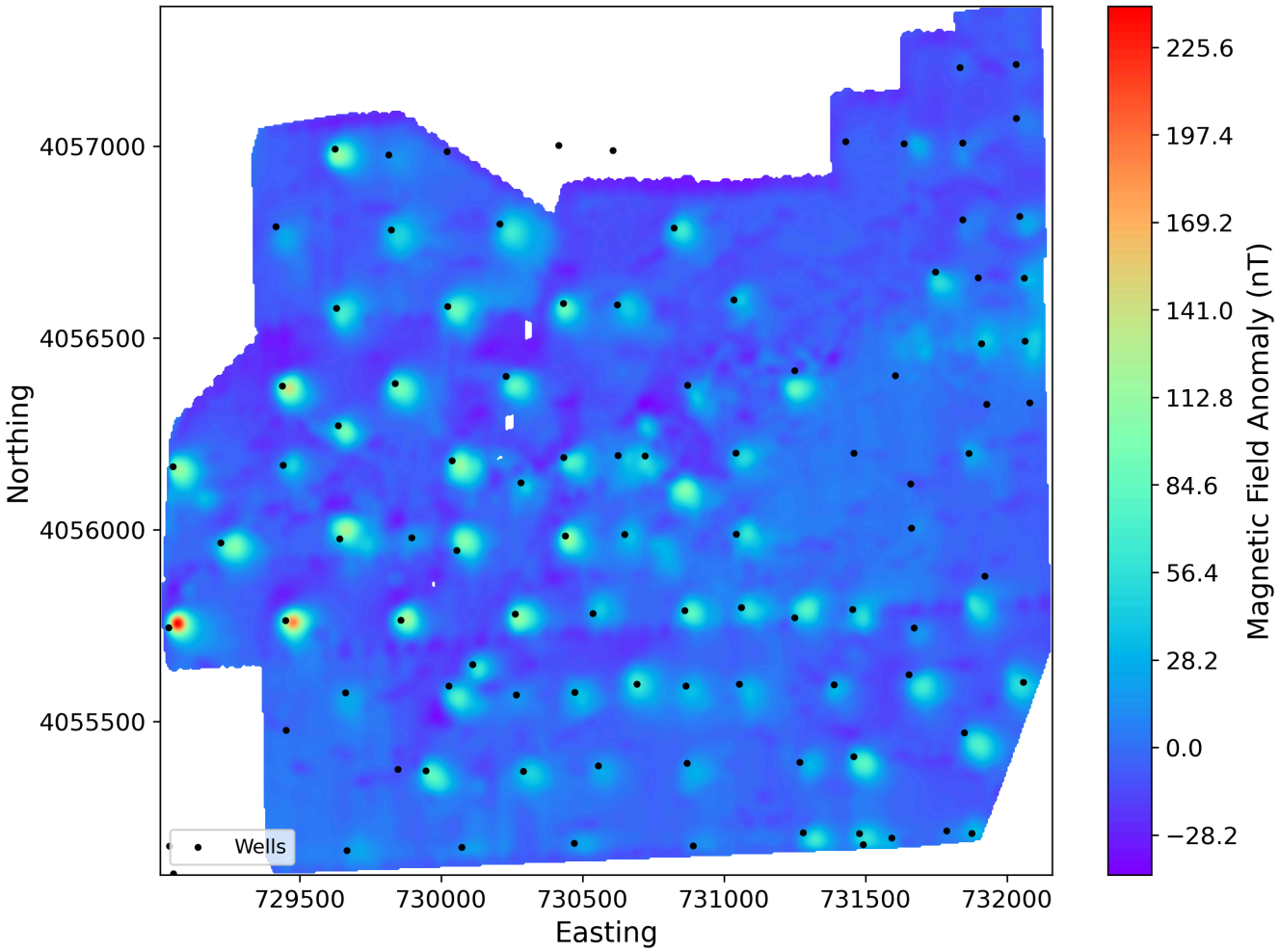
```python
import pandas as pd
from WellDetective import WellDetective

# Load raw flight line data
data = pd.read_csv("my_flight_data.asc", sep=r"\s+", skiprows=2)
data.rename(columns={"Longitude[°]": "Long", "Latitude[°]": "Lat", "Totalfield[nT]": "Mag"},
inplace=True)

# Initialize the processor
processor = WellDetective(data)

# Compute headings and remove turns
processor.add_heading_column()
primary, secondary = processor.find_primary_secondary_headings(tolerance=20)
processor.remove_turning_data(primary, secondary)

# Apply heading correction
processor.auto_equalize_heading_correction(primary, secondary)

# Project to UTM coordinates
processor.project_coordinates(utm_zone=12)

# Segment the flight lines and remove short segments
processor.segment_and_filter_data(max_gap_distance=15, min_segment_length=100)

# Grid and filter the data
rtp_grid = processor.grid_and_filter_data(
    inclination=60.2,
    declination=2.1,
    grid_res=500,
    cutoff_sigma=3,
    proximity_threshold=40
)

# Plot the result
rtp_grid.plot(cmap="viridis")
```

Flight Line with Heading Visualization



Heading ranges shown with colored bands over a single flight segment.

Gridded & Filtered Magnetic Field



Final output magnetic field grid after RTP and low-pass filtering, ready for anomaly detection.