IcePick and irlib Documentation

Release 0.5

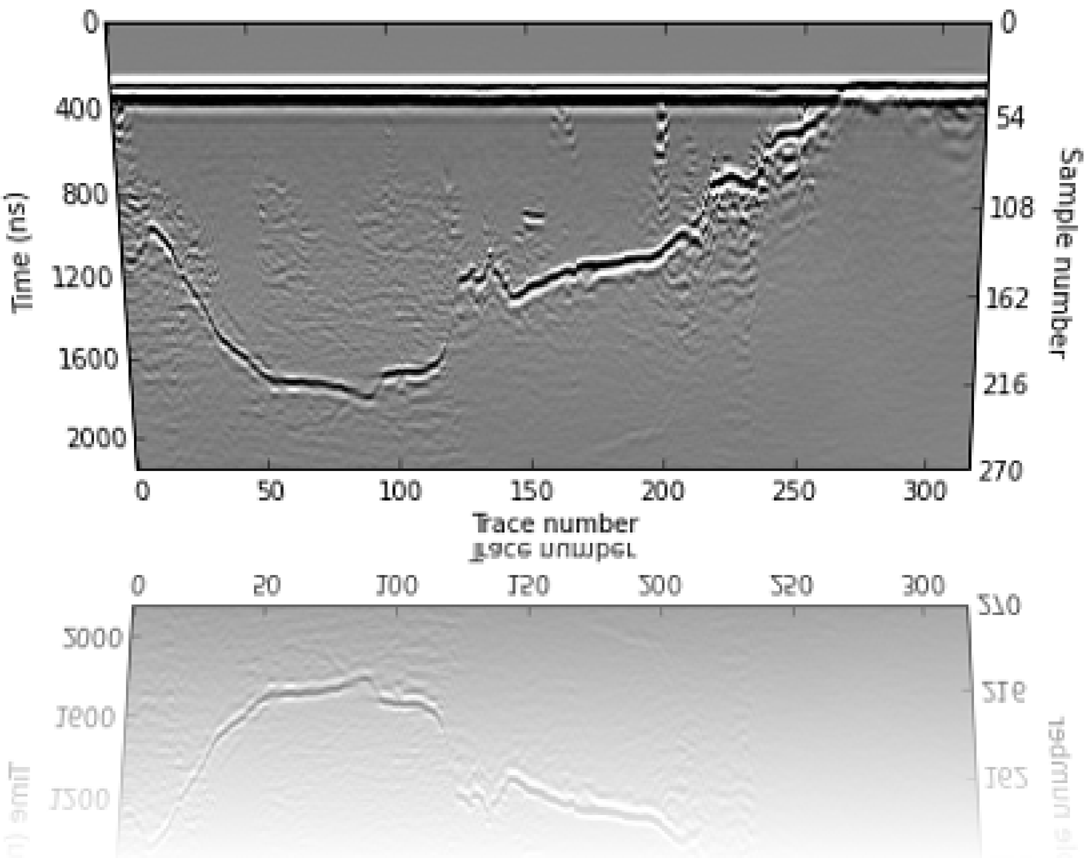
Nat Wilson

Jun 09, 2022

CONTENTS:

CHAPTER ONE

INTRODUCTION



radar_tools is a Python package and set of applications that I wrote in order to view and analyze radar data. I have made the package open-source (http://njwilson23.github.com/radar_tools/), and written this documentation to make it easier for others to use.

There are two sides to *radar_tools*. The first is the set of command-line and graphical utilities that perform common operations on ice-penetrating radar data. This includes concatenating datasets, projecting spatial coordinates, extracting metadata, viewing and filtering radar lines, and picking reflection wavelets. A tutorial following this half of the manual steps through a typical workflow for processing radar data. For the Python-doubtful, a script is included in this section that converts raw HDF datasets into matfiles for processing in MATLAB.

The second part of *radar_tools* is the `irlib` API, which is used by creating custom Python scripts. `irlib` serves the role of abstracting radar data into easily-used datastructures with built-in analysis functionality. Accessing the API directly is useful for experimental data exploration, filter construction, individualized plotting, and interfacing *radar_tools* capabilities with external Python scripts. The commandline and GUI utilities in *radar_tools* are themselves built on `irlib`.

Irlib was first developed prior to 2012 back in Python2 days. Since then Python 2 is no longer supported so everything from version 0.5 on works with Python 3. Some of the functions may be backward compatible but this has not been fully tested. If you need Python 2 functionality, please use and older version of irlib. Users may also wish to check out another open source suite of radar tools. See: https://impdar.readthedocs.io/en/latest/

The following documentation is as complete as possible. For the **beginner user** it starts with how to install the software and then progresses through some of the data handling utilities and then reviews how to pick radar returns. Following this, an **intermediate user** might wish to follow the in-depth tutorial. At the end of the manual there are topics for **advanced users and developpers**.

Beginner User - Introduction - Installation - Command-line Utilities - IcePick2 - icerate (optional)

Intermediate User - Tutorial

Advanced Users and Developpers - Changes in IEI h5 file format - Adding customized filters to the irlib GUI apps - Documentation - irlib API

CHAPTER TWO

INSTALLATION

2.1 Installation with Conda

Installing and setting up irlib is best managed in a conda environment. Follow the conda instructions followed by Linux or Windows instructions depending on your system. Mac should be similar to linux. The source code is on github at https://github.com/njwilson23/irlib

2.1.1 Setup a conda environment

These steps will set up and manage your Python environment and dependencies for irlib. For more on the dependencies themselves, see below.

1. Install Anaconda or Miniconda Python 3 64-bit
2. Open the *Anaconda Prompt* **as an administrator** or with **write permissions** to the conda directory and create an environment specifically to use irlib (Run one of these options):

   Use the environment file in the repository:

   ```
   conda create -n environment.yml
   ```

   This will be a bare-bones installation to run irlib:

   ```
   conda create -n irlib -c conda-forge python h5py scipy
   matplotlib cython geopandas
   ```

   As above but also installs vitables, an hdf viewer, and sphinx with numpydoc (for documentation):

   ```
   conda create -n irlib -c conda-forge python h5py scipy
   matplotlib cython geopandas sphinx vitables numpydoc
   ```

3. To run irlib you need to work out of a conda-aware console and type, *this must be done before every session*:

```
conda activate irlib
```

2.1.2 Make an irlib directory

To follow this example create a folder called 'py' in your own home directory and follow directions to add irlib files in it. However, you can install irlib wherever you wish (including within the conda environment folders). Once you have decided where to put irlib, change directory into that folder.

2.1.3 Download irlib

Go to https://github.com/njwilson23/irlib and download a zip of latest irlib and unzip it in that folder. Alternative: if you have git installed you can type the following in the terminal:

```
>> git clone git@github.com:njwilson23/irlib.git
```

This makes a directory in your home folder called py/irlib. Note: if you are going away from the internet, make a copy of the irlib zip file or directory for safekeeping. If you start messing around, it's good to get the original back without any fuss.

2.1.4 Set the operating system path

This step is not strictly necessary but needs to be done if you want to type the irlib command line executables from any folder (including where your data files are located).

LINUX

Instructions assume you are using Bash and you installed to the directory from the example above.

Find the hidden .bashrc file and open it in an editor. At the bottom of this file type and then save the file:

```
# Set path for irlib python scripts HERE
export PATH=$PATH:~/py/irlib-master
```

Then, in a terminal, type the following to make the change permanent:

`source .bashrc`

WINDOWS

To add path for the current session:

`set PATH=%PATH%;C:\your\path\to\irlib\code`

To permanently add path, but not for current session:

`setx PATH=%PATH%;C:\your\path\to\irlib\code`

To view you current operating system path:

`echo %PATH%`

Alternatively, on Windows, one should be able to modify the *Path* variable by right clicking on **My Computer** and going to *Properties -> Advanced System Settings -> Environment Variables*.

2.1.5 Set the conda environment path

This must be done so Python can find the irlib libraries when in the irlib conda environment.

- Activate the conda environment and ensure that the irlib files are always available. In a terminal type:

  `conda activate irlib`

  `conda develop [irlib code location]`

- You may need to restart conda or reactivate irlib environment for this to take effect.
- **NOTE** that if you are sharing your conda environment with other users they will be using that same version of irlib that you have specified!

2.1.6 Testing

Open a terminal, activate your irlib conda environment and type:

`h5_dumpmeta.py -h`

You should see the useage message starting like so:

```
usage:  h5_dumpmeta.py [-h] [-o OUTFILE] [-c] [-w] [-l] [--clobber]
[--swap_lon] [--swap_lat] infile
```

Then see if it works with an h5 file (in this example it is called 'survey.h5'):

`h5_dumpmeta.py survey.h5`

It will output some metadata to the screen.

If that doesn't work:

- check your conda environment is activated
- check your paths are set
- make sure that the python files are executable

2.2 Dependencies

In this section the main irlib dependencies are listed and discussed. *If you installed with conda as above you should have these dependencies already and you don't need to read this section.*

radar_tools is built upon a number of standard tools from the scientific Python ecosystem. The following are *required*:

- Python : Already installed for Linux/Mac OS X users
- Numpy : Basic array type, analogous to a matrix in MATLAB, except better
- Scipy : Wrappers for scientific libraries used for efficient filtering
- h5py : interface for HDF datasets
- matplotlib : Plotting library required for GUI tools
- pandas : Powerful Python data analysis toolkit
- geopandas : Python library that enables geopspatial data interchange.
- Cython : Python compiler for improving performance

Finally, these are *nice to have*:

- Sphinx : Documentation generator library.
- numpydoc : A sphinx extension containing styling.
- Vitables : An hdf viewer to look at the structure of h5 files more visually. (you can also use hdfview or another alternative)

2.3 Alternative installations

These instructions are based on older versions of irlib and have not been tested on version 0.5.

Using a package manager (e.g. APT, rpm, pacman, or Homebrew) download all the dependencies listed above.

The latest version is on Github. After downloading either directly or using the command

```
>> git clone git@github.com:njwilson23/irlib.git
```

Installation can be done with `pip`, a Python package manager.

```
>> cd irlib/    # or wherever it's downloaded to
>> pip install .
```

Assuming that dependencies are available (see above), this will take care of installing `radar_tools` properly.

To use the *pywavelet* wavelet transform algorithms, navigate to `irlib/external` and follow the directions in the `README` file, being sure to move the created file `pywavelet.so` to some place from which it can be imported.

Alternatively, *irlib* can be build in place without `pip` by doing

```
>> python setup.py build_ext --inplace
```

For convenience, programs that make up *radar_tools* should be on the execution `PATH`. If `pip` was used, this should be taken care of. Otherwise, follow instructions in section 2.1.4 above.

CHAPTER

THREE

COMMAND-LINE UTILITIES

The command-line utilities in *radar_tools* are useful for performing data management and pre-processing tasks on HDF radar datasets, as well as for performing basic data exploration and conversion tasks.

In general, typing any of the utilities without arguments yield invocation and usage instructions that are printed to the screen. This section summarizes individual tool's functionality.

3.1 Recommended data cleaning workflow

The following steps are very helpful for data cleaning and streamlining workflow. Some of the steps are prerequisites for subsequent analyses, so **do this in the correct order**. It is really very important that you **take notes on what you did so that your workflow can be recreated later**. It is recommmend you open a document and copy paste what you did from the terminal in there for safekeeping. Also, you can copy the screen output there too. As you go be aware that some scripts will overwrite files. Recommend that you use unique file names that represent the step that you just completed.

- `h5_dumpmeta.py` : examine metadata, visualize on a GPS
- `h5_consolidate.py` : combining h5 files streamlines workflow, take notes of the lines you want to work with
- `h5_replace_gps.py` : if you have better GPS data use it to refine position
- `h5_add_utm.py` : allows irlib to calculate distances easier using Cartesian coordinates
- `h5_dumpmeta.py` : check that all is well by comparing to earlier metadata
- `h5_dumpmeta.py` : generate caches to speed up data access and do some more metadata

Once this has been been completed the data is ready to be used for ice thickness determination.

3.2 Data management

3.2.1 h5_consolidate

h5_consolidate combines multiple datasets into a single dataset. In the process, lines are re-numbered so that they stay in sequential order. Concatenating datasets is useful, for example, to combine multiple surveys collected on different days into a single file that is easier to manage (but larger).

> SYNTAX: h5_consolidate.py [-h] [-o OUTFILE] [infile . . .]
>
> > Combines multiple datasets (>1) into a single concatenated dataset.

You can list the infiles you want to combine in order or use a wildcard.

3.2.2 h5_replace_gps

If GPS data collected from the on-board receiver are missing or of poor quality, they can be replaced by data from a hand-held GPS receiver. The data from the hand-held receiver must be exported as or converted to GPX format, which is a standard open format. Calling `h5_replace_gps` creates a copy of the original dataset with the new coordinates inserted. Command-line flags can be used to specify matching tolerances and which lines to work on.

SYNTAX: h5_replace_gps.py [-h] [-t TZOFFSET] [-l LINE] [-d DELTATIMEMAX] [-o OFFSETELEV] [-n] [-p] infile outfile gpsfile {gpx,ppp} {iprgps,iprpc,both}

This tool replaces the existing geographical data in a ice radar HDF database with data taken from a GPX file, e.g. obtained from a handheld or external GPS unit or from a CSV file, e.g. obtained from a PPP output of GPS data

Positional arguments:

-infile	input HDF (.h5) filename, with or without path, for which GPS or PC timestamps exist
-outfile	output HDF (.h5) filename, with or without path, if this file exists, it will be overwritten
-gpsfile	GPS filename(s), with enhanced location, with or without path / wildcards
-gpx_ppp	Select which format the gps file is in - either gpx or ppp
-iprgps_iprpc_both	Select which timestamp to match gps timestamps to: iprgps (recommended), iprpc (if iprgps not available) or both (use caution)

Optional arguments:

-t hh	The hour offset (hh) of the GPR computer from UTC (default = 0)
-l n	Work only on line (n); default works on all lines
-d n	Set the max time delta permissible for matching locations to (n) seconds; default is 15 seconds
-o n	Adds an offset (n) to the elevations to account for the height of GPS off the ice or different geoid, use a neg. number to subtract.
-n	Replace coordinates in HDF with no appropriate supplementary GPS counterpart with 'NaN'. By default, the original coordinates are retained.
-p	Keep all coordinates positive (use with old h5 format where coordinates are Lat_N and Long_W).

3.2.3 h5_add_utm

h5_add_utm uses the *pyproj* library to append projected UTM zone coordinates to datasets that only include lon-lat coordinates. This is a required step for many of the data processing operations that might be used later.

SYNTAX: h5_add_utm.py [-h] [–swap_lon] [–swap_lat] infile outfile

Replaces geographical coordinates in INFILE with UTM coordinates in OUTFILE. Does not perform any datum shift. Projection is calculated assuming that the data from neither from western Norway nor Svalbard.

The UTM zone is calculated based on a naive algorithm that is ignorant of the exceptional UTM circumstances in the vicinity of western Norway and Svalbard.

Works with 2 formats from BSI HDF files:

Old format -

Latitude and longitude data in BSI HDF files are unsigned. It is assumed to be in the western hemisphere by default. Passing the –swap_lon key forces longitudes to be interpretted from the eastern hemisphere. UTM projection is calculated assuming that the data from neither from western Norway nor Svalbard.

New format -

Latitude and longigude data in BSI HDF files are signed to indicate hemisphere. If any lat or lon values are negative, the –swap_lon key is disabled

3.2.4 h5_generate_caches

When it is time to do your picking this step will make the data handling more efficient. There are also some important data cleaning steps here that are helpful.

SYNTAX: h5_generate_caches.py [-h] [-d DIR_CACHE] [-r REMOVE_WITHIN] [–dc DC] [-n] [-i] [-s] [-b] [-g] [-f] [-q] [-v] infile

-d [DIR] cache directory (default: cache/) -g fix static GPS issues -s smoothen coordinates -b remove blank traces caused by triggering failure -r remove stationary traces by averaging all traces within # m (defaults to 0 m or off), recommend 3 for L1 GPS -f force regeneration of existing caches -q silence standard output -e print failed datacaptures -dc=[#] specify datacapture (default: 0) -n remove traces with NaN coordinates -i interpolate over NaN coordinates (overrides -n) -v print failed datacaptures

Caching improves performance and is a very good idea. `h5_generate_caches` creates caches (`.ird` files) for every line within a survey, and optionally applies a number of pre-processing steps to the data:

- **static gps correction**: attempt to recognize period when the GPS was in "static mode", and interpolate continuous positions.
- **smoothen coordinates**: filter noisy position data
- **remove blank traces**: exclude empty soundings from the cache
- **remove stationary traces**: attempt to recognize period when the radar sled was motionless, and remove redundant soundings

`h5_generate_caches` should be the last of the data management scripts to run, because modifying the original HDF dataset won't affect the caches until they are regenerated.

3.3 Exploration and conversion

3.3.1 h5_dumpmeta

h5_dumpmeta exports the radar metadata to a CSV file or a shapefile. The actual sounding data is not included.

SYNTAX: h5_dumpmeta.py [-h] [-o OUTFILE] [-c] [-w] [-l] [–clobber] infile

Positional arguments:
-infile input HDF (*.h5) filename, with or without path, if you use wildcards in linux, put this in quotes

Optional arguments:

-o	output file BASENAME [if missing, will be automatically generated]
-c	create csv metadata file
-w	create a waypoint metadata shapefile
-l	create a line metadata shapefile
--clobber	overwrite existing files

3.3.2 h5_export

h5_export.py exports a line from HDF5 to an ASCII, REFLEX or BINARY file.

SYNTAX: h5_export.py [-h] [-o OUTFILE] [-l LINE] [–clobber] {ascii,binary,reflex} infile

Positional arguments:

-ascii_binary_reflex	Select which format to export to - either ascii, binary or reflex
-infile	input HDF (.h5) filename, with or without path

Optional arguments:

-o OUTFILE	output filename, basename only NO extension; defaults to infile
-l LINE	line number to export - defaults to all
--clobber	overwrite existing files

3.3.3 h52mat

h52mat converts HDF data to a MATLAB `.mat` file. The filters from `h5_generate_caches` are available. For those who prefer MATLAB, the rest of this document can be ignored.

SYNTAX: h52mat SURVEYFILE OUTFILE [options]

SURVEYFILE is the HDF5 file generated by IceRadar. OUTFILE is the anme of the *.mat file to be generated.

Options:

-g	fix static GPS issues
-s	smoothen coordinates
-b	remove blank traces (trigger failure)
-r	remove stationary traces
-o	overwrite
-q	silence standard output

3.4 Thickness Determination

Once Data Management and Exploration and Conversion steps have been completed, the process of thickness determination can begin.

3.4.1 icepick2

icepick2 allows for interaction with radargrams. See chapter 4 for full description.

SYNTAX: icepick2 <HDF_survey> [-L line_number]

3.4.2 mergepicks

This script allows users to reprocess older picks if, for example, the preprocessing steps were changed. The script goes through the FIDs for the new (reprocessed) input h5 file and creates picking files that include the older picks.

SYNTAX: mergepicks.py [-h] [-d DIR_CACHE] [-n] [–dc DC] infile outdir oldpicks

Positional arguments:

-infile	input HDF (.h5) filename
-outdir	subfolder where new picking files will be written
-oldpicks	folder where old picking files are found

Optional arguments:

-d	cache directory, default: cache/
-n	will priviledge new picks over old picks in case of conflict
--dc	specify datacapture, default: 0

3.4.3 joinradar

join_radar combines information from picking, rating, offset, and HDF5 files, and computes ice thickness at each valid observation location. You must have a subdirectory 'picking' to run this script If there is no rating directory, all picks will be processed with a rating of '-9' If there is a rating directory, ONLY lines with ratings will be processed. If there is no offsets directory, you can specify –offset that will be applied to all traces

Caution – This script will overwrite files in the results subdirectory.

SYNTAX: join_radar.py [-h] [-v VELOCITY] [-q QUAL_MIN] [-c] [-w] [-o OFFSET] [-n] infile

Positional Arguments:
infile input HDF (*.h5) filename, with or without path

Optional Arguments:

-v VELOCITY	radar velocity in ice, defaults to 1.68e8 m/s
-q QUAL_MIN	the minimum rating value to include 1 to 5 (defaults to -9, which signifies unrated picks)
-c	create csv file with fid,lon,lat,elev,thickness,error
-w	create a waypoint shapefile
-o OFFSET	if no offsets directory exists, provide antenna offset (m) for all traces
-n	remove any trace that has no thickness data

3.4.4 icerate

icerate is a tool that evaluates the quality of picks, see chapter 5 for full decription.

SYNTAX: icerate -f file_name [-L line_number] [–pick pick_filename]

CHAPTER

FOUR

ICEPICK2

IcePick2 (added in v0.4) is a single tool for browsing, processing, and picking radar data. Prior to this, the seperate tools *irview* and *icepick* performed these tasks.

IcePick2 is started from the command line using the syntax

```
icepick2.py -f HDFNAME [-L LINENO]
```

where `HDFNAME` is the name of the radar survey file and `LINENO` is an optional line number (default `0`).

Upon launch, the terminal window is converted into an *IcePick2* console, and a Radargram window (B-scan) is opened automatically for the current file and line. Clicking on the Radargram shows information about the trace and depth clicked.

Commands can be typed in the console. Typing `help` gives an idea of the options. Examples include:

- `info` provides more detailed information about the currently opened line.
- `open LINENO` allows different lines to be viewed without restarting *IcePick2*
- `ylim T0 TF` adjusts the vertical (time) range shown, where T0 and TF are specified in nanoseconds
- `exit` and `q` both close *IcePick2*.

4.1 Windows

The console is the root of the IcePick, however data are displayed and interacted with in various windows. Windows are generally launched (or closed) by typing the name of the window followed by `on` or `off`. For example,

```
pick on
```

opens a window displaying a set of individual radar traces, which can be clicked with the mouse for picking, and

```
map on
```

opens a simple map of the current line.

4.1.1 Radargram

The Radargram, opened by default, is the primary graphical window, and shows the `Gather` data and annotations. The Radargram window can also be used for digitizing features, such as englacial scattering (EXPAND HERE…)

4.1.2 PickWindow

The PickWindow reproduces the functionality of *icepick.py*, prior to v0.4. A series of radar traces are shown, and can be changed by pressing the **h** and **l** (ell) keys to move up or down the radar line. A set of yellow lines in the Radargram shows which traces are shown by the PickWindow.

The process of picking is fairly simple. Click the mouse on the part of the trace representing a reflection to be timed. Right-clicking removes the pick if you've made a mistake. Fine adjustments can be made by pressing the **j** (down) and **k** (up) keys.

The "picking mode" can be switched between bed-picking and airwave-picking by pressing the middle mouse button.

Automatic picking can be performed by typing `pick dc` and `pick bed`, for the direct-coupling and bed reflection, respectively. A pair of optional arguments can be appended to these commands, i.e.

```
pick bed [min, max]
```

which uses the integers `min` and `max` as constraints in identifying the reflection. This works well where the reflection is clear and the radargram is clean, but will typically require manual clean-up.

Once the picks are satisfactory, type `pick save` to save the timing data to the folder `picking/`.

4.1.3 MapWindow

The MapWindow shows the current line as a series of dots shown on a Mercator projection.

4.2 Filters

In addition to viewing raw data, *IcePick2* provides access to pre-defined filters on the fly using the `f` command (mnemonic "filter"). For example, typing:

```
f gc
```

applies a linear gain control operator to each trace, increasing the amplitude of later-arriving events. This modifies the data in memory, and **it does not alter the data in the HDF file**. Help for individual filters is obtained by typing `help [filtername]`. In order to determine what filters have been applied to a dataset, typing `f` alone lists them, along with any parameters required to reproduce the presently displayed data. The data can be reset to the version originally loaded by typing `nf` (mnemonic "no filter").

The presently defined filters include common operations such as time-domain and frequency-domain filtering, dewow, linear and automatic gain control, F-K migration, and instrument ringing removal. Defining custom filters can be done by constructing a subclass of irlib.app.filters.Command. The operation performed by the filter is contained in the `apply()` method, and can include *irlib* calls, or any other Python manipulation of the `Gather` data.

Non-comprehensive list of filters

Command	Description
Gain Control	
gc	Applies a linear gain enhancement
agc	Applies an automatic (nonlinear) gain enhancement
Convolutions	
dewow	"Dewowing" filter to remove instrument drift
lowpass	Performs a frequency-domain lowpass filter with a cutoff frequency of 25 MHz
highpass	Performs a frequency-domain lowpass filter with a cutoff frequency of 25 MHz
lowpass_td	Performs a time-domain lowpass filter (moving average)
highpass_td	Performs a time-domain highpass filter (inverted moving average)
Recursive	
iir30low	Chebyschev lowpass filter with cutoff at 30 MHz
iir25high	Chebyschev highpass filter with cutoff at 25 MHz
Migration	
migfk	Stolt (F-K) migration
Misc	
abs	Displays the absolute value of the data
wiener	Wiener statistical noise filter
ringing	Horizontal ringing filter based on singular value decomposition
project	Project radar line to straight segments with equal trace spacing

A comprehensive list is provided by typing `help` with no arguments.

4.3 Caching

The performance of *icepick2* can be enhanced substantially by pre-caching of the radar lines. This can be done using the API (`Gather.Dump()`), or by running the commandline utility `h5_generate_caches` (discussed previously). Any filter can be applied at the time of cache generation. Caches are Python "pickles" (serialized data), and contain a snapshot of the radar data, as well as a reference to `irlib`. Substantial changes to `irlib` may require cache regeneration.

4.4 Recommended IcePick2 workflow

Below is a recommended workflow for IcePick2 which is to be used with command line utility cleaned data (previously introduced). This approch can be altered to fit specific needs by adding additional commands, but this is a good place to start.

- Launch IcePick2: `` `icepick2.py ipr_survey.h5` ``
- Open each line one at a time and and follow the below workflow: `` `open 1` ``
- Turn on the PickWindow: `` `pick on` ``
- **Auto pick DC: `` `pick dc uppersample# lowersample#` ``**
    - check for and correct any errors
- **Apply filters**
    - find the best combination for optimal visibility (see filter options above)

- **Pick bed using the PickWindow**
  - Can auto pick as was done for DC, however, it is much less accurate
- Save picks: `` `pick save` ``
- It is suggested that you take notes while picking to ease interpretation later

CHAPTER FIVE

ICERATE

icerate is a tool for rating the quality of picks before surface interpolation. The interface is similar to *IcePick2*, although missing a number of features.

When working in a previously picked file, open icerate window:

```
$ icerate.py -f survey_ppp_utm.h5
```

Open the line you wish to work in:

```
open 1
```

Enter a number from 1-5 to assign a rating that corresponds to quality of the pick displayed in the window. These ratings are subjective evaluations that are used to quantify the certainty of each pick.

Rating	Approximate Error
5	1.4 m
4	1.7 m
3	2.2 m
2	3.5 m
1	7.1 m

Once the rating of the selected line is complete, save the rating:

```
save
```

Once saved this ratings can be found in “rating/“.

CHAPTER SIX

TUTORIAL

Note: This tutorial, while helpful, is not necessary for operating irlib. For a basic introdution to irlib and its workflow refer to Command-line Utilities and IcePick2. This extra information is helpful for intermediate and advanced irlib users.

This tutorial is for an older version of irlib. The same workflow still applies, with the difference being that picking is performed in IcePick2. I have adjusted the text where needed, but some inconsistencies may have been overlooked.

I am going to use the radar data collected in Spring 2012 to create a bed map for Glacier 3 ("East Glacier"). I'll document everything I do here, so that this can serve as a step-by-step tutorial on on using the radar interpretation tools. I'm working from a Linux computer, so adapt terminal commands as necessary.

The radar codes are subject to change. I'll be using a current git revision as of February 13, 2013 (`commit 3dc4638de82867b58c40cd19727d3cfd980112f6`). Most likely, it will be best to use the most recent version available. Furthermore, I'll be using gstat for interpolation.

6.1 Set-up

6.1.1 File structure

I've created a directory to work from. I named it `gl3bedmap`, and I added a subdirectory called data that contains a copy of the raw radar data. Create subdirectories called `cache`, `picking`, `rating` and `offsets` too, as they'll be needed later. The directory tree should now look like:

```
gl3bedmap:
    data:
        gl3_radar_may14.h5
        gl3_radar_may16-17.h5
    cache:
    picking:
    rating:
    offsets:
```

6.1.2 Concatenating datasets

The field data are contained in two HDF5 datasets. Since they both come from the same glacier and the same field campaign, it would be nice to just deal with one file. This is what the *h5_consolidate.py* tool is for. From the `data` directory:

```
h5_consolidate.py gl3_radar_may14 gl3_radar_may16 -o gl3_radar_2012.h5
```

which creates a single dataset with all of the data. The line numbers in all but the first dataset are changed to avoid repeats. If I were to type the name `h5_consolidate.py` by itself, it would print out a reminder of how it works.

6.1.3 Viewing the metadata

It's convenient to be able to read the data we're dealing with. There are a number of ways to do this. HDF software comes with a number of utilities, including *h5dump* and *hdfview*. I find these good for looking at raw data, but the output can be a bit clunky to wade through. Therefore, I wrote *h5_dumpmeta*, which parses each radar sounding and writes a CSV file that can be opened quickly in MATLAB (or as a spreadsheet).

```
h5_dumpmeta.py data/gl3_radar_2012.h5 > data/gl3_radar_2012_metadata.csv
```

Note that this is just the metadata, and doesn't actually print out the receiver sled's digitizer readings.

6.1.4 FID Interpretation

Every row in the metadata is assigned an identification (FID) that is unique within the file. The FID is a sixteen character string.

Characters	Meaning	Notes
1-4	Line	Collections of traces
5-8	Trace	Individual soundings
9-12	Datacapture	Used for channels in dualdar
13-16	Echogram	Presently unused

Care should be taken to preserve the FID in all of the files that follow, because they will be important for matching the pieces of data properly.

6.1.5 UTM coordinates

As can be seen by viewing the data in one of the ways above, the radar data contains geographical longitude and latitude coordinates. It's frequently easier to work on a projected coordinate system, so I would run

```
h5_add_utm.py data/gl3_radar_2012.h5 data/gl3_radar_2012_utm.h5
```

to create a copy of the file with UTM coordinates appended. Repeating the previous step, it can be seen that the *eastings*, *northings*, and *zones* columns are now populated. **This assumes that you've installed pyproj** (see *Dependencies*).

- Note: `h5_add_utm` is pretty dumb about UTM zones. It works fine most places, but if you're working in SW Norway or Svalbard, there are exceptions to the normal 6° grid and you might need to tweak the code.

6.1.6 Pre-caching (optional)

Finally, as an optional step, I'm going to generate a cached copy of the radar data. This will speed things up while picking, and allows me to do some initial preprocessing to remove bad radar soundings, etc.

```
h5_generate_caches -d cache -g -b -r --dc=0 data/gl3_radar_utm.h5
h5_generate_caches -d cache -g -b -r --dc=1 data/gl3_radar_utm.h5
```

This creates a preprocessed binary copy of each line in the directory `cache` for which

- fixes for when the GPS was in "car-mode" (static GPS) have been attempted
- repeated soundings in the same location have been filtered out
- blank traces from caused by triggering errors have been removed

The `--dc` switch specifies which channel to operate on, which is only important for the "dualdar" system (otherwise it should be `0`, which is the default).

To see what preprocessing options are available, type `h5_generate_caches` without any argument.

6.2 Ice thickness picking

6.2.1 Basics

Picking is the process of measuring the time before the arrival of each interesting return wavelet. Picking is labour-intensive, although I tried to automate the easy parts.

To get started, run

```
icepick2 -f data/gl3_radar_utm.h5
```

and then type `pick on` in the console.

The way this works is that the window that opens shows a grey-scale *radargram* in the one panel, with eight individual traces in the other panel. The location that the traces are from is shown by the vertical yellow lines in the Radargram. Assuming there are more than eight traces (normally the case), the display can be panned across the Radargram with the **h** and **l** (ell) keys. In my case with the 2012 radar data from Glacier 3, the first line only contains a single trace, so panning doesn't do anything.

The terminal in which `icepick2` was launched now accepts icepick-specific commands. Typing

```
info
```

gives information about the current line. For this data, it tells me

```
data/gl3_radar_2012_utm.h5
line: 0
# traces: 1
# samples: 256
sample interval: 4e-09 s
depth resolution: 0.336 m
vertical range: 86.016 m
pick-mode: bed
```

From top to bottom, this tells me what file I'm operating on, the line number (starts at 0, as in the HDF dataset), the number of traces (`nx`), the number of samples per trace (`nz`), the sampling interval, and estimates of the vertical resolution and the maximum depth imaged, assuming the material is ice. The final line, `pick-mode`, indicates that any picks we perform now are for the glacier `bed` (more on that in a moment).

Typing

```
help
```

gives a (potentially non-exhaustive) list of valid commands. To switch to line #1, type

```
open 1
```

The process of picking is fairly simple. In the lower panel of the icepick window (where the individual traces are shown), click the mouse on the part of the trace representing a reflection to be timed. Right-clicking removes the pick if you've made a mistake. Fine adjustments can be made by pressing the **j** (down) and **k** (up) keys. Whenever the side-scrolling keys are pressed (**h** and **l**), a line representing the picks is drawn on the radargram. Presumably, the bed should be picked on every trace where it can be identified.

Once the picks are satisfactory, type `save` to save the timing data to the folder `picking`.

6.2.2 Filtering

There are a number of filters that can be applied with the `f` command, using the syntax

```
f FILTERNAME
```

Some common filter names are:

- `dewow`: applies a "dewowing" highpass filter
- `lowpass`: applies a generic frequency lowpass filter
- `lowpass_td`: applies a generic time-domain lowpass filter
- `gc`: applies a linear gain control
- `agc`: applied a nonlinear automatic gain control (usually more fun than useful)
- `migfk`: performs F-K (Stolt) migration, and takes a sample number as an optional argument indicating time zero (the airwave)

Furthermore,

- Typing `f` without an option lists the filter history, so you can see exactly how the current data has been modified.
- Typing `nf` undoes all filter effects (except for those that happened during cache-generation or automatically when loading the line), and restores the original data.

The following is out of data, deprecated - use `irlib.components.filters` instead

There are lots of other filters. All filters are defined in the file ``filter_defs.py``, which is in the place where ``irlib`` is installed. Modifying this file permits custom filters to be defined.

A final adjustment is `gain`, which adjusts the display contrast of the radargram. All filters accessed through `f` or `gain` are reversible, so there is no risk of permanently damaging the data by experimenting.

6.2.3 Direct coupling

In order for timing data to be generated, a reference time must be known. Because it's not easy for us to know the exact time that the transmitter emitted a pulse into the ice, we use the airwave as a timing reference. The airwave travels directly from the transmitting antennas to the receiving antennas at the speed of light ($\approx 3 \times 10^8$ m s^{-1}, so the emission time can be calculated by knowing the airwave arrival time.

To switch to direct-coupling mode, type press the middle mouse button (button 2) on the PickWindow, and a label should appear in picking window indicating the mode change. All picks made in `dc` mode will have a red dot rather than blue.

To change back to bed mode, press the middle mouse button again.

6.2.4 Automated picking

To save time, picking can be done automatically. For example, to automatically pick the airwave across the whole radar line, use the `pick dc` command. If me know that the airwave is between samples 75 and 125 (right vertical axis on the radargram), then we can give this as a hint by typing

```
pick dc 75 125
```

icepick2 then uses a set of heuristics to try and figure out where the airwave is in each trace, subject to the vertical constraints.

- There is a minimum vertical range for the algorithm to work. I forget what it is, but it's something around 20. If `autodc` doesn't work, try increasing the range arguments.

Automatically picking the airwave usually works pretty well. Automatically picking the bed reflection is more hit-and-miss. The command `pick bed` works pretty much the same way as above, and usually does a decent job when the radargram is very clear. Even when the radargram is more complicated, I usually give `pick bed` a shot, and then go through making the (many) necessary corrections.

6.3 Pick rating

Rating is used to quantify the certainty of each pick. I use the following rating table

Rating	Approximate Error
5	1.4 m
4	1.7 m
3	2.2 m
2	3.5 m
1	7.1 m

Ratings could be tabulated manually. For efficiency, I use a program similar to *icepick*

```
icerate -f data/gl3_radar_2012_utm.h5
```

but this program is not polished to the same standard as *icepick* and *irview*.

6.4 Ice thickness calculation

6.4.1 Antenna spacing

A last ingredient before ice thickness can be calculated is an *offsets* file, which contains information about how much antenna spacing there was for each line. Hopefully this information is contained in field notes. Then run:

```
antenna_spacing data/gl3_radar_utm_metadata.csv 60
```

The first parameter is the CSV created previously with `h5_dumpmeta` and the second is antenna spacing in meters. This creates `offsets/gl3_radar_2012_utm_offsets.txt` containing FID (see *FID interpretation*) and antenna spacing.

6.4.2 Data join

Calculating ice thickness is fairly trivial, so the only challenge is in properly integrating all of the data. The steps are:

- Take all soundings for which both a pick and a rating exist
- Find the proper antenna spacing
- Assuming an ice velocity, calculate reflector depth with the Pythagorean theorem

I use the script `join_radar.py` to do all of this.

```
python join_radar.py gl3_radar_2012_utm data/gl3_radar_2012_utm.h5
```

which should generate a file containing data similar to:

fid	longitude	latitude	altitude (m)	depth (m)	error
0000000000000000	6.339396	59.942123	1187.9	170.08	3.125
0000010100000000	6.339395	59.94209	1186.0	170.76	3.125
0000016300000000	6.339312	59.942139	1186.7	170.76	3.125
0000016500000000	6.33919	59.942217	1187.4	176.9	3.125
0000016700000000	6.339072	59.942306	1188.3	178.26	3.125
0000016900000000	6.338967	59.942402	1188.9	180.99	5.55555555556
0000017100000000	6.338861	59.942494	1189.3	189.84	5.55555555556
0000017300000000	6.338745	59.942603	1190.4	201.39	5.55555555556
0000017500000000	6.338674	59.942708	1191.1	210.9	5.55555555556
0000017700000000	6.338608	59.942817	1191.9	220.4	3.125
0000017900000000	6.338557	59.942926	1192.9	234.63	3.125
0000018100000000	6.338486	59.943049	1194.0	234.63	3.125

6.5 Raster interpolation

The general interpolation scheme is discussed in *Interpolation*. A brief description and the commands I used to generate a bed map are given below.

6.5.1 Mask file

I generate a mask covering the area of Glacier 3 based on the outline traced from satellite imagery. This provides a domain for the interpolation scheme. Using the outline shapefile from *Outlines*:

```
gdal_rasterize -of GTiff -a id -tr 20 20 -te 606100 6757400 611100 6760800\
               -l outline_gl3 outline_gl3.shp mask_gl3.tif
gdal_translate -of AAIGrid mask_gl3.tif mask_gl3.asc
```

6.5.2 Data concatenation

Since ice thickness needs to be zero at the glacier margin (assuming no cliffs or steep bulges), I append the depth sounding data generated *above* with samples taken from the glacier margin. I produced the margin file using a GIS, and prescribed a depth of 0 m and a variance of 0.1 m at every point (*gstat* doesn't like zero uncertainties). Then,

```
cat depth_gl3_radar_2012_utm.xyz gl3_outline_100m.xy > \
    kriging/gl3_depth_outline_2012.xyz
```

6.5.3 Variogram estimation

I created a proto-*gstat* configuration file called `gl3_12_2p.gst` and containing the lines:

```
data(gl3): 'depth_outline_2012.xyz', x=1, y=2, v=3, V=4, d=2, \
           average=1, max=100, radius=1000;
set zero=20;
```

The first line creates a datasource from the concatenated ice thicknesses, and indicates that the columns correspond to x and y spatial coordinates, the interpolated value (v), and the variance (V), respectively. The argument `d=2` assumes a quadratic trend, `average=1` permits averaging of points that are very close, `max=100` sets a maximum number of observations for each interpolated point, and `radius=1000` sets a maximum search neighbourhood.

The second line declares that points within 20 m are indistinguishable from each other.

Running this

```
gstat gl3_12_2p.gst
```

opens an interactive *gstat* session, from which variogram estimates can be saved. I assume that, because Glacier 3 is roughly east-west oriented, the variogram should be split into east-west and north south components, and I save a variogram estimate for each.

6.5.4 Model variogram fitting

Variogram fitting can be performed in *gstat*, but I use a Python script (`fit_variogram.py`) because it gives me more control over the fitting routine and is more suited for anisotropic variograms than the built-in tools.

Once a suitable model variogram has been found, the *gstat* configuration file can be modified:

```
variogram(gl3): 1439 Sph(1043.8, 90, 0.4428) + 514 Sph(151.7, 0, 0.9750);
mask: 'mask_gl3.asc';
predictions(gl3): 'predictions/pred_gl3_12.asc';
variances(gl3): 'variances/var_gl3_12.asc';
```

Running this again will perform the interpolation. See the *gstat* manual for details.

CHAPTER

SEVEN

CHANGES IN IEI H5 FILE FORMAT

The following is a brief overview of the various different hdf file formats that have come about from different versions of the IceRadar Software for acquiring BlueSystem Radar data.

These modifications have broken irlib code in the past but version 0.5 works-around these format differences.

An easy way to examine the format of the h5 file you have is to look at it with an hdf viewer like vitables or hdfview.

Version 6.2 IceRadar (Mar 2022) – Not supported by irlib version 0.5

An HDF format version number will be available. Metadata tags used to have spaces within them. This is now cleaned up. For more changes see: https://iprdoc.readthedocs.io/en/latest/X.IceRadar_software/#file-attributes

Version 5.1 IceRadar (Sept 2016)

Added correction for lat and lon such that the western and southern hemispheres are negative numbers
Lat and Long are the field names and the values are floating point numbers

PCSavetimestamp attributes contains the same string as before but the timestamp is before the GPSCaptureEvent_StartBufferCapture.ms field. The format is dd/mm/yy_HH:MM:SS am/pm

Version 4.4.1 IceRadar: Format of PCSavetimestamp string is:

With no GPS used: GPSCaptureEvent_StartBufferCapture.ms:-99,BufferCaptureTime.ms:264,PPS_NO
With standard GPS reading: GPSCaptureEvent_StartBufferCapture.ms:72,BufferCaptureTime.ms:336,PPS_NO
With PPS GPS reading: GPSCaptureEvent_StartBufferCapture.ms:72,BufferCaptureTime.ms:336,PPS_YES

Importantly, in some cases the actual timestamp is saved as a comment field. The format is dd/mm/yyyy_HH:MM:SS am/pm

Version ? IceRadar: (ca. 2009)

lat and lon were both recorded as positive integers in fields Lat_N and Long_W. If either the lat and lon are negative numbers, then irlib assumes that this paradigm is not in effect. The user can use –swap_lon and –swap_lat to change the sign of either lat and lon in h5_add_utm.py and –positivecoords in h5_replace_gps.py

Computer time stored as PCSavetimestamp: mm/dd/yyyy_HH:MM:SS am/pm like “3/12/2014_11:49:20 AM” There are no GPSCapture stats.

Version ? IceRadar: (ca. 2008)

Computer time stored as “Save timestamp”

CHAPTER

EIGHT

ADDING CUSTOMIZED FILTERS TO THE IRLIB GUI APPS

icepick dynamically loads commands and filters when it starts. This happens in the lines in the `icepick2.py` file that look like:

```
console.register(irlib.app.filters)
console.register(irlib.app.pickcommands)
console.register(irlib.app.mapcommands)
```

Adding custom filters has two parts:

1. writing new `Command` classes and placing them in a Python module
2. registering the module

8.1 Writing a custom Command module

Terminology:

In traditional object-oriented programming, we talk about **classes** and **instances**. An instance is a bundle of data, while a class is a category of instances. Classes can inherit from other classes, receiving some of their attributes. As an example, in real life an `Apple` class might be a subclass of a `Fruit` class, and the apple sitting on my desk is an object, or a specific instance of an `Apple`.

In irlib, commands are represented by classes, and you can think of an instance being created and used whenever you run one.

Commands are defined by creating a Python class definition that inherits from `irlib.app.commands.Command`. If our command is a filter, it's better to inherit from `irlib.app.filters.FilterCommandBase`, which is a subclass of `Command`.

The command class has two important attributes and one important method: - The `cmd` attribute is a string that defines the command that runs the filter - The `helpstr` attribute is a string containing a description of how the command is used and what it does. - The `apply(G, args)` method is a function that takes a radar `LineGather` instance and a list of `args` and does something to it. This is the part that makes the command do something.

For example, the linear gain control filter (invoked with `gc`) is defined as (with annotations added):

```
class LinearGainControl(FilterCommandBase):
    # these are the two methods that make the command usable and give it
    # help documentation
    cmd = "gc"
    helpstr = """Linear gain control

    gc [n]
```

(continues on next page)

(continued from previous page)

```
    Apply a time-dependent linear gain control (t^n) to each trace, with the
    exponent `n` taking the default value of 1.0. """

    # This is the method that performs an action
    #
    # The first argument (called `self` out of tradition) is a reference to the
    # calling instance, and can be ignored.
    # G is a LineGather instance, which is the irlib object that holds the radar
    # data from a single line
    # `args` is a list. If we typed "gc 1 2 3" in icepick, args would be [1, 2, 3]
    def apply(self, G, args):
        # args may be an empty list, or it may contain a number used to set the
        # exponent on the gain filter.
        if len(args) > 0:
            npow = float(args[0])
        else:
            npow = 1.0
        # This filter simply calls the LineGather method that implements gain
        # control. We could do anything here, however.
        G.DoTimeGainControl(npow=npow)
        return
```

Here is an example of a filter that, for illustrative purposes, reverses the radar wave polarity and sets a maximum voltage.

```
# saved as myfilters.py

import numpy as np
from irlib.app.filters import FilterCommandBase

class ReversePolarityAndCap(FilterCommandBase):
    cmd = "toy_filter"
    helpstr = """Toy filter illustrating how to build commands

    toy_filter arg1 arg2
    """

    def apply(self, G, args):
        for arg in args:
            # Print each argument
            print("argument: " + str(arg))

        # Reverse wave polarity
        G.data = -G.data

        # Clip data to a maximum and a minimum value
        G.data = np.clip(G.data, -1.0, 1.0)
        return
```

You can place as many custom commands as you like in `myfilters.py`.

8.2 Registering the commands

At the moment, `icepick2.py` needs to be modified directly to add new filters. So to use the command in `myfilters.py`, we would first import it at the top of `icepick2.py`:

```
# [other import statements]
import myfilters
...
```

and then we would register the module toward the bottom, but before the *icepick* main loop starts:

```
...
console.register(myfilters)
console.start()
```

8.3 Testing it out

Opening `icepick`, we can see our new command:

```
>> help

...

    Available Filter commands

    dewow
    lowpass_td
    ringing
    lowpass
    agc
    migfk
    reverse
    power
    highpass
    toy_filter          <-- we did this!
    highpass_td
    gc

...

>> help toy_filter

Toy filter illustrating how to build commands

    toy_filter arg1 arg2

>> f toy_filter hello world!
argument: hello
argument: world!
```

. . . and the radargram data gets flipped! (Totally useful.)

CHAPTER

NINE

DOCUMENTATION

In addition to the basic information here, documentation can be found in *doc*. In order to build the documentation, [Sphinx](http://sphinx-doc.org/) must be installed, with the `numpydoc` extension. The extensions may be installed by

> conda install sphinx

or

> conda install numpydoc

Then, from the `doc/` directory, type

> make html

If LaTeX is available, the documentation can be compiled into a PDF. Type

> make latexpdf

A pdf copy of the current documentation can be found on the project github site

CHAPTER

TEN

IRLIB API

The following sections describe the classes of the `irlib` API. These can then be used directly from a Python script or terminal to view and manipulate radar data.

10.1 Surveys (Collections of lines in HDF Files)

Contains the *Survey* class, which is the overarching *irlib* structure. *Survey* classes handle interaction with the raw HDF datasets, and spawn off *Gather* classes for analysis. Each HDF dataset can be opened as a *Survey*, which stores file references and collects metadata in the form of a *FileHandler*. Radar lines can be created from a *Survey* using the *ExtractLine* method, which returns a *Gather*.

class `irlib.survey.`**`Survey`**(*datafile*)

Surveys can be broken down into **Gathers** and *traces*. To create a survey and extract a gather, do something like:

```
# Create the survey
S = Survey("mysurvey.h5")

# Create the gather (Line)
linenumber = 0      # This can be any nonzero integer
datacapture = 0     # This corresponds to the channel frequency in
                    # dualdar setups
L = S.ExtractLine(linenumber, dc=datacapture)

# To see what can be done with `L`, refer to the `CommonOffsetGather
# documentation
```

`ExtractLine`(*line*, *bounds=(None, None)*, *datacapture=0*, *fromcache=False*, *cache_dir='cache'*, *print_fnm=False*, *verbose=False*, *gather_type=<class 'irlib.gather.CommonOffsetGather'>*)

Extract every trace on a line. If bounds are supplied (min, max), limit extraction to only the range specified. Return a CommonOffsetGather instance.

Parameters

line
[line number to extract [integer]]

bounds
[return a specific data slice [integer x2]]

datacapture
[datacapture subset to load [integer]]

fromcache
[attempt to load from a cached file [boolean]]

cache_dir
[specify a cache directory [str]]

print_fnm
[print the cache search path [boolean]]

`ExtractTrace` (*line*, *location*, *datacapture=0*, *echogram=0*)
Extract the values for a trace and return as a vector.

Parameters

line
[line number [integer]]

location
[trace number [integer]]

datacapture
[(default 0) channel number [integer]]

echogram
[(default 0) echogram number [integer]]

`GetChannelsInLine` (*lineno*)
Return the number of channels (datacaptures per location) in a line. If the number is not constant throughout the line, then return the maximum number.

Parameters

lineno
[line number [integer]]

`GetLineCacheName` (*line*, *dc=0*, *cache_dir='cache'*)
Return a standard cache name.

Parameters

line
[line number [integer]]

dc
[datacapture number [integer]]

cache_dir
[(default *cache/*) cache directory [string]]

`GetLines` ()
Return a list of the lines contained within the survey.

`WriteHDF5` (*fnm*, *overwrite=False*)
Given a filename, write the contents of the original file to a new HDF5 wherever self.retain is True. The usage case for this is when bad data have been identified in the original file.

Note that for now, this does not preserve HDF5 object comments.

Parameters

fnm
[file path [string]]

overwrite
[(dafault *False*) overwrite existing file [boolean]]

class `irlib.survey.`**`EmptyLineError`**(*message='No message'*)

10.2 Gathers (Common-offset, Common-midpoint, etc.)

Defines various kinds of *Gather* classes, which are perhaps the most important classes in the *irlib* library. These contain the data from individual radar lines, and make it easy to apply various processing steps to the data. The *Gather* class is a base class for the *CommonOffsetGather* and *CommonMidpointGather* daughter classes. The *LineGather* object is deprecated and is now just an alias for the *CommonOffsetGather*, kept for backwards compatibility.

class `irlib.gather.`**`Gather`**(*arr*, *infile=None*, *line=None*, *metadata=None*, *retain=None*, *dc=0*)

Gathers (radar lines) are collections of radar traces (soundings). This is the base class for CommonOffsetGather and CommonMidpointGather classes, which should be chosen instead when directly creating an object.

A new *Gather* (or one of it's subclasses) is typically created by calling the *ExtractLine* method of a *Survey* instance. Alternatively, a *Gather* (or it's subclasses) can be created by passing a Numpy array as a first argument, e.g.

```
G = CommonOffsetGather(mydata)
```

where *mydata* is a data array. Some gather functionality will require that metadata be provided in the form of a *RecordList* instance.

`ConstructEigenimage`(*i*)

Return the i-th eigenimage of data.

`Dewow`(*cutoff=4000000.0*)

Apply a dewow (highpass) filter to remove very low frequency signals. This is a step up from using a simple demean operation.

Parameters

cutoff
[NOT USED]

`DoAutoGainControl`(*timewin=2e-07*)

Apply the RMS-based AGC algorithm from Seismic Unix.

Try to use a fast Cython-accellerated version. If that fails, fall back to the Numpy version.

Cohen, J.K. and Stockwell, J.W. CWP/SU: Seismic Unix Release 42. Colorado School of Mines, Center for Wave Phenomena. (1996)

Parameters

timewin
[time half-window in seconds]

dt
[sample interval in seconds]

DoMoveAvg (*width*, *kind='blackman'*, *mode='lowpass'*)
Time domain convolution filter implementation over A scan. Width is the filter window in discrete samples, and must be odd.

DoMoveAvgB (*width*, *kind='blackman'*, *mode='lowpass'*)
Time domain convolution filter implementation over B scan. Width is the filter window in discrete samples, and must be odd.

DoMurrayGainControl (*npow=2.0*, *tswitch=100.0*)
Apply a time-power gain enhancement up to a time limit, after which gain is constant (e.g. Murray et al, 1997).

Parameters

tswitch
[sample number at which to switch to constant gain]

DoRecursiveFilter (*wp*, *ws*, *gp=0.001*, *gs=20.0*, *ftype='cheby1'*)
Perform filtering with an IIR-type recursive filter. Shallow wrapper around scipy.iirdesign. Performs zero-phase filtering, otherwise picking times might not be accurate.

DoTimeGainControl (*ncoef=1.0*, *npow=1.0*, *nexp=0.0*, *gamma=1.0*, *bias=0.0*)
Apply a gain enhancement as a function of time.

Transform F:

$$f(t) \rightarrow F(f(t)) \; F = (f(t) * t^{\text{npow}} * \exp(\text{nexp} * t))^{\text{gamma}} + \text{bias}$$

Note: *ncoef* functionality has been removed and average power is preserved instead.

Claerbout (1985) suggests *npow* = 2 and *gamma* = 0.5.

DoWienerFilter (*window=5*, *noise=None*)
Noise removal using a Wiener statistical filter. *window* must be an odd positive integer.

DoWindowedSinc (*cutoff*, *bandwidth=2000000.0*, *mode='lowpass'*)
Implement a windowed sinc frequency-domain filter. This has better performance characteristics than a Chebyschev filter, at the expense of execution speed.

Parameters

cutoff
[cutoff frequency in Hz]

bandwidth
[transition bandwidth in Hz]

mode
['lowpass' or 'highpass']

Dump (*fnm=None*)
Dumps self into a cache with the given filename using pickle. Returns boolean on exit indicating success or failure.

FindFID (*fid*)
Find the index if a FID within the data array.

Parameters

fid
[FID(s) to find [either string or list(strings)]]

GetCacheName(*cache_dir='cache'*)
Returns a logical name for a pickled cache of self.

GetDigitizerFilename()
Returns the likely name of a file containing digitized line data.

GetFID(*loc*)
Return a FID. This solves the problem of knowing the FID of an array that has been sliced.

Parameters

location
[location index [integer]]

InterpolateGPSNaNs()
Interpolate positions over holes in the coordinates.

LoadLineFeatures(*infile*)
Load digitized line features, such as those generated by irview's dexport command. Returns a dictionary with the feature data.

LoadTopography(*topofnm=None*, *smooth=True*)
Load topography along the Gather's transect. If a *topofnm* is provided, then the *metadata* attribute must be valid and must contain *northings* and *eastings*.

Parameters

topofnm
[Either and ESRI ASCII grid file or None, in which case] elevations are loaded from the onboard GPS.

smooth
[(default *True*) apply a boxcare filter to soften the effects] of the DEM's discretization [boolean]

MultiplyAmplitude(*multiplier*)
Just makes viewing easier.

PprintHistory()
Return a printable string summarizing the filter history.

RemoveBlankTraces(*nsmp=100*, *threshold=4e-05*)
Attempt to identify and remove traces that did not trigger properly based on the cumulative energy in the first nsmp (int) samples.

RemoveGPSNaNs()
Remove traces with missing coordinate information.

RemoveHorizontal()
Remove horizontal features (clutter) by de-meaning each row in the data array. For example, this will attenuate the air wave.

RemoveMetadata(*kill_list*, *update_registers=True*)
Remove the metadata corresponding to traces indicated by indices in kill_list (iterable). This is more granular than the RemoveTraces method.

RemoveRinging(*n=2*)
Attempt to filter out ringing clutter by substracting the first *n* eigenimages.

RemoveTraces (*kill_list*)
Remove the traces indicated by indices in kill_list (iterable). Remove metadata as well.

Reset ()
Reset data and metadata using the internal ``*``_copy attribute variables. Does not undo the effects of operations that overwrite these attribute (such as most preprocessing routines).

RetainEigenimageRange (*rng*)
Replace data with a slice of its eigenimages. Takes a slice object *rng* as an argument.

Reverse ()
Flip gather data.

SmoothenGPS (*win=5*)
Run a moving average over the GPS northings and eastings.

SmoothenTopography ()
Load topography along line gather, reading from an ASC file. Obviously, this requires the Gather to have a valid metadata attribute.

If *smooth* =True, then apply a boxcar filter to soften the effects of the DEM discretization.

WaveletTransform (*trno*, *m=0*)
Perform a wavelet transform of a single trace.

Parameters

trno
[trace number to transform (< self.data.shape[1])]

mother
[mother wavelet (0==Morlet (default), 1==Paul, 2==DOG)]

Returns

wva
[complex 2-D array, where amplitude is np.abs(wva)]

scales
[scales used]

period
[fourier periods of the scales used]

coi
[e-folding factor used for cone-of-influence]

class irlib.gather.**CommonOffsetGather** (*arr*, *infile=None*, *line=None*, *metadata=None*, *retain=None*, *dc=0*)
This is a subclass of *Gather* that defines a common-offset radar line.

FindLineBreaks (*threshold=0.35*)
Find vertices along a scattered line.

FixStaticGPS ()
Attempt to approximate locations with data from when the GPS was operating on static mode.

GetTopoCorrectedData ()
Stop-gap method for getting topographically corrected radargrams. Eventually a more graceful way should be built into the API, but for now, this returns a new array without messing with self.data (the repercussions of which I haven't fully considered yet).

`Interpolate` (*X_int*, *X*, *arr=None*)

Interpolate data over space.

Parameters

X_int
[location to interpolate at]

X
[data locations]

Returns

D_int
[linearly interpolated data]

`LineProjectMultiSegment` (*dx=4.0*, *threshold=0.35*, *verbose=False*)

Projects data to a sequence of approximating line segments with even spacing.

THIS MAY BREAK THINGS because picks and metadata aside from coordinates aren't handled. Use with caution.

Parameters

dx
[point spacing in meters]

threshold
[threshold for segment bending]

verbose
[print out status message]

Returns

segments

Xdbg

Ydbg

Pdbg

Pgriddbg

`LineProjectXY` (*bounds=None*, *eastings=None*, *northings=None*, *sane=True*)

Project coordinates onto a best-fit line.

Parameters

bounds
[(optional) side limits on projection [tuple x2]]

eastings
[(optional) coordinate eastings; overrides coordinates in] self.metadata [numpy.ndarray]

northings: (optional) coordinate northings; overrides coordinates in
self.metadata [numpy.ndarray]

sane
[(default *True*) perform sanity checks on coordinate [boolean]]

Returns

X
[projected eastings]

Y
[projected northings]

p
[a list of polynomial fitting information]

`LineProject_Nearest` (*bounds=None*, *eastings=None*, *northings=None*, *dp=4.0*)
Populate a best-fit line with traces nearest to equally-spaced points.

Parameters

bounds
[(optional) side limits on projection [tuple x2]]

eastings
[(optional) coordinate eastings; overrides coordinates in] self.metadata [numpy.ndarray]

northings: (optional) coordinate northings; overrides coordinates in
self.metadata [numpy.ndarray]

dp
[(default *4.0*) point spacing in meters [float]]

Returns

Xmesh
[projected eastings]

Ymesh
[projected northings]

proj_arr
[projected data]

sum_dist
[best-fit line length in meters]

`MigrateFK` (*dx=4.0*, *t0_adjust=0*, *verbose=True*)
Perform Stolt migration over multiple sections.

Parameters

dx
[gridding interval]

t0_adjust
[zero-time offset from top of self.data, in samples] e.g. if the radargram contains data from before t0, this corrects for that.

Returns

migsections
[list of quasilinear migration sections]

RemoveBadLocations(*bbox=None*)
Remove traces where the location is None or outside of an optional bounding box.

Parameters

bbox
[(optional) [east, west, south, north] bounding box [tuple]]

RemoveStationary(*threshold=3.0*, *debug=False*)
Remove consecutive points with very similar GPS locations. Do this by finding points within a minimum distance of each other, and averaging them. Beware incorrect GPS readings, which need to be fixed first.

Parameters

threshold
[(default *3.0*) minimum offset in meters to recognize that] a position has moved [float]

debug
[(default *False*) print debugging messages [boolean]]

class irlib.gather.**CommonMidpointGather**(*arr*, *infile=None*, *line=None*, *metadata=None*, *retain=None*, *dc=0*)
Subclass defining common-midpoint specific data and operations.

ReadIndex(*index*)
Read CMP index file and return the offsets as a 1-d array and location key as a 2-d array.

class irlib.gather.**PickableGather**(*data*, *infile=None*, *line=None*, *metadata=None*, *retain=None*, *dc=0*)
Subclass of gather adding attributes and methods relevant to event picking.

CalcAveragePicks(*key*, *picks*)
Average picks by location, based on a key. The key is a list of paired iterables, containing first and last locations for every shot gather. This is useful for CMP-style surveys, where multple shots need to be averaged.

LoadPicks(*infile*)
Load bed picks from a text file. Employs a FileHandler.

PickBed(*sbracket=(60, 200)*, *bounds=(None, None)*, *phase=1*)
Attempt to pick bed reflections along the line. Return pick data and estimated polarity in vectors (also stored internally).

Parameters

sbracket
[tuple defining minimum and maximum times (by] sample number) during which the event can be picked

bounds
[location number limits for picking]

PickDC (*sbracket=(20, 50)*, *bounds=(None, None)*)
Attempt to pick direct coupling waves along the line. Return pick data in a vector (also stored internally).

Parameters

sbracket
[tuple defining minimum and maximum times (by] sample number) during which the event can be picked

bounds
[location number limits for picking]

RemoveMetadata (*kill_list*, *update_registers=True*)
Remove the metadata corresponding to traces indicated by indices in kill_list (iterable). This is more granular than the RemoveTraces method.

RemoveTraces (*kill_list*)
Remove the traces indicated by indices in kill_list (iterable). Remove metadata as well.

Reset ()
Reset data and metadata using the internal *_copy attribute variables. Does not undo the effects of operations that overwrite these attribute (such as most preprocessing routines).

Reverse ()
Flip gather data.

SavePicks (*outfile*, *picks*, *mode='bed'*)
Save picks to a text file.

class irlib.gather.**LineGatherError** (*message='No message'*)

10.3 Metadata

Contains the *RecordList* class, which in addition to being useful for functions that try to directly read XML metadata from HDF datasets, is also used as the metadata container for *Gather* objects.

class irlib.recordlist.**RecordList** (*filename=None*)
Class to simplify the extraction of metadata from HDF5 radar datasets.

Usage:

- initialize a RecordList instance with a filename (arbitrary, but should be the HDF filename)
- add datasets by passing h5 dataset objects to *self.AddDataset()*

AddDataset (*dataset*, *fid=None*)
Add metadata from a new dataset to the RecordList instance. Updates the RecordList internal lists with data parsed from the radar xml.

Does not read pick data.

Parameters

dataset
[an h5py dataset at the *echogram* level] (fh5[line][location][datacapture][echogram])

fid
[pre-defined FID for the dataset]

Returns None

CropRecords ()
Ensure that all records are the same length. This should be called if adding a dataset fails, potentially leaving dangling records.

Cut (*start*, *end*)
Drop section out of all attribute lists in place.

Reverse ()
Reverse data in place.

Write (*f*, *eastern_hemisphere=False*)
Write out the data stored internally in CSV format to a file object f.

IF lat and lon are both al

class irlib.recordlist.**ParseError** (*message=''*, *fnm=''*)

10.4 Picking and rating file management

Contains the *FileHandler* class, which abstracts the process of reading and writing picking files.

class irlib.filehandler.**FileHandler** (*fnm*, *line*, *fids=None*)
Class for reading and writing pick and rating files cleanly. If no list of FIDs is provided, then they will be guessed assuming locations are in order and complete. If a list of FIDs is provided, it will be used instead.

AddBedPicks (*fids*, *vals*)
Add reflection picks at locations given by FIDs

AddDCPicks (*fids*, *vals*)
Add direct wave picks at locations given by FIDs

ComputeTravelTimes ()
Where possible, subtract dc times from bed times.

GetEventVals ()
Return the airwave and bed reflection values (lists).

GetEventValsByFID (*fids*)
Return the airwave and bed reflection picks for a list of FIDs.

GetEventVals_Interpolated (*max_fid=None*)
Similar to GetEventVals, however returns a value for every FID. If max_fid is given, it is taken to be the ending FID, otherwise the last value in self.fids will be used as the ending FID.

self.fids must not be None, otherwise this won't work.

This method is only valid if missing records can be assumed to be linearly related to existing records. A less naive approach is to use locations from a line's metadata to do this interpolation.

Parse (*recs*)
Read pick file records, and split into fields.

Write ()
Write to file.

sort()
Sort bedvals and dcvals by FID in-place.

class irlib.filehandler.**FileHandlerError**(*message='No message'*)

10.5 itools

10.6 Application building

The following modules contain the building blocks the graphical applications (such as IcePick2), as well as user-defined filters. They would be used to develop or extent the existing graphical tools.

10.6.1 Console

The Console class is the controller for irlib-based apps. Windows can be attached and detached from the Console, which handles user input and passes directives to its windows.

class irlib.app.console.**Console**(*progname*, *bannertext=''*)
App-controller with a user input readline loop.

add_appwindow(*ref*)
Add a window to be managed by the Console.

get_appwindows(*t=None*)
Get all windows of a particular type from the window list.

get_command()
Get a command from console input.

handle_command(*cmd*)
Parse and handle user input. This will be deprecated int he future for a more modular approach that allows commands to be added and removed dynamically.

open_line(*lineno*, *dcno=0*, *fromcache=True*)
Open a line from a survey

print_syntax()
Print start-up syntax for the forgetful.

register(*module*)
Load commands from a module a register them for use.

remove_appwindow(*ref*)
Remove a window from Console management.

start()
Begin input-output loop

10.6.2 Windows

Define application components

class `irlib.app.components.`**`AppWindow`**(*winsize*)

This is the generic application window class, and contains an axes instance and event-handling and update machinery.

Subclasses must overload _onclick() and _onkeypress() methods.

`update`()

Redraw the axes

class `irlib.app.components.`**`Radargram`**(*L*)

Shows a radargram, and has methods for displaying annotations and collecting digitized features.

`get_digitizer_filename`()

Return an automatically-generated filename for digitized features.

`load`(*f*)

Parse a digitizer file and return a dictionary with list entries that can be dropped directly into an ImageWindow.

`remove_annotation`(*name*)

Properly remove an annotation from the Radargram axes.

`repaint`(*lum_scale=None*, ***kwargs*)

Redraw the radargram raster

`save`()

Returns a list of dictionaries containing the latitude, longitude, and the y-axis value of each vertex. Dictionary keys are standard linloc FIDs.

`update`()

Display a radargram on axes. Paints in background, and all subsequent calls update lines. Passing repaint as True forces the background to be redrawn (for example, after a filter opperation).

class `irlib.app.components.`**`PickWindow`**(*L*, *ntraces=8*)

Show a series of A-scan traces and allow picking

`autopick_bed`(*t0=150*, *tf=10000*, *lbnd=None*, *rbnd=None*)

Attempt to pick the first break of the direct-coupling wave. Optional constraints on start and end time can be passed to improve results.

Parameters

t0

[start time, in samples]

tf

[end time, in samples]

`autopick_dc`(*t0=10*, *tf=150*)

Attempt to pick the first break of the direct-coupling wave. Optional constraints on start and end time can be passed to improve results.

Parameters

t0

[start time, in samples]

tf
[end time, in samples]

change_mode (*mode*)
Change picking mode between bed and direct coupling.

connect_radargram (*rg*)
Connect a Radargram instance so that the PickWindow can modify it.

load_picks (*fnm=None*)
Load picks. If no *fnm* is provided, attempt to load from an autogenerated location.

save_picks (*fnm=None*)
Save picks. If no fnm is provided, generate one based on self.L.

update ()
Redraw axes and data

update_radargram ()
Send picking annotations to a connected Radargram.

class irlib.app.components.**MapWindow** (*L*)
Displays a simple map of trace locations

update ()
Redraw the map.

10.6.3 Command parsing

exception irlib.app.command_parser.**CommandApplicationError** (*exception*)

irlib.app.command_parser.**apply_command** (*registry*, *inputs*, *stateobj*, *cmdtype*)
Attempt to apply a command, raising a KeyError if a suitable command cannot be found.

Parameters

registry: dictionary of commands and their associated Command classes

inputs: list containing first the command string, and then any arguments

stateobj: the stateful object that may be modified by the command
e.g. a Console or Gather instance

irlib.app.command_parser.**help_command** (*registry*, *cmd*)
Print the help documentation for *cmd*, or raise Key if it cannot be found.

10.6.4 Filter framework

This file defines the filter commands available from irlib-based apps.

class irlib.app.filters.**AutoGainControl**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**Dewow**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**FilterCommandBase**
A FilterCommand is implemented as a class with a command-line signature and an *apply* method that takes an appropriate Gather object as an argument.

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**Highpass_FD**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**Highpass_TD**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**LinearGainControl**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**Lowpass_FD**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**Lowpass_TD**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**MigrateFK**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class irlib.app.filters.**ReflectionPower**

apply (*G*, *args*)
Overload this method to perform operations on Gather object *G*.

class `irlib.app.filters.`**`RemoveRinging`**

  `apply`(*G*, *args*)
    Overload this method to perform operations on Gather object *G*.

class `irlib.app.filters.`**`Reverse`**

  `apply`(*G*, *args*)
    Overload this method to perform operations on Gather object *G*.

CHAPTER ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

G

H

I

L

M

O

P

R

S

U

W