gnssrefl

Kristine M. Larson and GNSS-IR community

Jun 08, 2026

CONTENTS:

1 Installation 3

2 Understanding 5

3 Files, Formats, Frequencies 25

4 Quick Links/Expert Modes 41

5 Example Use Cases 43

6 Community Information 45

7 What is a good GNSS Reflections Site? 47

8 API documentation 51

Python Module Index 243

Index 245

Date: Jun 08, 2026

`gnssrefl` is an open source python-based software package for GNSS interferometric reflectometry (GNSS-IR). Please note that the link to API documentation shown below will provide access to every single piece of code in gnssrefl. For the the main modules in gnssrefl, it is probably easier and faster to use Quick Links to the Code. For individuals using a locally installed python version, python version 3.10 is required.

CHAPTER

ONE

INSTALLATION

You can access this package via Jupyter notebooks, Docker containers, or traditional github/pypi package installation. **If you are using Windows, you must use dockers.** I believe you can also use a linux emulator and follow instructions for linux.

1.1 Jupyter Notebooks

Install Instructions

Notebook Area for Use Case Examples

1.2 Docker Container

Install Instructions

1.3 Local Python Install for Linux/MacOS

As of gnssrefl version 3.19.4, python version >= 3.10 is required. If you have problems with the install, please let us know by posting a github Issue.

1.3.1 Environment Variables

You should define three environment variables:

- EXE = where various executables will live. These are mostly related to manipulating RINEX files.
- REFL_CODE = where the reflection code inputs (SNR files and instructions) and outputs (RH) will be stored (see below). Both snr files and results will be saved here in year subdirectories.
- ORBITS = where the GPS/GNSS orbits will be stored. They will be listed under directories by year and sp3 or nav depending on the orbit format. If you prefer, ORBITS and REFL_CODE can be pointing to the same directory.

If you are running in a bash environment, you should save these environment variables in the .bashrc file that is run whenever you log on.

If you don't define these environment variables, the code *should* assume your local working directory (where you installed the code) is where you want everything to be (to be honest, I have not tested this in a while). The orbits, SNR

files, and periodogram results are stored in directories in year, followed by type, i.e. snr, results, sp3, nav, and then by station name.

1.3.2 Direct Python Install

If you are using the version from gitHub:

- You may want to install the python3-venv package apt-get install python3-venv
- apt-get install git
- git clone https://github.com/kristinemlarson/gnssrefl
- cd into that directory, set up a virtual environment, a la python3 -m venv env **make sure you are running the correction of python, as discussed at the top of the page** You can have two versions of python on your machine. To have it run 3.10 instead of 3.11 (for example), type python3.10 -m venv env
- activate your virtual environment source env/bin/activate
- pip install . **Make sure you are using pip3 - either directoyr or linked to pip**
- so please read below or type installexe -h

1.3.3 PyPi Install

- make a directory, cd into that directory, set up a virtual environment, a la python3 -m venv env **Make sure you are running the correct version of python as discussed at the top of the page**
- activate the virtual environment, source env/bin/activate
- pip install gnssrefl **Make sure you are using pip3 - either directly or linked to pip**
- Please read below or type installexe -h

1.3.4 Non-Python Code

installexe should download and install two key utilities used in the GNSS community: CRX2RNX and gfzrnx. It currently works for linux, macos and mac-newchip options. If you are using docker or Jupyter notebooks **you do not need to run this.**

We no longer encourage people to use **teqc** as it is not supported by EarthScope/UNAVCO. We try to install it in case you would like to use it on old files.

1.3.5 Homework 0: Test installation.

For some of the shortcourses, we compiled a **Homework 0** that walks a new user through a few simple tests for validating successful gnssrefl installation.

CHAPTER

TWO

UNDERSTANDING

gnssrefl is an open source/python version of my GNSS interferometric reflectometry (GNSS-IR) code.

2.1 Goals

The goal of the gnssrefl python repository is to help you compute (and evaluate) GNSS-based reflectometry parameters using standard GNSS data. This method is often called GNSS-IR, or GNSS Interferometric Reflectometry. There are three main sections:

- Translation: Use either **rinex2snr** or **nmea2snr** to translate native GNSS formats to what gnssrefl needs. The output is called a *SNR* file.
- **quickLook** gives you a quick (visual) assessment of a SNR file without dealing with the details associated with **gnssir**. It is not meant to be used for routine analysis. It also helps you pick an appropriate azimuth mask and quality control settings.
- **gnssir** computes reflector heights (RH) from SNR files.

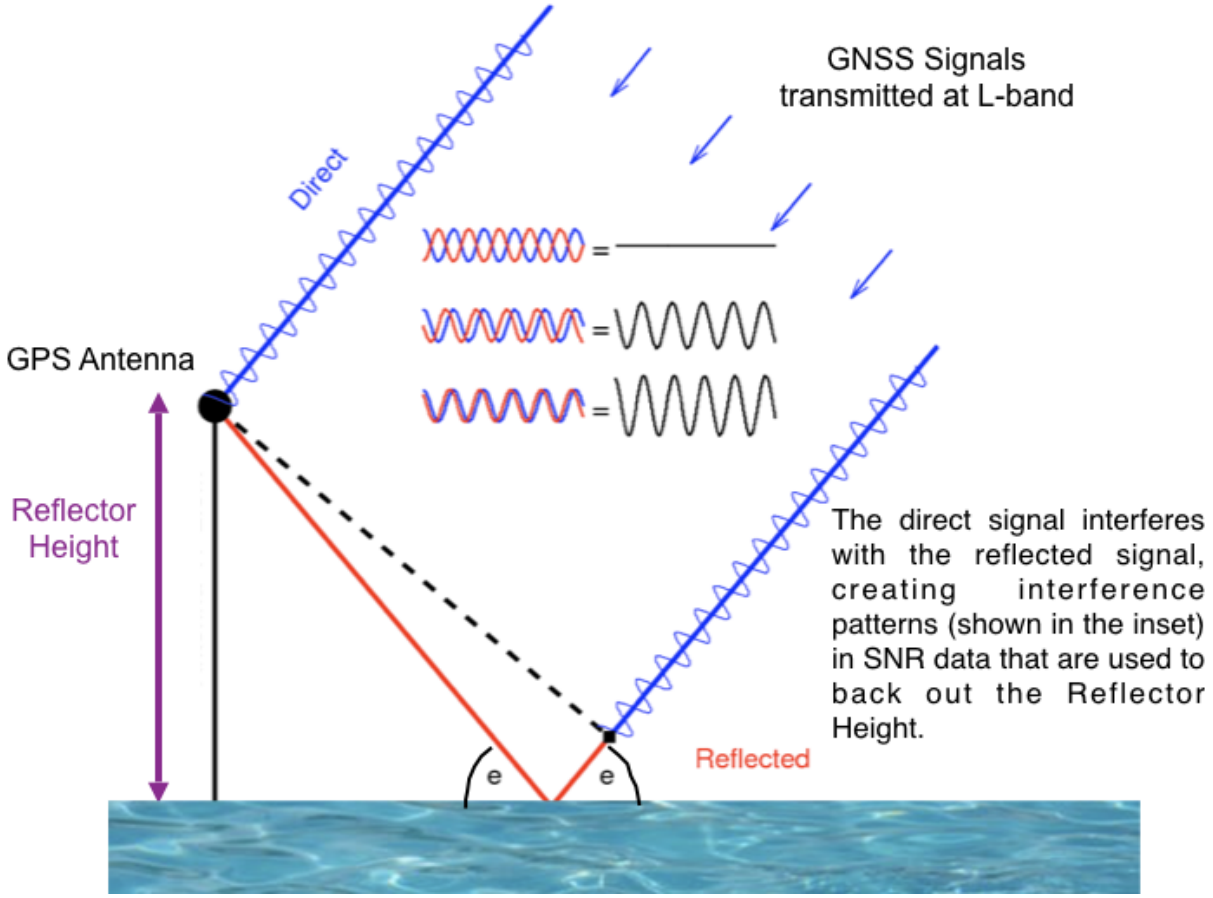
There are also various *utilities* you might find to be useful. If you are unsure about why various restrictions are being applied, it is really useful to read Roesler and Larson (2018) or similar. You can also watch some background videos on GNSS-IR at youtube.

2.2 Philosophy

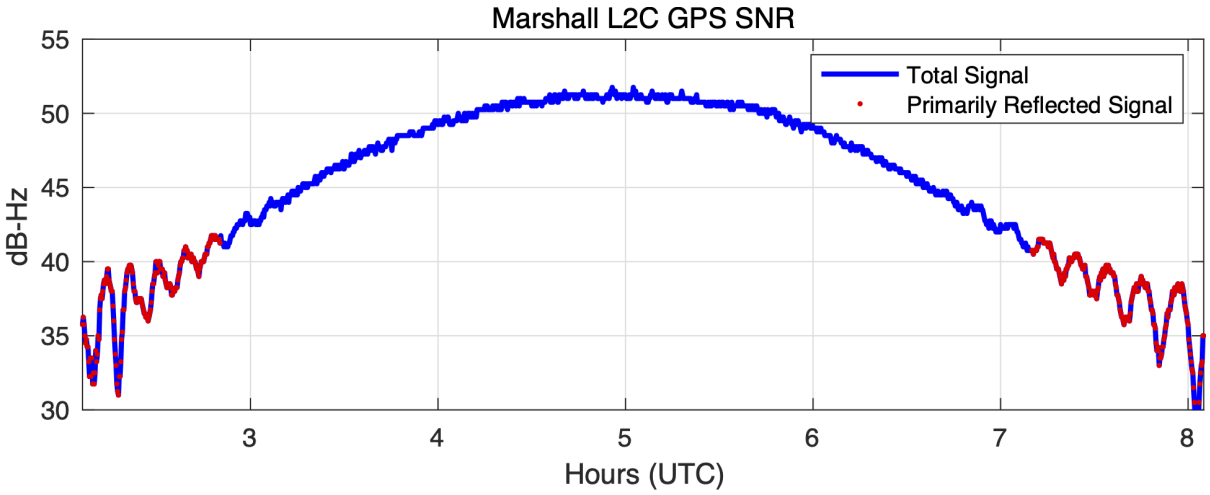
In geodesy, you don't really need to know much about what you are doing to calculate a reasonably precise position from GPS data. That's just the way it is. (Note: that is also thanks to the hard work of the geodesists that wrote the computer codes). For GPS/GNSS reflections, you need to know a little bit more - like what are you trying to do? Are you trying to measure water levels? Then you need to know where the water is! (with respect to your antenna, i.e. which azimuths are good and which are bad). Another application of this code is to measure snow accumulation. If you have a bunch of obstructions near your antenna, you are responsible for knowing not to use that region. If your antenna is 10 meters above the reflection area, and the software default only computes answers up to 6 meters, the code will not tell you anything useful. It is up to you to know what is best for the site and modify the inputs accordingly. I encourage you to get to know your site. If it belongs to you, look at photographs. If you can't find photographs, use Google Earth. You can also try using my google maps web app interface.

2.3 Reflected Signal Geometry

To summarize, direct (blue) and reflected (red) GNSS signals interfere and create an interference pattern that can be observed in GNSS Signal to Noise Ratio (SNR) data as a satellite rises or sets. The frequency of this interference pattern is directly related to the height of the GNSS antenna phase center above the reflecting surface, or reflector height RH (purple). *The primary goal of this software is to measure RH.* This parameter is directly related to changes in snow height and water levels below a GNSS antenna. This is why GNSS-IR can be used as a snow sensor and tide gauge. GNSS-IR can also be used to measure soil moisture, but the code to estimate soil moisture is not as strongly related to RH as snow and water.

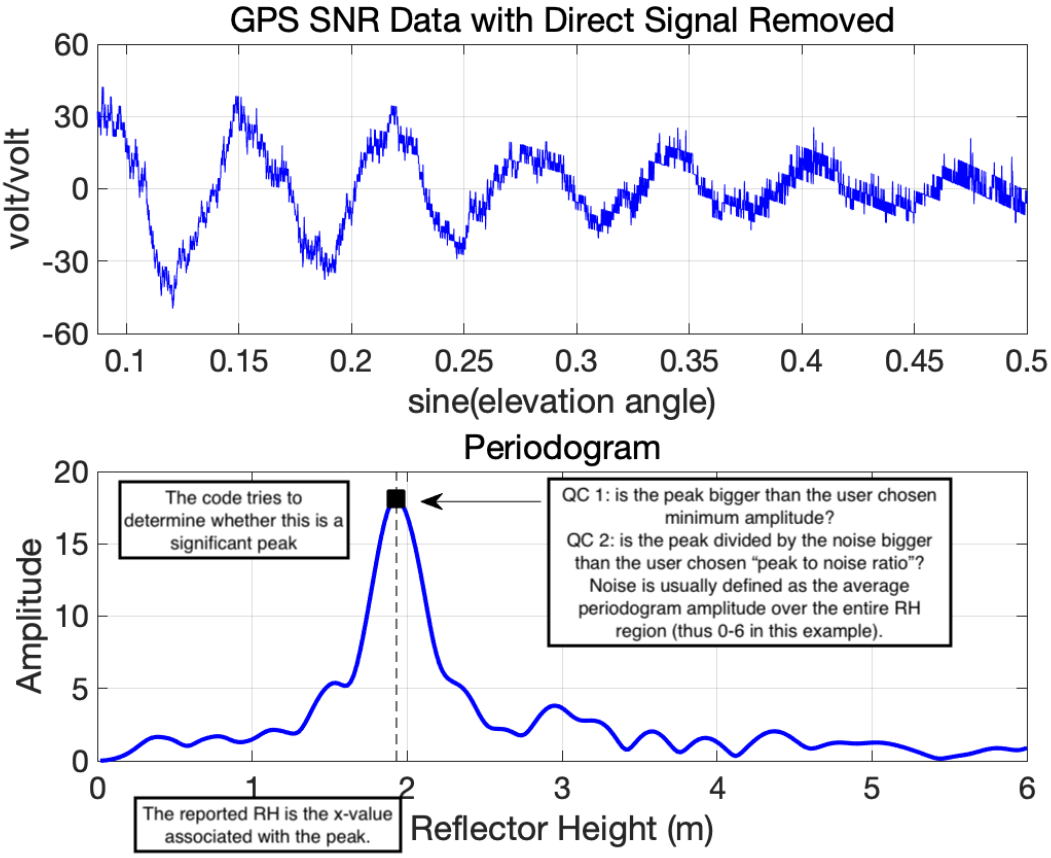


This code is meant to be used with Signal to Noise Ratio (SNR) data. This is a SNR sample for a site in the the northern hemisphere (Colorado) and a single GPS satellite. The SNR data are plotted with respect to time - however, we have also highlighted in red the data where elevation angles are less than 25 degrees. These are the data used in GNSS Interferometric Reflectometry GNSS-IR. You can also see that there is an overall smooth polynomial signature in the SNR data. This represents the dual effects of the satellite power transmission level and the antenna gain pattern. We aren't interested in that so we will be removing it with a low order polynomial (and we will convert to linear units on y-axis). After the direct signal polynomial is removed, we will concentrate on the *rising* and *setting* satellite arcs. These are shown in red.



For a more dynamic example, look at these SNR data from Kachemak Bay

Once the direct signal is removed (and units changed), you will have a dataset as shown below. The x-axis is now in sine(elevation angle) instead of time, as this is the easiest way to analyze the spectral characteristics of the data. Below the SNR data is the periodogram associated with it. This periodogram is what allows us to estimate the reflector height of the antenna.



In a nutshell, that is what this code does - it tries to find the rising and setting arcs for all GNSS satellites in a datafile, computes periodograms to find the dominant frequencies which can be related to reflector heights, and ultimately defines environmental characteristics from them.

There are three big issues :

1. You need to make sure that dominant frequency is meaningful (**Quality Control**).
2. You need to make sure that the reflected signals are actually coming from where you want them (**Reflection Zones**)
3. Your receiver must be collecting data at sufficient rate so that your GNSS-IR results are not violating the Nyquist frequency (**Maximum Resolvable Reflector Height**).

2.4 Quality Control

This code uses a Lomb Scargle periodogram (LSP). This type of periodogram allows the input data to be sampled at uneven periods. The primary inputs are :

- how precise (in reflector height units) do you want the periodogram calculated at?
- how far (in reflector height units) do you want the periodogram calculated for?

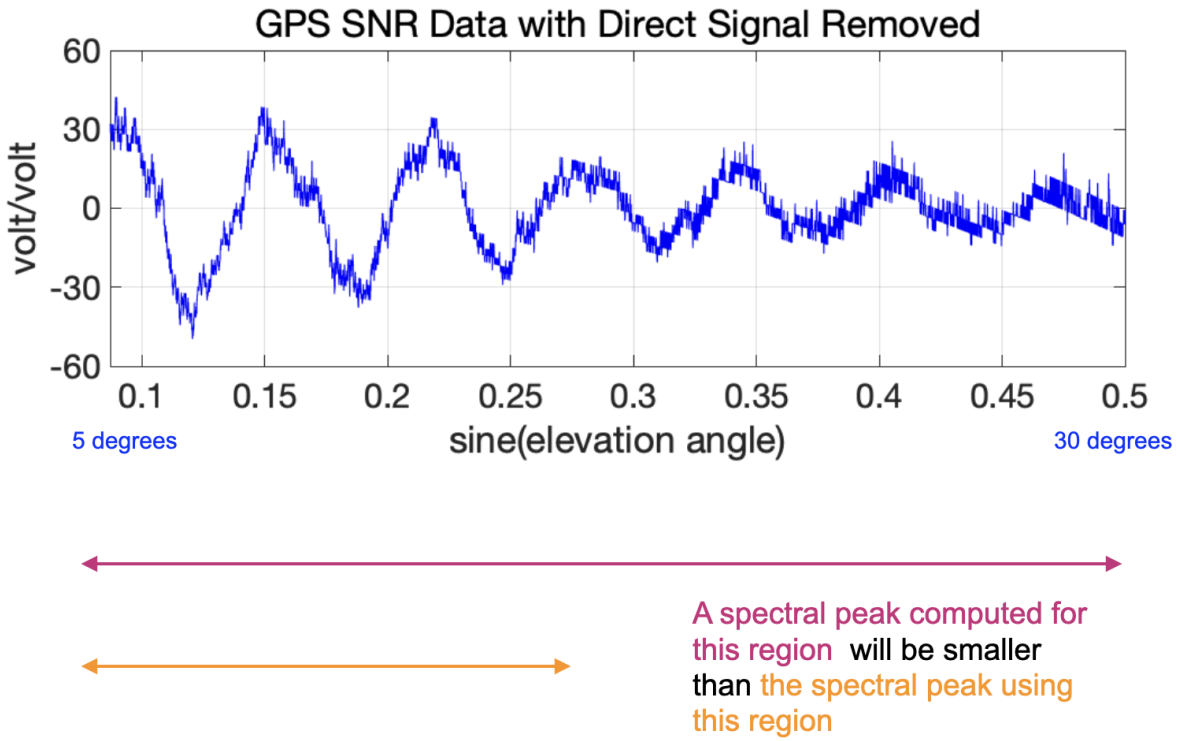
In other words, how densely sampled on the x-axis will your periodogram be and how far along the x-axis will it be? The first parameter should not set to something that makes no sense (i.e. so small the code takes forever to run). In this code the second parameter is the max reflector height (h2). The minimum reflector height is always zero, and then the values lower than the minimum reflector height (h1) are thrown out.

It is easy to compute a periodogram and pick the maximum value so as to find the reflector height. It is more difficult to determine whether it is one you should trust.

- is the peak larger than a user-defined value (amplitude of the dominant peak in your periodogram)
- is the peak divided by a "noise" metric larger than a user-defined value. This noise metric is defined over a user defined reflector height region. (peak2noise).
- is the data arc sufficiently "long" (ediff)

The amplitude and peak2noise ratio are influenced by choices you make, i.e. the elevation angle limits and the noise region used to compute peak2 noise. And they are also influenced by the kind of experiment you do and receiver you use.
Some examples follow:

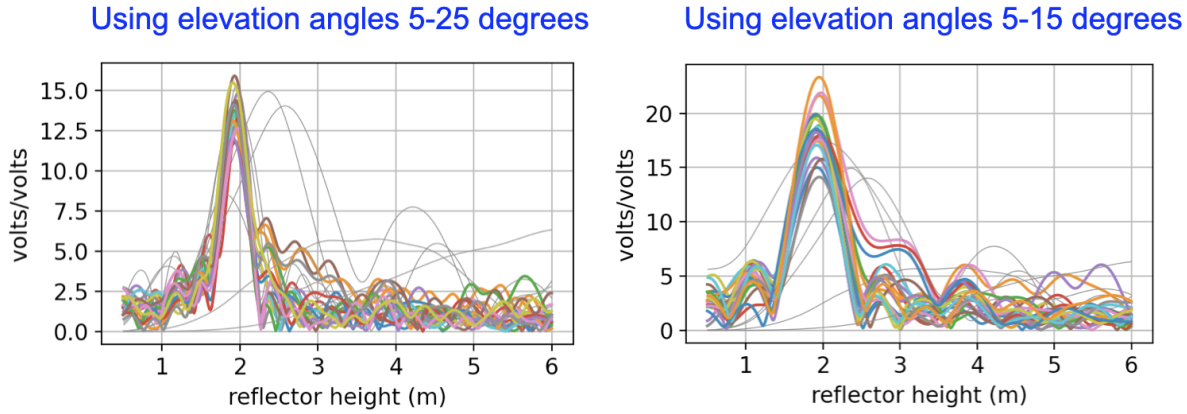
Here we show a SNR series - outlining two different elevation angle regions in colors.



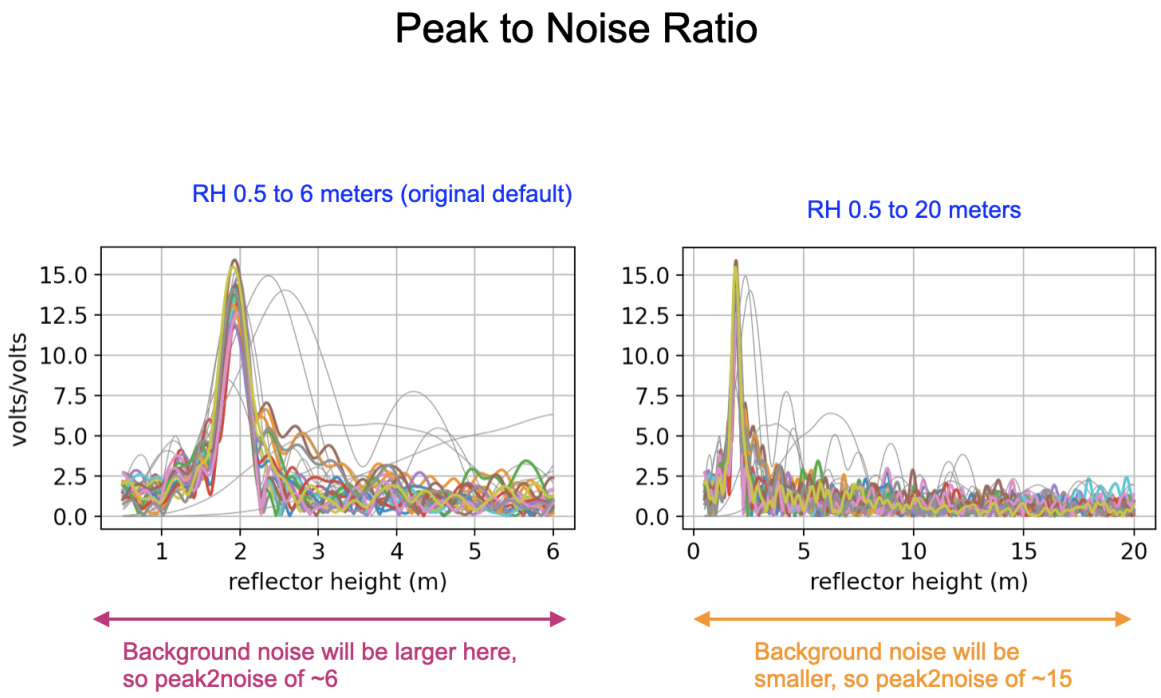
Kristine M. Larson, 2023 GNSS-IR Short Course

We should expect that the periodograms will look different for these two regions and they are. The peak amplitudes are larger when you only use the lower elevation angle data. But the periodograms are wider (why?).

Example periodograms for SNR data

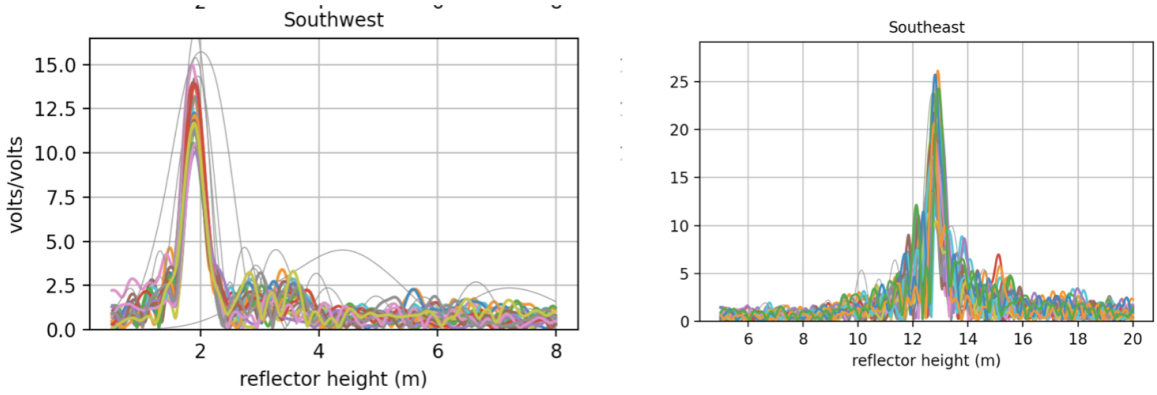


Peak2noise depends on the noise region. In quickLook it uses the same RH limits for noise as for computing the periodogram. You can easily see that if you said you wanted all H values below 20 meters, the noise region is much much larger, which means the peak value dividied by the noise values will be much much bigger.



This is an example where two different stations with different surfaces are shown. The peak amplitudes of the periodograms are different. This simply means that the ice has a different dielectric constant than soil. You can verify this using the Nievinski simulator.

Peak amplitudes depend on the surface



here the x-axis tells you the reflector height value is ~1.9 meters. The different colors are different satellites

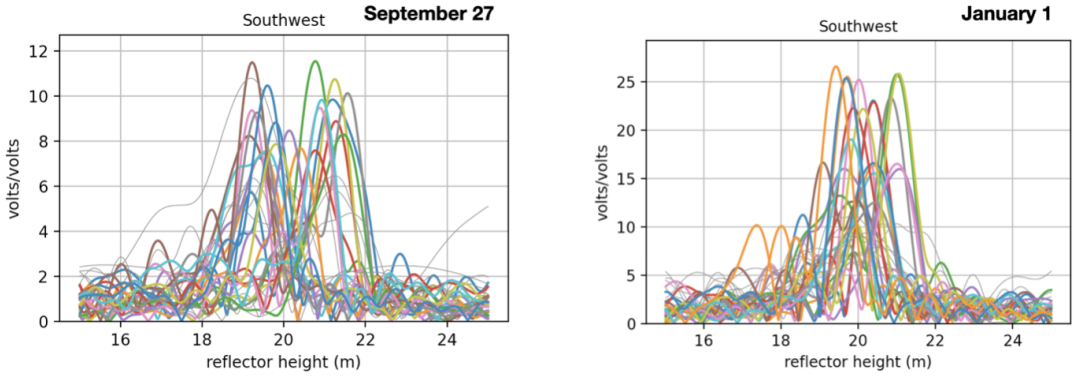
in this case the antenna is 13 meters above the Greenland ice sheet

PBO H2O site - bare soil

ice sheet

Here is an example where the same station is used in both periodograms - but the surface itself changed.

How is September in northern Greenland different than January?



These are for a site that measures water reflections in Thule Greenland. Why are the peaks at different x-axis values?

In addition to amplitude and peak2noise, the code uses a quality control parameter called **ediff.** to test whether the data arc is sufficiently "long" in an elevation angle sense. ediff has units of degrees. If you set your desired elevation angle limits to 5 and 20 degrees, and ediff was 2, which is the default, then the code will require all arcs to track from at least 7 degrees and go up to 18 degrees. If you had a very short elevation angle range, i.e. 5-10 degrees, you might want to make that a little stricter, minimum of 6 and at least go up to 9 degrees, so an ediff of 1. If you don't want to enforce this, just set it to something big. But you can't turn off all quality control. Since the amplitude can be influenced by the kind of receiver you are using, if you aren't sure what a good value would be, you can set that to zero. And you can use quickLook to get an idea of what it should be.

One more warning: if you tell the code that you want to use elevation angles of 5 to 25 degrees and it turns out that your receiver was using an elevation mask of 10 degrees, you will almost certainly end up with no useful results. Why? Because the best you will do is have a min elevation angle of 10 degrees, and the code will expect them to start at 7 degrees (i.e. 5 + 2). Some cryosphere community members use 7 degree masks on their receivers for no reason that I can understand - so that situation would also end up with a lot of arcs thrown out.

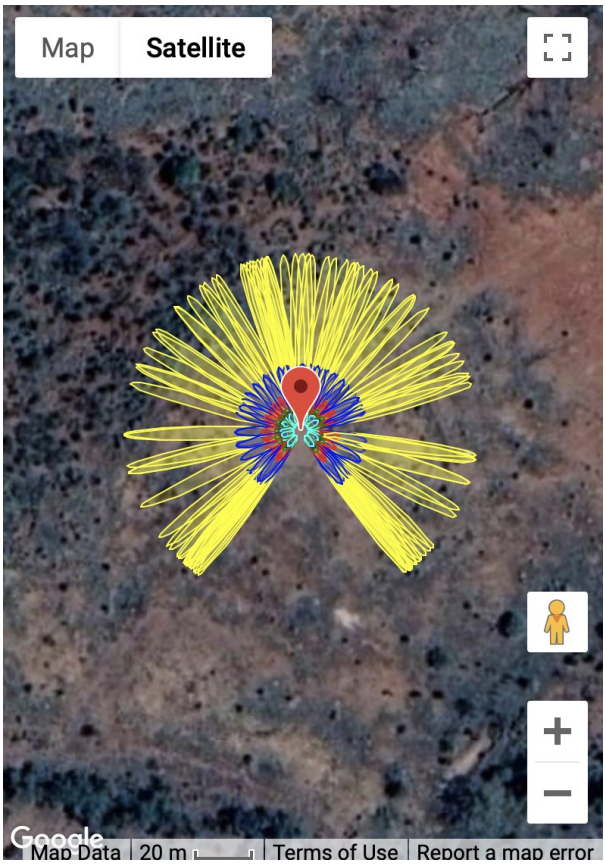
Another way of thinking about how long an arc is measured in time units. The parameter is called delTmax in the code and is defined in minutes. The default is very long - 75 minutes - as this code is meant to be useable for soil moisture, snow, and tides. This will get you into trouble if you are measuring tides and the tide rates of change are large. In those cases, you might wish to reduce delTmax. See Grauerort for an example of this problem.

Even though we analyze the data as a function of sine of elevation angle, each satellite arc is associated with a specific time period. The code keeps track of that and reports it in the final answers. Each track is associated with an azimuth. In the initial versions of the code this was the average azimuth for all the data in your track. From version 1.4.5 and on, it is the azimuth of the lowest elevation angle in your arc.

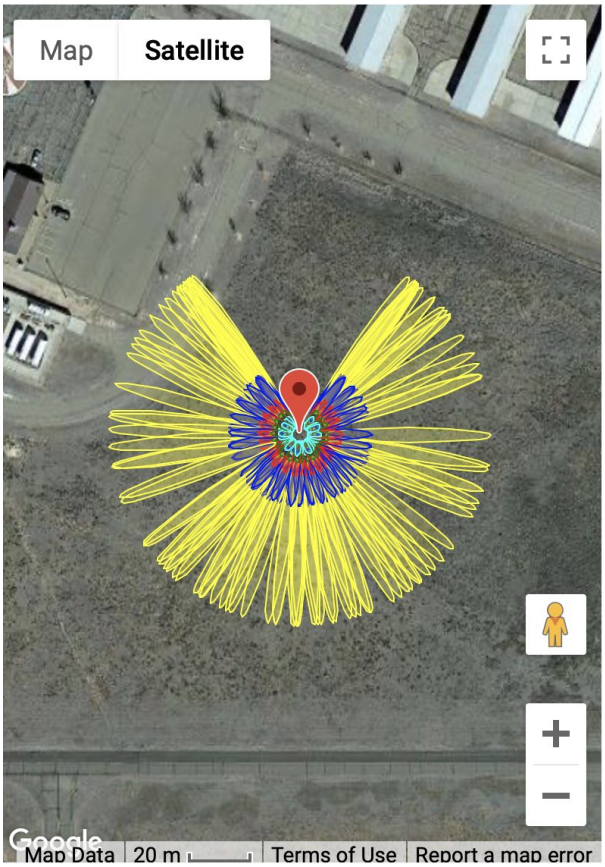
2.5 Reflection Zones

What do these satellite reflection zones look like? Below are photographs and reflection zone maps for two standard GNSS-IR sites, one in the northern hemisphere and one in the southern hemisphere.

Mitchell, Queensland, Australia

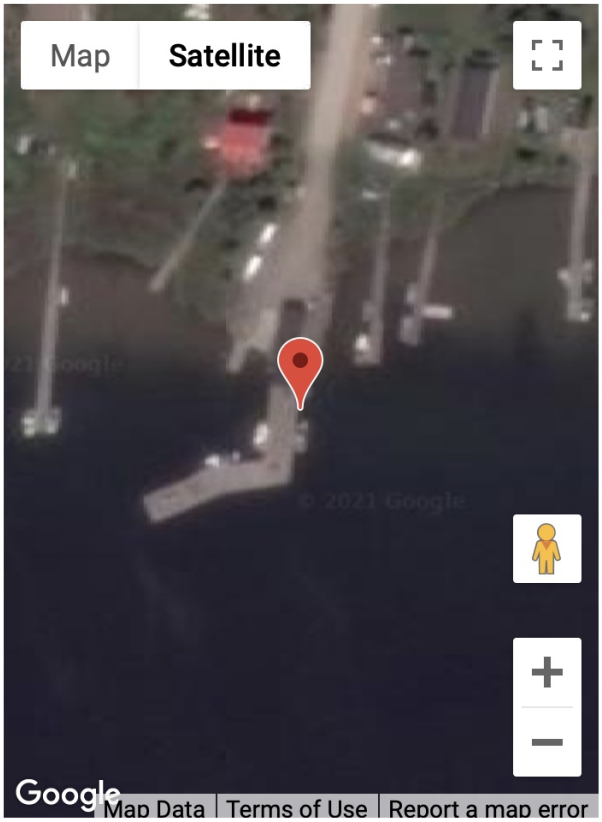


Portales, New Mexico, USA



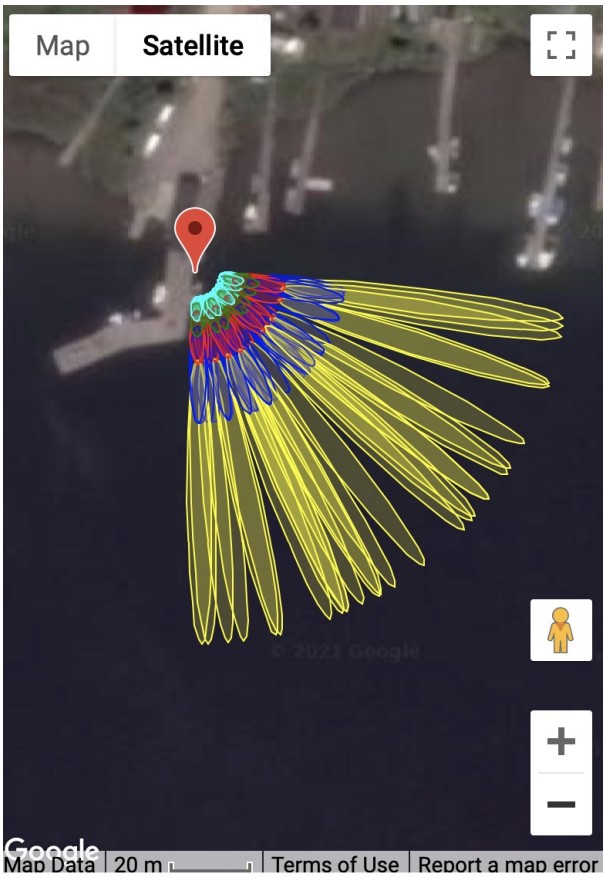
Each one of the yellow/blue/red/green/cyan clusters represents the reflection zone for a single rising or setting GPS satellite arc. The colors represent different elevation angles - so yellow is lowest (5 degrees), blue (10 degrees) and so on. The missing satellite signals in the north (for Portales New Mexico) and south (for Mitchell, Australia) are the result of the GPS satellite inclination angle and the station latitudes. The length of the ellipses depends on the height of the antenna above the surface - so a height of 2 meters gives an ellipse that is smaller than one that is 10 meters. In this case we used 2 meters for both sites - and these are pretty simple GNSS-IR sites. The surfaces below the GPS antennas are fairly smooth soil and that will generate coherent reflections. In general, you can use all azimuths at these sites.

Now let's look at a more complex case, station ross on Lake Superior. Here the goal is to measure water level. The map image (panel A) makes it clear that unlike Mitchell and Portales, we cannot use all azimuths to measure the lake. To understand our reflection zones, we need to know the approximate lake level. That is a bit tricky to know, but the photograph (panel B) suggests it is more than the 2 meters we used at Portales - but not too tall. We will try 4 meters and then check later to make sure that was a good assumption.

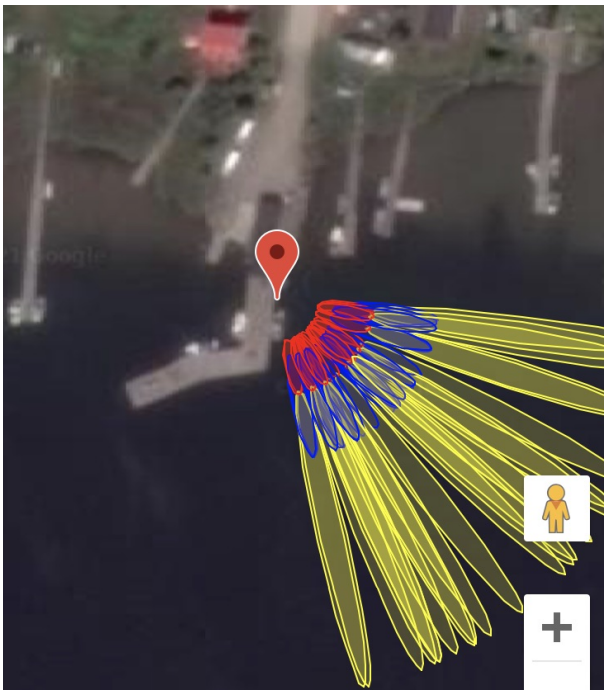


A. Google Map of station ROSS

B. Photograph of station ROSS



C. Reflection zones for GPS satellites at elevation angles of 5-25 degrees for a reflector height of 4 meters.



D. Reflection zones for GPS satellites at elevation angles of 5-15 degrees for a reflector height of 4 meters.

Again using the reflection zone web app, we can plot up the appropriate reflection zones for various options. Since

ross has been around a long time, http://gnss-reflections.org has its coordinates in a database. You can just plug in ross for the station name and leave latitude/longitude/height blank. You *do* need to plug in a RH of 4 since mean sea level would not be an appropriate reflector height value for this case.

Start out with an azimuth range of 90 to 180 degrees. Using 5-25 degree elevation angles (panel C) looks like it won't quite work - and going all the way to 180 degrees in azimuth also looks it will be problematic. Panel D shows a smaller elevation angle range (5-15) and cuts off azimuths at 160. These choices appear to be better than those from Panel C. It is also worth noting that the GPS antenna has been attached to a pier - and *boats dock at piers*. You might very well see outliers at this site when a boat is docked at the pier.

Note: we now have a refl_zones tool in the gnssrefl package.

Once you have the code set up, it is important that you check the quality of data. This will also allow you to check on your assumptions, such as the appropriate azimuth and elevation angle mask and reflector height range. This is the main reason quickLook was developed.

2.6 Maximum Resolvable Reflector Height

The "Nyquist" is complicated for GNSS-IR for various reasons - one being the units are not the same as the units of what we care about, the Reflector Height. So I am going to call it the Maximum Resolvable Reflector Height, which is a mouthfull, but at least you have some idea what it means.
If you are interested in the details of this calculation, please see the Roesler and Larson paper. If you want to compute it for your site, please use max_resolve_RH That's all I am going to say on the matter.

2.7 Refraction

We would welcome help from the community to add a discussion here of refraction models and GNSS-IR. Currently we only provide an explanation of refraction corrections inside of our code, specifically inside gnssir_input.py, which is where the refraction model is set. Please see that documentation for details.

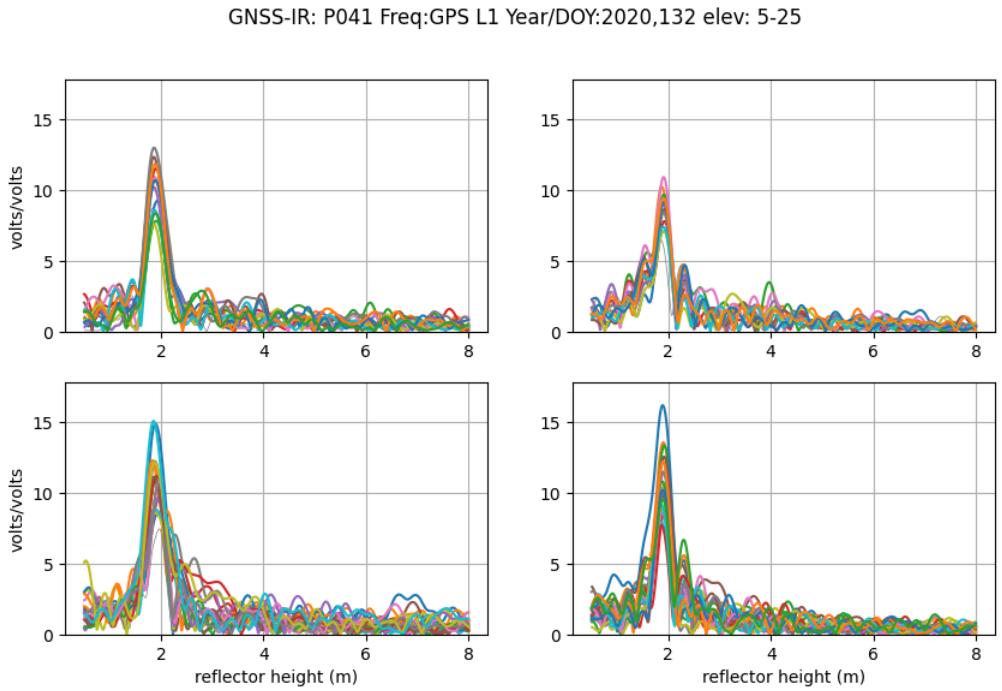
2.8 quickLook

quickLook is meant to provide the user with a visual sense of the data at a given site. It has stored defaults that work for stations with reflectors that are lower than 8 meters. You can change those defaults on the command line.

Example from Boulder

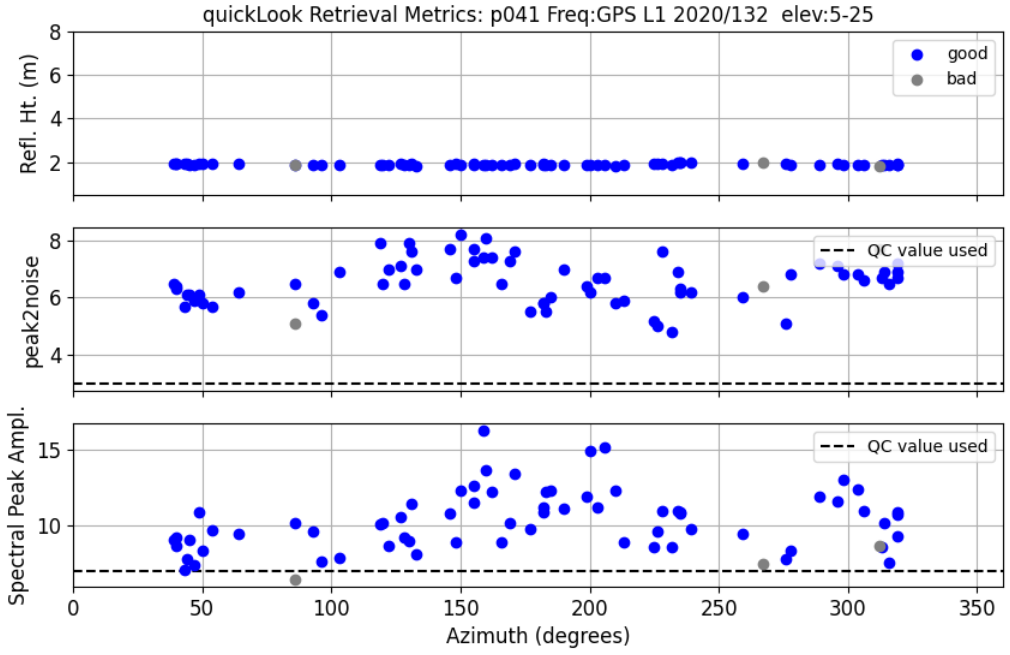
quickLook p041 2020 132

That command will produce this periodogram summary :



By default, these are L1 data only. Note that the x-axis does not go beyond 8 meters. This is because we have used the defaults. Furthermore, note that results on the x-axis begin at 0.5 meters. Since you are not able to resolve very small reflector heights with this method, this region is not allowed. These periodograms give you a sense of whether there is a planar reflector below your antenna. The fact that the peaks in the periodograms bunch up around 2 meters means that at this site the antenna phase center is ~ 2 meters above the ground. The colors represent different satellites. If the data are plotted in gray that means you have a failed reflection. The quadrants are Northwest, Northeast and so on.

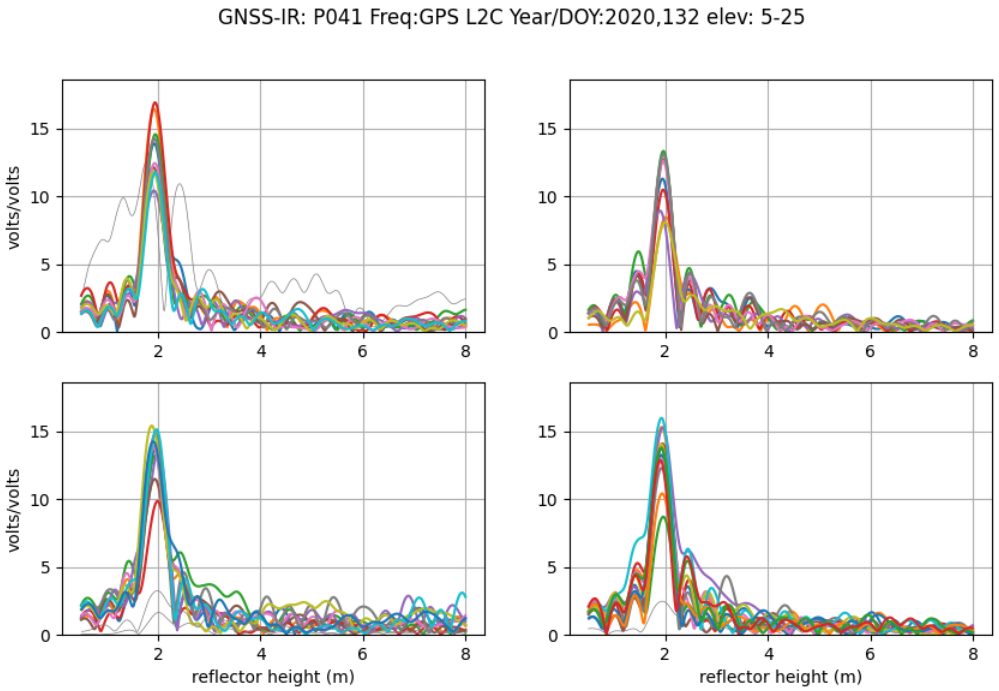
quickLook also provides a summary of various quality control metrics:



The top plot shows the sucessful RH retrievals in blue and unsuccessful RH retrievals in gray. In the center panel are the peak to noise ratios. The last plot is the amplitude of the spectral peak. The dashed lines show you what QC metrics quickLook was using. You can control/change these on the command line.

If you want to look at L2C data you just change the frequency on the command line. L2C is designated by frequency 20:

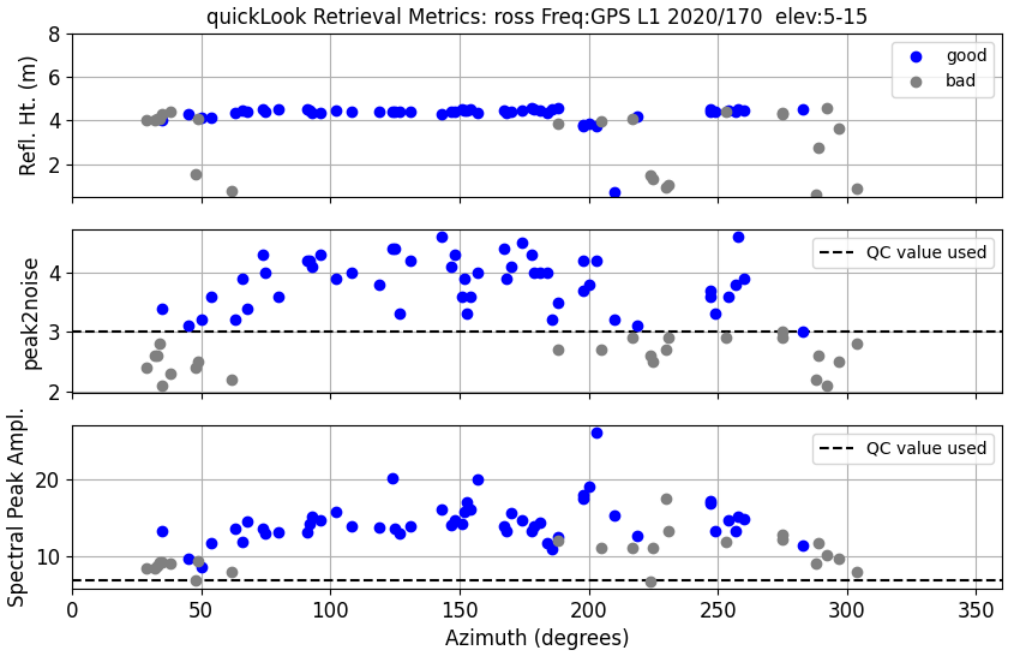
quickLook p041 2020 132 -fr 20



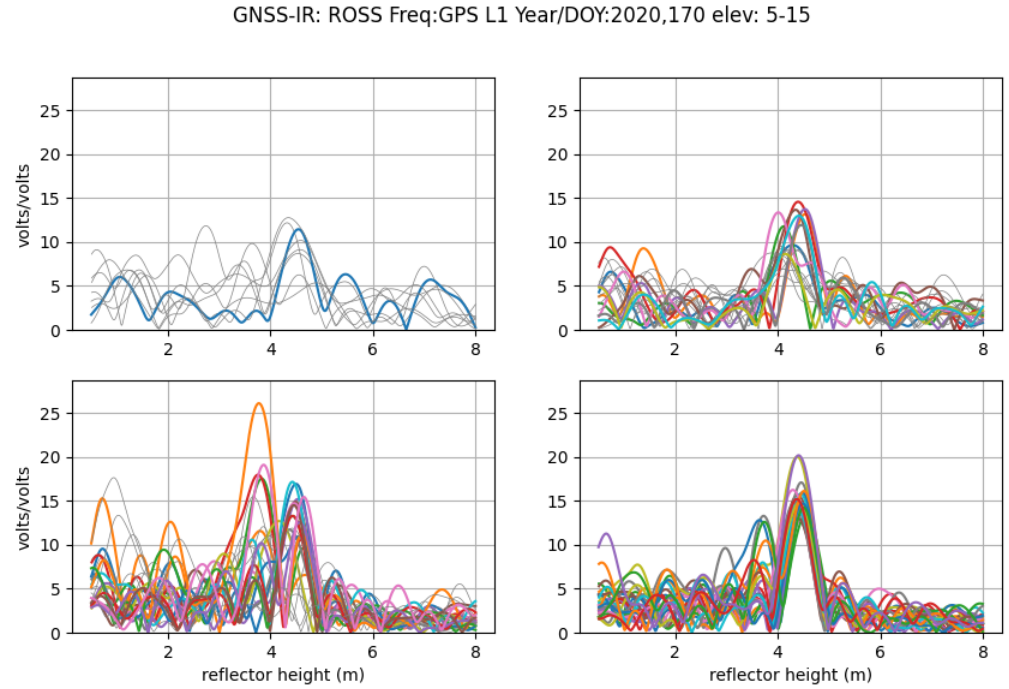
L2C results are always superior to L1 results. They are also superior to L2P data. If you have any influence over a GNSS site, please ask the station operators to track modern GPS signals such as L2C and L5 **and** to include it in the archived RINEX file.

Example for Lake Superior

quickLook ross 2020 170 -e1 5 -e2 15



The good RH estimates (in blue in the top panel) are telling us that we were right when we assessed reflection zones using 4 meters. We can also see that the best retrievals are in the southeast quadrant (azimuths 90-180 degrees). This is further emphasized in the next panel, that shows the actual periodograms.



Example for a site on an ice sheet

Example for a taller site on an ice sheet

Warning: quickLook calculates the minimum observed elevation angle in your file and prints that to the screen so you

know what it is. It also uses that as your emin value (e1) if the default is smaller. It does this so you don't see all arcs as rejected. Let's say your file had a receiver-imposed elevation cutoff of 10 degrrees. The default minimum elevation angle in quickLook is 5 degrees. With the default **ediff** value of 2, not a single arc would reach the minimum required value of 7 (5 + 2); everything would be rejected. quickLook instead sees that you have a receiver-imposed minimum of 10 and would substitute that for the default emin. gnssir does not do this because at that point you are supposed to have chosen a strategy, which is stored in the json file.

quickLook -screenstats True provides more information to the screen about why arcs have been rejected.

2.9 Looking at raw SNR data

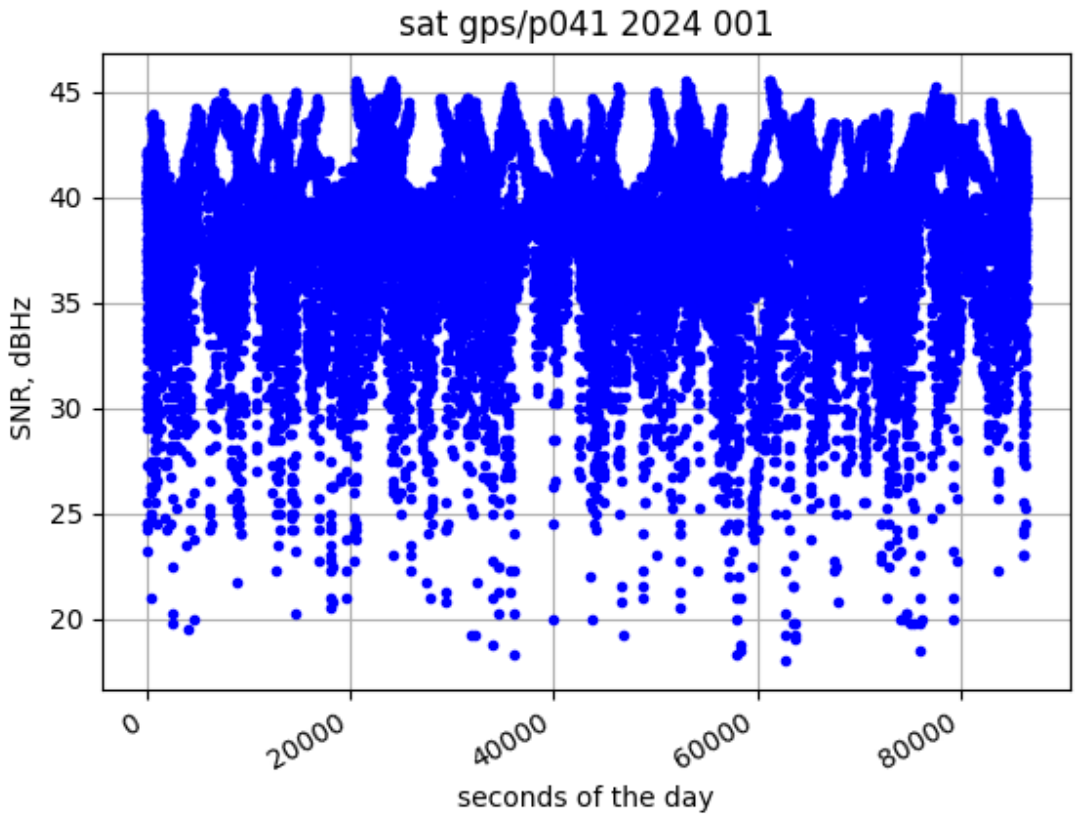
This code was not developed with an emphasis on *looking* at raw SNR data. But, I can offer a workaround. You can use quickplt. This works on SNR files - so you do have to at least translate either RINEX or NMEA data first. Because I find it annoying to have to provide the full path and gunzip files, it will add that path and gunzip for you if necessary. I'll just give one example.

Assume you have a SNR file. I will use rinex2snr p041 2024 1 -archive sopac. It creates

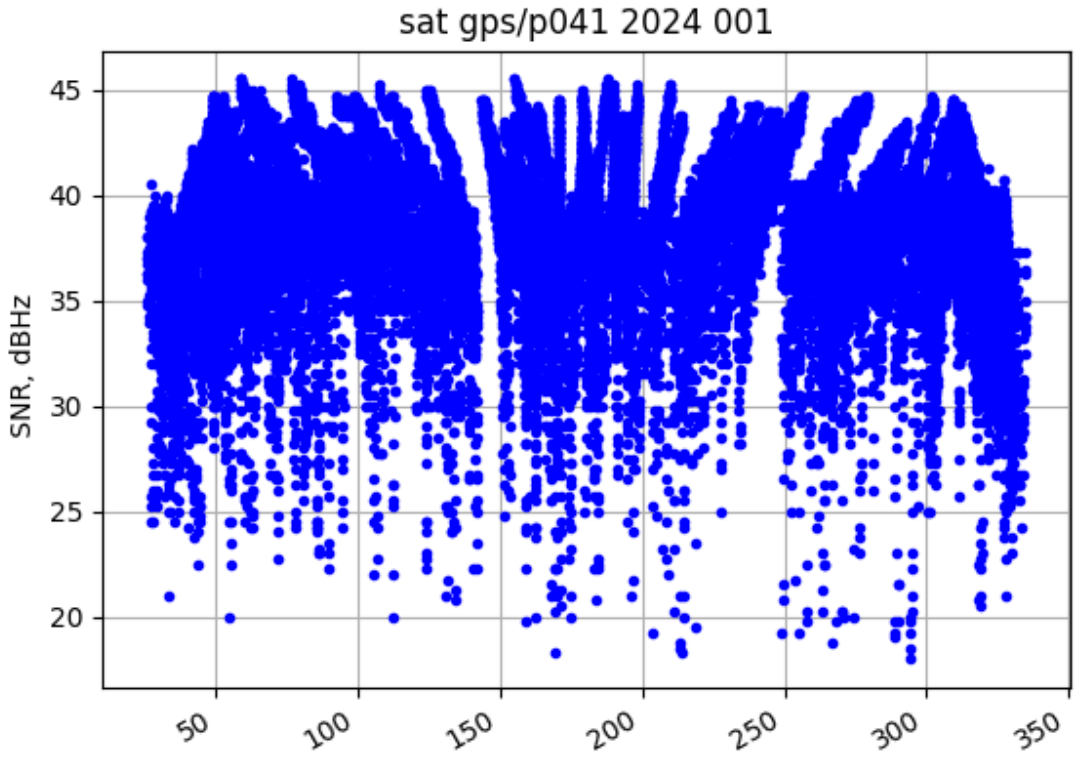
SNR file p0410010.24.snr66 which is stored in $REFL_CODE/2024/snr/p041 From the file formats discussion you should know that column 2 is elevation angle, azimuth is column 3, time is column 4, and L1 SNR is in column 7. If i wanted to plot the raw SNR GPS L1 data for this file:

quickplt p0410010.24.snr66 4 7 -sat gps

The -sat option tells quickplot you are working with a SNR file.



quickplt p0410010.24.snr66 2 7 -sat gps would show azimuth on the x-axis



You can also ask for glonass, galileo, beidou etc. You can change symbols, restrict x-axis, etc with quickplt. Check the documentation for more detail.

You can also specify a single satellite if you want (make sure you add 100 for Glonass, etc).

If you want to look at the SNR data after the direct signal has been removed, you can use gnssir -savearcs T which has a beta version that prints plain text files for each arc. However, this is beta - I am unlikely to fix bugs in it. Please submit a PR if you find bugs.

2.10 gnssir

gnssir_input

A full listing of the possible inputs and examples for gnssir_input can be found here.

Your first task is to define your analysis strategy. We use station p101 as an example. If the station location is in our database:

gnssir_input p101

If you have your own site, you should use -lat, -lon, -height as inputs.
If you happen to have the Cartesian coordinates (in meters), you can set -xyz True and input those instead.

The json file of instructions will be stored in $REFL_CODE/input/p101/p101.json.

The default azimuth inputs are from 0 to 360 degrees. You can set your preferred azimuth regions using -azlist2. Previously you were required to use multiple azimuth regions, none of which could be larger than 100 degrees. That is no longer required. However, if you do need multiple distinct regions, that is allowed, e.g.

gnssir_input p101 -azlist2 0 90 180 270

If you wanted all southern quadrants, since these are contiguous, you just need to give the starting and ending azimuth.

gnssir_input p101 -azlist2 90 270

You should also set the preferred reflector height region (h1 and h2) and elevation angle mask (e1 and e2). Note: the reflector height region should not be too small, as it is also used to set the region for your periodogram. If you use tiny RH constraints, your periodogram will not make any sense and your work will fail the quality control metrics.

gnssir

gnssir estimates reflector heights. It assumes you have made SNR files and defined an analysis strategy. The minimum inputs are the station name, year, and doy.

gnssir p041 2020 150

Additional inputs

Where would the code store the files for this example?

- analysis instructions are stored in $REFL_CODE/input/p041/p041.json
- SNR files are stored in $REFL_CODE/2020/snr/p041
- Reflector Height (RH) results are stored in $REFL_CODE/2020/results/p041/150.txt

For more information about the decisions made in gnssir, set **-screenstats T**

To have plots come to the screen, set -plt to T or True.

If you want to try different strategies, make multiple json files with the -extension input. Then use the same -extension command in gnssir.

This is a snippet of what the result file would look like

```
% gnssrefl, https://github.com/kristinemlarson
% Phase Center corrections have NOT been applied
% year, doy, RH, sat,UTCtime, Azim, Amp,  eminO, emaxO,NumbOf,freq,rise,EdotF, PkNoise  DelT     MJD   refr-appl
% (1)  (2)   (3) (4)  (5)     (6)   (7)    (8)    (9)   (10)  (11) (12) (13)    (14)     (15)    (16)   (17)
%            m        hrs    deg   v/v    deg    deg  values              hrs            min         1 is yes
 2020 151  1.925   1 14.952 120.40   7.90   5.14  24.95  260   1  1  0.84517   5.59   64.75 58999.622998 1
 2020 151  1.995   1  9.410 225.31   6.51   5.18  24.93  236   1 -1 -0.75974   3.91   58.75 58999.392095 1
 2020 151  1.930   1  5.717 319.30   7.74   5.16  24.91  207   1  1  0.67554   5.15   51.50 58999.238194 1
 2020 151  1.880   2 18.277 199.81  10.58   5.22  24.94  182   1  1  0.58884   5.58   45.25 58999.761539 1
 2020 151  1.920   3 12.002 190.39   7.44   5.16  24.94  198   1 -1 -0.63144   4.68   49.25 58999.500081 1
 2020 151  1.910   3  7.115 306.63   7.21   5.19  24.97  216   1  1  0.69272   4.59   53.75 58999.296435 1
 2020 151  1.892   4 14.154 160.30  10.03   5.14  24.99  187   1 -1 -0.59952   6.32   46.50 58999.589757 1
 2020 151  1.970   5  1.456  54.56   7.63   5.21  24.94  218   1 -1 -0.70039   4.20   54.25 58999.060671 1
 2020 151  1.950   5 20.506 171.31  10.14   5.17  24.97  198   1  1  0.63146   5.63   49.25 58999.854421 1
 2020 151  1.940   6 22.623  62.47   7.28   5.18  24.91  238   1 -1 -0.76263   4.88   59.25 58999.942616 1
 2020 151  1.900   6 17.331 181.76   8.14   5.15  24.96  186   1  1  0.59437   4.77   46.25 58999.722130 1
 2020 151  1.925   7 17.185 148.30   6.95   5.16  24.93  186   1 -1 -0.60020   4.53   46.25 58999.716053 1
 2020 151  1.977   8 15.860  44.52   8.07   5.19  24.96  208   1 -1 -0.67217   4.96   51.75 58999.660845 1
 2020 151  1.880   8 11.648 150.39   8.53   5.16  24.94  214   1  1  0.68352   5.86   53.25 58999.485324 1
 2020 151  1.897   9 15.227 178.16   7.34   5.22  24.89  190   1 -1 -0.61098   4.09   47.25 58999.634456 1
```

Note that the names of the columns (and units) are provided (this may be out of date):

- *Amp* is the amplitude of the most significant peak in the periodogram (i.e. the amplitude for the RH you estimated).
- *DelT* is how long a given rising or setting satellite arc was, in minutes.
- *emin0* and *emax0* are the min and max observed elevation angles in the arc.
- *rise/set* tells you whether the satellite arc was rising (1) or setting (-1)

- *Azim* is the average azimuth angle of the satellite arc
- *sat* and *freq* are as defined for gnssrefl (i.e. 101 is Glonass L1)
- MJD is modified julian date
- PkNoise is the peak to noise ratio of the periodogram values
- last column is currently set to tell you whether the refraction correction has been applied
- EdotF is used in the RHdot correction needed for dynamic sea level sites. The units are hours/rad. When multiplied by RHdot (meters/hour), you will get a correction in units of meters. For further information, see the subdaily code.

Kristine M. Larson

August 8, 2024

CHAPTER

THREE

FILES, FORMATS, FREQUENCIES

3.1 Environment Variables

You need three environment variables to run this code: REFL_CODE, ORBITS, and EXE. If you are using the jupyter notebooks or the docker, they are defined for you.

If you are working with pypi or github clone install, you must define them EVERY TIME YOU USE THE CODE. This is most easily done by setting them in your setup script, which on my machine is called .bashrc.

If you are working with the docker, these should all be set up for you. But knowing that they exist can be helpful in looking for files, etc.

3.2 How do I collect my own GNSS data?

We do not have instructions in this software package for how you can operate your own receiver for GNSS-IR. Currently we need you to save your observation data as Rinex 2.11, Rinex 3, or NMEA formats (see below). At a minimum you **must** save the SNR data; we strongly urge you to track/save **modern GPS signals**, whicha are L2C and L5. If you have multi-GNSS capabilities, we strongly encourage you to use them. And never use an elevation mask on your receiver. They are completely unncessary for positioning (which allows masking to be done at the software level) and are extremely harmful to GNSS-IR.

3.3 Where should I store station coordinates?

The software comes with a long list (almost 20,000) of station coordinates taken from the University of Nevada Reno. If you are analyzing data from any of those stations, you should not have to enter any coordinates (Note: you can use **query_unr** to see if your station is included in the UNR database).

If you are analyzing your own data, eventually you will need to tell the software where your stations are. This location does not have to be super precise, within a few meters is perfectly acceptable, as it is primarily used for the refraction correction. The better your site coordinates, the better your reflection zone maps would be, however. Previously you input this information (latitude, longitude, and ellipsoidal height) when you set your analysis strategy in **gnssir_input**.

As of version 3.6.4, there is now another option. If you create a plain txt file with the name llh_local.txt and store it in the $REFL_CODE/input directory, the code will use this as your *a priori* station coordinates. The format of this file for **each line** should have four entries with spaces between them. These entries are the four character station name, the latitude, the longitude, and ellipsoidal height. The units of the last four are degrees, degrees, and meters. Example for a station called **xxxx**:

You can add comment lines to the file with a percent sign. This file is read in the query_coordinate_file function found in gps.py. The local coordinate file is read by nmea2snr. This means you no longer have to enter station coordinates on the command line when using that code.

3.4 How do I analyze my own GNSS data?

To analyze your own GNSS data you must comply with the software expectations for how the files should be named. The naming conventions for GNSS observation files are given in the next section. You are always better off to use lowercase for Rinex 2.11. For RINEX 3 we follow the convention of the GNSS archives, where the first part is upper case and the stem is lower.

If you are working with the docker, I have made some notes in the docker install section that might be helpful to you about where to store your files.

If you are working with git clone or pypi install, you should be able to have the RINEX files in the directory you are currently working in. Or you should put them in the rinex directory as defined below in the *Where Files are Stored* section, i.e. $REFL_CODE/YYYY/rinex/abcd where abcd is the station name.

Examples are given in the rinex2snr code. Documentation can always be improved, so if you would like to add more examples or find the current documentation confusing, please submit a pull request. If you don't know how to read the documentation, I suggest you try typing rinex2snr -h. You will notice that there is an option called nolook. **You have to set this to True.**

Example rinex2snr p041 2025 191 -nolook T will look on your coputer for a RINEX 2.11 file called p0411910.25o. It might look for the Hatanaka compressed file too - I can't remember.

You are telling the code not to look for the data at an archive. Perhaps that is not the best name for an option, but that is what it is. If you don't say -nolook T, it will think you want a file in a global GNSS archive and well, it won't find it if you have the only copy of your data.

If you are using the gnssrefl notebooks, unfortunately no notebook was developed by Earthscope for this option. I am unable to provide you with any assistance. If you would like to share such a notebook as a PR, that would be great.

If you have questions about converting NMEA files, the best I can offer is that you read the next section on that specific format. The command is nmea2snr.

Many file conversion programs produce orbit files as well as observation files. These orbit files are unnecessary in this software package. The code is set up to find the appropriate orbit files for you.

3.5 GPS/GNSS Observation Data Formats

Please keep in mind that there are multiple issues here:

- Are your observation files stored in what gnssrefl considers to be a compliant format?
- Are your observation files properly named?
- Are your observation files stored where the code expects to find them?
- Do your observation files include the data we need for GNSS-IR (the SNR observables)
- Did you compress you file in some way - and does gnssrefl recognize this kind of compression? (Hatanaka, gzip, Z, etc etc)

Unfortunately all of these issues come into play, and it can be confusing to figure out where the problem is. We have tried as best we can to make screen output that will help you with your problem.

Input observation formats: the code only recognizes RINEX 2.11, RINEX 3 and NMEA input files.

3.5.1 RINEX 2.11

We strongly prefer that you use lower case filenames. I cannot promise you that the code will find files that are stored in uppercase. Lowercase filenames are the standard at global archives. They must have SNR data in them (S1, S2, etc) and have the receiver coordinates in the header. The files should follow these naming rules:

- all lowercase
- station name (4 characters) followed by day of year (3 characters) then 0.yyo where yy is the two character year.
- Example: algo0500.21o where station name is algo on day of year 50 from the year 2021

It is also standard to use the Hatanaka files. Instead of ending in an o the Hatanaka files end in a d.

Example filename : onsa0500.22d

We also generally allow two kinds of compression, unix compression and gzip:

Unix compression example filename : onsa0500.22d.Z

gzip example filename : onsa0500.22o.gz

We do not make any effort to find files with the zip ending. If your files have this ending, you must unzip them before running gnssrefl.

3.5.2 RINEX 3

While we support RINEX 3 files, we do not read the RINEX 3 file itself - we rely on the gfzrnx utility developed by Thomas Nischan at GFZ to translate from RINEX 3+ to RINEX 2.11 If you have RINEX 3 files, they should be all upper case (except for the extension rnx or crx).

Example filename: ONSA00SWE_R_20213050000_01D_30S_MO.rnx

- station name (9 characters where the last 3 characters are the country), underscore
- capital R or capital S , with underscore on either side
- four character year
- three character day of year
- four zeroes, underscore,
- 01D, underscore
- ssS, underscore, M0.
- followed by rnx (crx if it is Hatanaka format). Note: these are lowercase

01D means it is one day. Some of the other parts of the very long station file name are no doubt useful, but they are not recognized by this code. By convention, these files may be gzipped but not unix compressed. If you want a generic translation program, you can try rinex3_rinex2. It has the requirement that you input the input and output RINEX file names.

For a few archives, we allow 1 sample per second files. Following the protocol of the IGS, these files are unfortunately 15 minutes long, which means **you have to download 96 of them.** UNAVCO/Earthscope is much more sensible about providing 1 sample per second files, and returns a single file, at least for the RINEX 2.11 format.

If you want the code to be able to find those highrate files, you must tell the code you want to use the -rate high files and provide -samplerate 1. Why two inputs? Because the -rate high option tells the code to look in a particular folder. The samplerate is related to the name of the file itself.

Recently the IGS and its sister archives have started making a single tar file of the 96 daily files after six months. gnssrefl now allows access to these older data from CDDIS and BKG.

Please see the rinex2snr documentation page for more examples.

3.5.3 NMEA

NMEA formats can be translated to SNR using nmea2snr. Inputs are similar to that used by rinex2snr: the 4char station name, the year, and day of year. NMEA files are assumed to be stored as:

$REFL_CODE + /nmea/ABCD/2021/ABCD0030.21.A

for station ABCD in year 2021 and day of year 3.

NMEA files may be gzipped.

This is different than the file structure we used for RINEX files and is entirely due to the wishes of the people that contributed this code. If you would like the code to also allow a traditional folder location ($REFL_CODE/2021/nmea/abcd or REFL_CODE/2021/nmea/ABCD), I am fine with that. I ask that you please submit a pull request.

Additional information about nmea2snr is in the code.

3.6 ORBITS

We have tried our best to make the orbit files relatively invisible to users. But for the sake of completeness, we are either using broadcast navigation files in the RINEX 2.11 format or precise orbits in the sp3 format. If you have nav files for your station, we recommend you delete them. They are not useful in this code.

The main things you need to know:

- if your files only have GPS data in them, there is no need to use multi-GNSS SP3 files. Flag -nav T
- if your files are multi-GNSS, the best option is gnss, which are final orbits. This is complicated for older data. Those files are reliably available from 2023. And they cover the four main constellations.
  My current default is rapid GNSS - but that does not always have Beidou in it.
- we also have ultra-orbit options, which are appropriate for real-time users. I cannot keep track of what ultra products are working. You can try ultra, wum, and wum2. The first is from GFZ and the latter two are from Wuhan. The second file comes from Wuhan directly while the first (I believe) comes from the one stored at CDDIS.

3.7 EXECUTABLES

There are two key executables: CRX2RNX and gfzrnx. For notebook and docker users, these are installed for you. pypi/github users must install them. The utility installexe should take care of this. They are stored in the directory defined by the EXE environment variable. We used to support the use of teqc but as Earthscope no longer provides technical support for it, we have mostly eliminated it.

3.8 Where Files are Stored

File structure for station abcd in the year YYYY (last two characters YY), doy DDD:

- REFL_CODE/input/abcd/abcd.json - instructions for gnssir analysis, refraction files
- REFL_CODE/YYYY/snr/abcd/abcdDDD0.YY.snr66 - SNR files
- REFL_CODE/YYYY/rinex/abcd/ - RINEX files of various flavors can be stored here
- REFL_CODE/YYYY/results/abcd/DDD.txt Lomb Scargle analysis goes here
- REFL_CODE/YYYY/phase/abcd/DDD.txt phase analysis
- REFL_CODE/Files/ - various output files and plots will be placed here. For water levels, everything is stored with an additional folder with the station name
- ORBITS/YYYY/nav/autoDDD0.YYn - GPS broadcast orbit file
- ORBITS/YYYY/sp3/ - sp3 files of orbits - these use names from the archives.

The RINEX files downloaded from archives are not stored by this code. Or at least not deliberately. If they are being translated, they are deleted.

If translating your own files, you should take care to not keep your only copy in your default directory. If they are stored in $REFL_CODE/YYYY/rinex/abcd you will be fine.

You do not need precise orbits to do GNSS-IR. We only use them as a convenience. Generally we use multi-GNSS sp3 files. See the rinex2snr documentation for more details on the orbits you can use.

Some of the utilities and environmental products code store files in REFL_CODE/Files The locations of these files are always provided in the screen output.

The inputs to gnssir are generally stored in the REFL_CODE/input folder. This primarily means the Lomb Scargle data analysis inputs, i.e. the "json" files, e.g. p041.json for station p041. It also includes the refraction file (p041_refr.txt) that is created automatically. This calculation requires a set of parameters stored in a "pickle" format, gpt_1wA.pickle. This file should be automatically stored for you.

3.9 The SNR data format

Reminder: UTC does not exist in our world. Everything should be GPS time, which is UTC without leap seconds.

The snr options are mostly based on the need to remove the "direct" signal. This is not related to a specific site mask and that is why the most frequently used options (99 and 66) have a maximum elevation angle of 30 degrees. The azimuth-specific mask is decided later when you run **gnssir**. The SNR choices are:

- 66 is elevation angles less than 30 degrees (**this is the default**)
- 99 is elevation angles of 5-30 degrees
- 88 is all data
- 50 is elevation angles less than 10 degrees (good for very tall sites, high-rate applications)

66,99, etc are not good names for files. And for this I apologize. It is too late to change them now. The SNR files are not self-documenting (i.e. there is no header) and for that I also apologize.

The columns in the SNR data are defined as:

- Satellite number (remember 100 is added for Glonass, 200 for Galileo, 300 for Beidou)
- Elevation angle, degrees

- Azimuth angle, degrees
- **Seconds of the day, GPS time**
- elevation angle rate of change, degrees/sec.
- S6 SNR on L6
- S1 SNR on L1
- S2 SNR on L2
- S5 SNR on L5
- S7 SNR on L7
- S8 SNR on L8

The unit for all SNR data is dB-Hz. In some cases these frequencies come from RINEX conventions. The constellation people themselves do not use them. See below for more information.

I do not currently use SBAS signals, but they would be easy to add if someone wants to make a PR. I suggest 400 be added to such signals and someone needs to define the various organizations that run SBAS satellites.

3.10 GNSS frequencies

gnssrefl uses numbers for the frequencies that are defined in the RINEX files:

- 1,2,20, and 5 are GPS L1, L2, L2C, and L5
- 101,102 are Glonass L1 and L2
- 201, 205, 206, 207, 208 are Galileo frequencies, which are set as 1575.420, 1176.450, 1278.70, 1207.140, 1191.795 MHz
- 302, 306, 307 are Beidou frequencies, defined as 1561.098, 1207.14, 1268.52 MHz

This means that if you want the Beidou frequency data from 1561.098, you need to look in the "L2" column. If you want the 1278.70 Galileo frequency, you need to look in the "L6" column.

3.11 Additional files

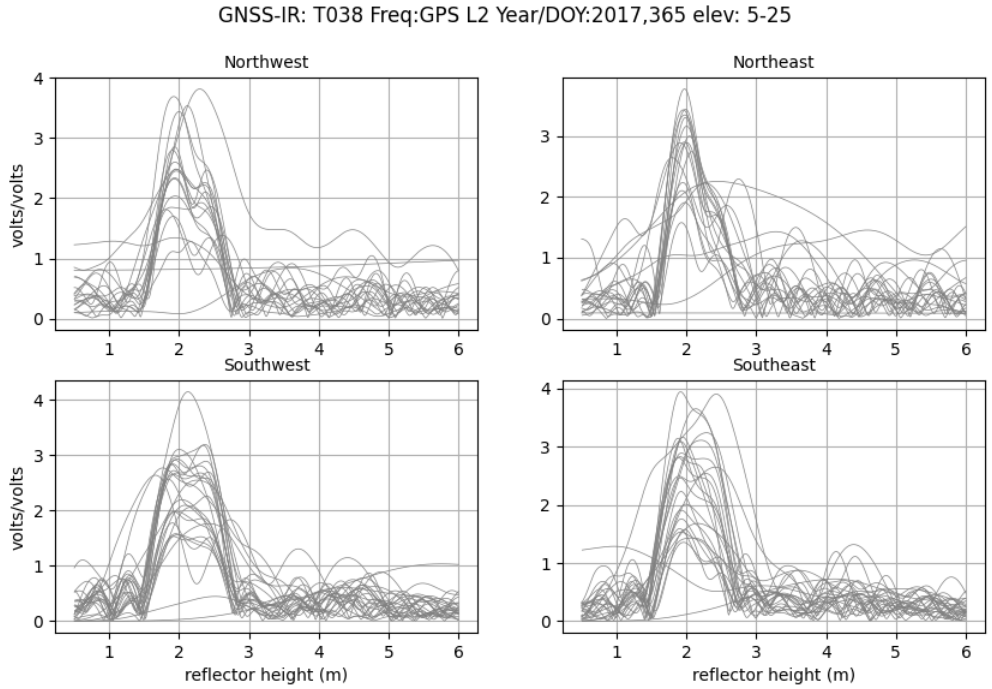
- EGM96geoidDATA.mat is stored in REFL_CODE/Files
- station_pos2024.db is stored in REFL_CODE/Files. This is an updated compilation of station coordinates from Nevada Reno.
- gpt_1wA.pickle is stored in REFL_CODE/input. This file is used in the refraction correction.
- GPSorbits_21sep17.txt, GALILEOorbits_21sep17.txt, etc are stored in REFL_CODE/Files. These are used for refl_zones and max_resolve_RH
- leapsecond file, ONLY for nmea2snr, REFL_CODE/Files/leapseconds.txt

3.12 Some comments about signals

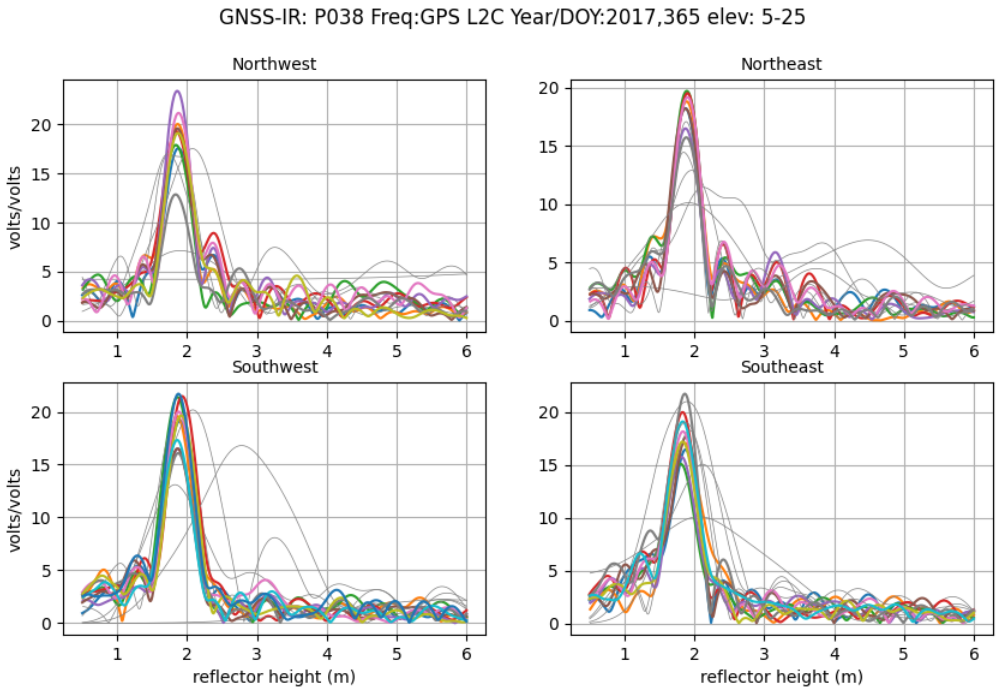
3.12.1 GPS L2C

Why do I like L2C? What's not to like? It is a modern **civilian** code without high chipping rate. That civilian part matters because it means the receiver knows the code and thus retrievals are far better than a receiver having to do extra processing to extract the signal. Here is an example of a receiver that is tracking **both** L2P and L2C. Originally installed for the Plate Boundary Observatory, it is a Trimble. The archive (unavco) chose to provide only L2P in the 15 second default RINEX file. However, it does have the L2C data in the 1 second files. So that is how I am able to make this comparison. P038 is a very very very flat site.

Here are the L2P retrievals:



Now look at the L2C retrievals.



If you were trying to find a periodic signal, which one would you want to use?

To further confuse things, when the receiver was updated to a Septentrio, unavco began providing L2C data in the default 15 second files. This is a good thing - but it is confusing to people that won't know why the signal quality improved overnight.

L2C is easy to extract from RINEX 3 files - and that is what is done by rinex2snr. However, I do not make the same restriction for RINEX 2.11 files. In principle I could, but for now, I translate all L2 signals in a RINEX 2.11 file. When you subsequently chose L2C (frequency 20) in gnssir, you will be given results for all GPS satellites that could be L2C. The list of L2C transmitting satellites is found in the gps.py library.

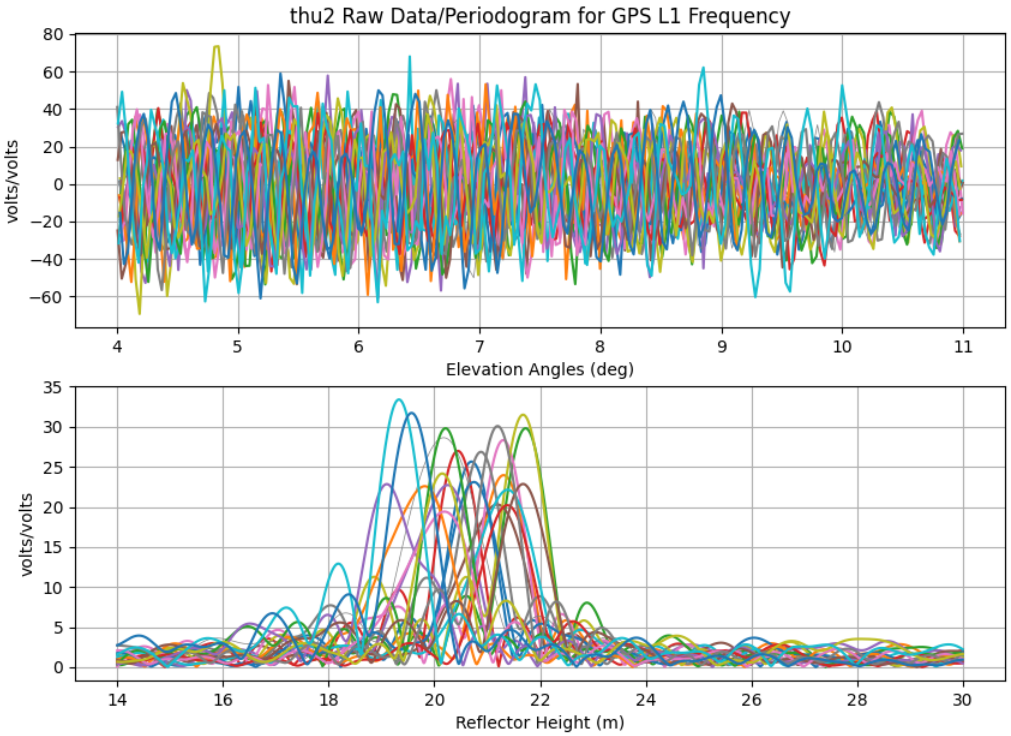
3.12.2 GPS L5

Another great signal. I love it. It does have a high chipping rate, which is relevant (i.e. bad) for reflectometry from very tall sites.

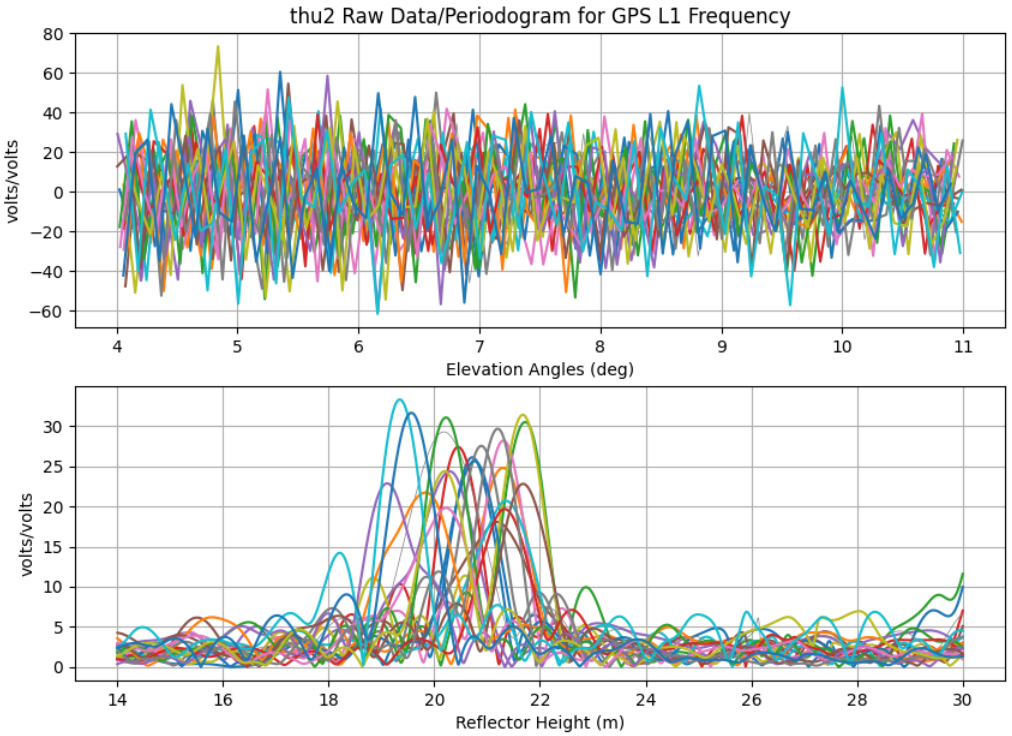
3.12.3 Aliasing

While it will show up in GPS results too - there seems to be a particularly bad problem with Glonass L1. I used an example from Thule. The RH is significant - ~20 meters. So you absolutely have to have at least 15 sec at the site or you violate the Nyquist. Personally I prefer to use 5 sec - which means I have to download 1 sec and decimate. This is extremely annoying because of how long it takes to ftp those files to my local machine. Let's look at L1 solutions using a 5 second file - but where I invoke the -dec option for gnssir. That way I can see the impact of the sampling. I also using the -plt T option.

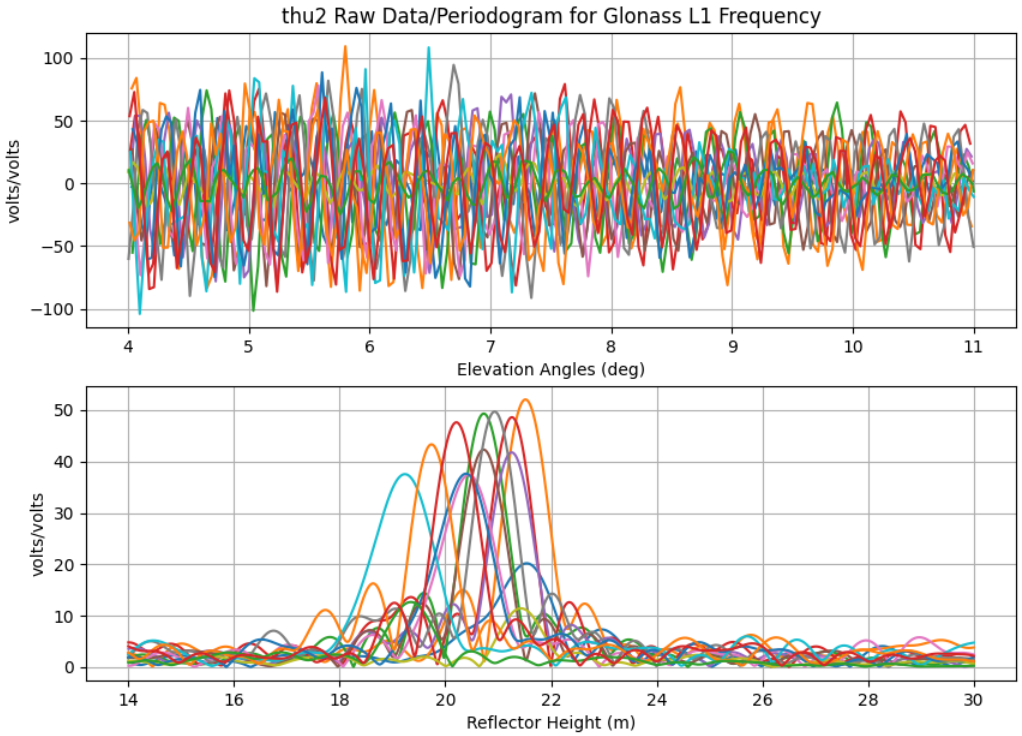
This is 5 second GPS L1.



This is 15 second GPS L1. You see some funny stuff at 30 meters, and yes, the periodograms are noisier. But nothing insane.



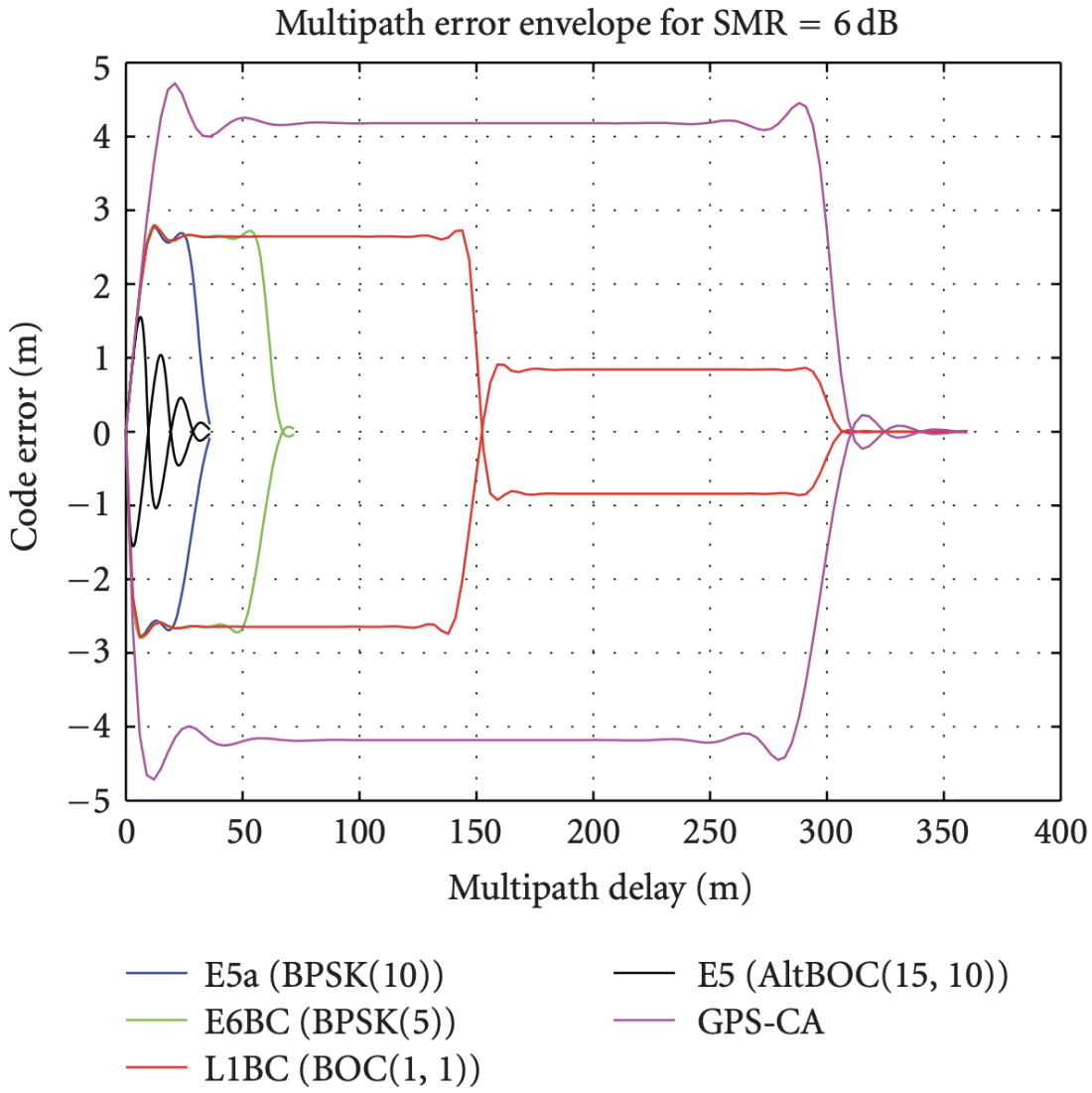
Now do 5 second Glonass L1



Contrast with the Glonass L1 results using 15 sec decimation! So yeah, aliasing is a problem.

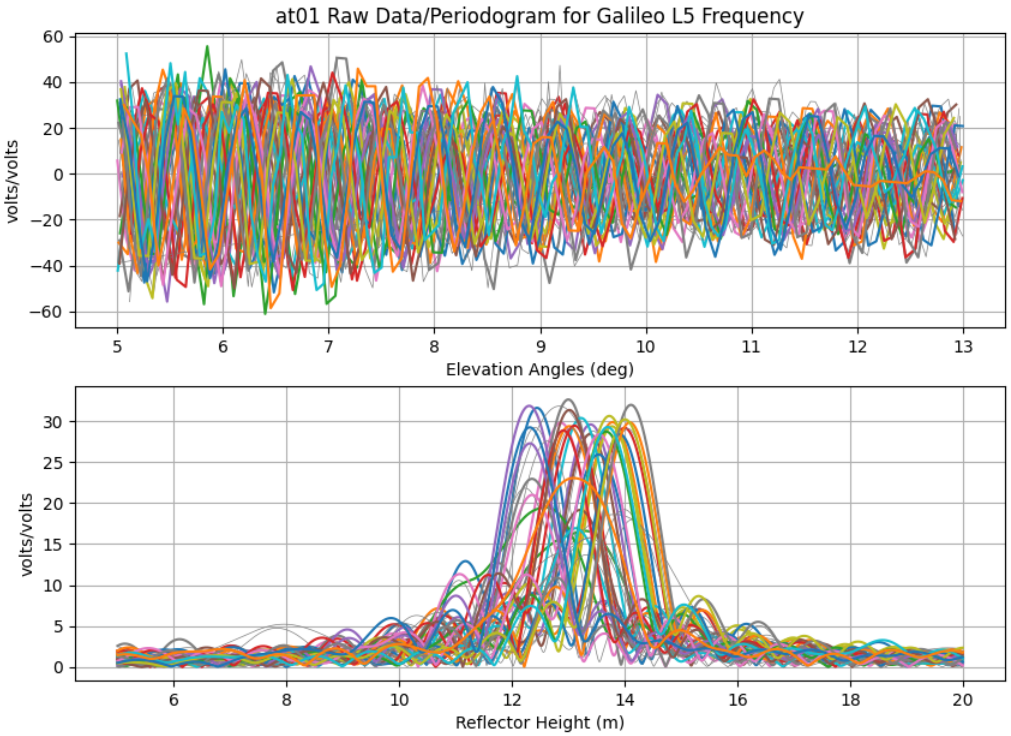
3.12.4 E5

Now about RINEX L8 . . . also known as E5. This is one of the new Galileo signals. Despite the fact that it is near the frequencies of the other L5 signals, it is **not** the same. You can see that it in the multipath envelope work of Simsky et al. shown below.

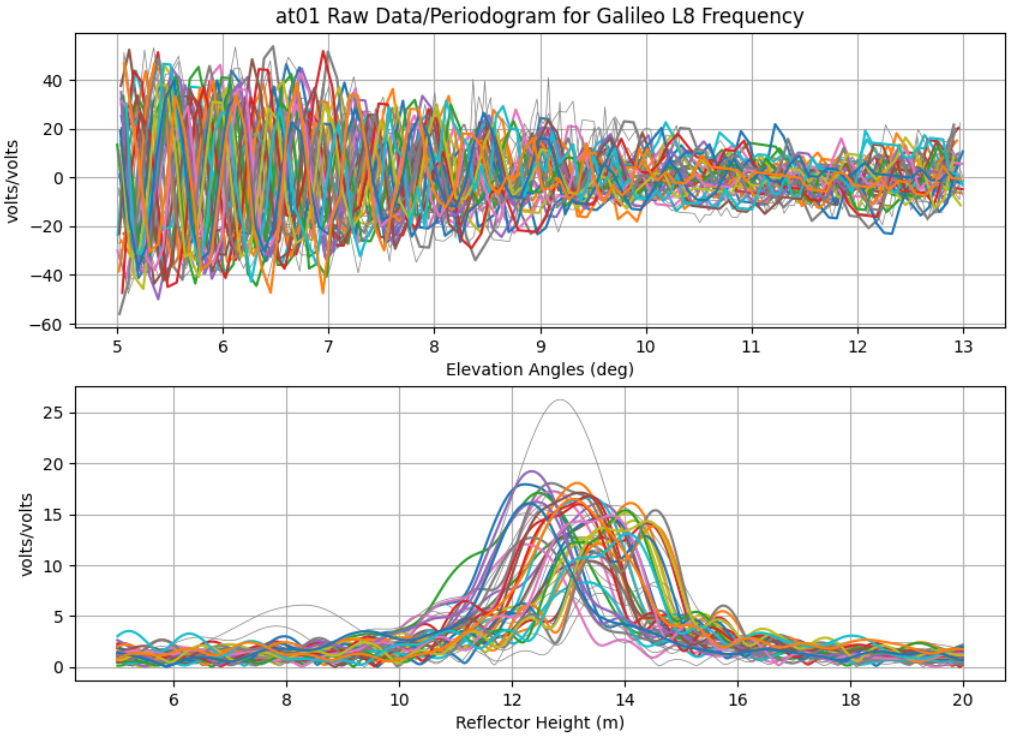


Most of you will not be familiar with multipath envelopes - but for our purposes, we want those envelopes to be big - cause more multipath, better GNSS-IR. First thing, multipath delay shown on the x-axis is NOT the reflector height (RH). it is $2RH\sin$(elevation angle). So even a pretty tall RH will not be obstructed by the new Galileo codes except for E5.

This is E5a



This is E5. Note that instead of nice clean peaks, it is spread out. You can also see that the E5 retrievals degrades as elevation angle increases, which is exactly what you would expect with the multipath delay increasing with elevation angle. I would just recommend only using this signal for RH < 5 meters. And even then, if you are tracking L8, you probably also have L5, L6, and L7, so there is not a ton gained by also using L8.

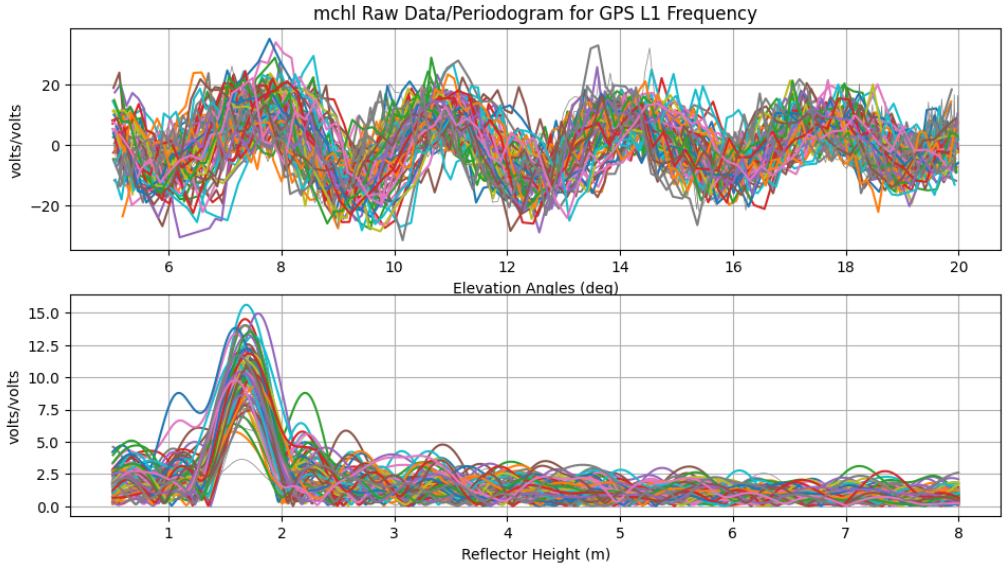


3.12.5 L1C

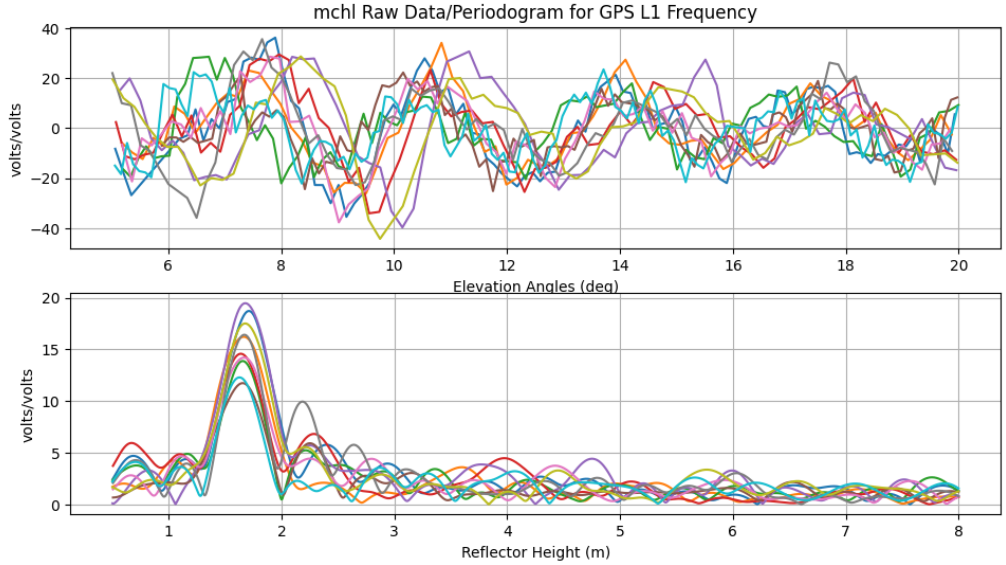
The Trimble Alloy receiver at Mitchell (MCHL) in Australia records both legacy L1 and modern L1C signals, which are available in the daily RINEX 3 files. RINEX 3 names the legacy signal "L1C" and the modern signal "L1X" (the SNR observables are thus "S1C" and "S1X"). RINEX 2 does not support both signals simultaneously, so two separate files must be created. For example, using gfzrnx:

```
gfzrnx -finp FILE.rnx -fout OUTPUT_L1C.rnx -ot S1C -vo 2
gfzrnx -finp FILE.rnx -fout OUTPUT_L1X.rnx -ot S1X -vo 2
```

These files can be renamed and processed using rinex2snr/gnssir. First, the legacy results for MCHL 2023/day 1:



And the modern signal results, showing marginally higher peaks in the LSP:



There is less data for L1X as the modern signal is only broadcast by Block III satellites, of which there are currently only 10 (in Feb 2025).

The multipath envelope figure is taken from:

Title: Experimental Results for the Multipath Performance of Galileo Signals Transmitted by GIOVE-A Satellite

Authors: Andrew Simsky,David Mertens,Jean-Marie Sleewaegen, Martin Hollreiser, and Massimo Crisci

International Journal of Navigation and Observation Volume 2008, DOI 10.1155/2008/416380

CHAPTER

FOUR

QUICK LINKS/EXPERT MODES

I originally made special documentation pages for different modules. I am now trying to put all documentation INTO the code intself using readthedocs since the format allows for example calls. While I still have some discussion pages (so a link to the code and a link to a discussion page), those discussion links will likely go away in the not very distant future.

4.1 Main Functions

- rinex2snr
- quickLook
- gnssir code, inputs

4.2 Important Helper Functions

- extract_arcs code, discussion
- *tracks code*, discussion
- nmea2snr
- daily_avg code, discussion
- invsnr code, discussion, input
- refl_zones
- max_resolve_RH
- quickplt
- ymd, ydoy
- installexe Installs executables for pip clone/pip install users. This is for Linux and MacOS users only.

4.3 Environmental Products

- snowdepth code, discussion
- tides (subdaily) code, discussion
- vwc code, discussion, vwc_input, phasecode

4.4 Download Scripts

- download_orbits
- download_rinex
- download_tides
- download_unr

4.5 Various Utilities

- check_rinex_file
- gpsweek
- llh2xyz
- mjd
- query_unr
- rinex3_rinex2
- rinex3_snr
- rinex_coords
- smoosh
- smoosh_snr
- xyz2llh

CHAPTER

FIVE

EXAMPLE USE CASES

5.1 Ice Sheets

- Lorne, Antarctica
- Dye2, Greenland
- Thwaites Glacier, Antarctica
- Summit Camp, Greenland
- Phoenix, Antarctica

5.2 Lakes, Reservoirs, and Rivers

- Michipicoten, Canada
- Lake Taupo, New Zealand
- Steenbras, South Africa
- St Lawrence River,Canada
- Lake Mathews, Riverside, USA
- Lake Malawi, Tanzania
- Lake Yellowstone, USA
- Guaiba Lake, Brazil (low-cost)
- Wesel, Germany (low-cost)

5.3 Soil Moisture

- Portales, New Mexico USA
- Mitchell, Australia (Legacy GPS tracks)
- Mitchell, Australia (Multi-GNSS)
- Victorville, California USA
- Boulder, Colorado USA
- Stillwater, Oklahoma USA (Advanced Vegetation Model)

5.4 Seasonal Snow Accumulation

- Marshall, Colorado USA
- Niwot Ridge, Colorado USA
- Half Island Park, Idaho USA
- Utqiagvik, Alaska USA

5.5 Tides

- Friday Harbor, Washington USA
- St Michael, Alaska USA
- Vlissingen, the Netherlands
- Puerto Penasco, Mexico
- Elbe River, Germany

About 75% of these use cases use access to the Earthscope/UNAVCO archive. In some cases, sopac can be used as an alternate archive. If at all possible, you should sign up for an EarthScope account.

Some of these use cases were created with an earlier version of the gnssrefl software. The plots might look slightly different and the defaults we used in the analysis might have changed.

5.6 GPS Tool Box Demonstration

- MNIS (not finished)

5.7 GNSS Reflection Geometry and SNR Simulation:

- GNSSRefGeometry-SNRSimulation.ipynb: Explore the simulation of GNSS reflection geometry and SNR using python in Jupyter Notebook

5.8 Homeworks from Previous Shortcourses:

- Homework 0: Make sure you have properly installed the software
- Homework 1: Practice setting your azimuth and elevation angle mask
- Homework 2: Learn how to measure snow surface variations
  - Homework 2 Solution
- Homework 3: Learn how to measure water levels
  - Homework 3 Solution

CHAPTER

SIX

COMMUNITY INFORMATION

6.1 E-mail list

Would you like to join our gnssrefl users email list? This is currently maintained by earthscope.org. To join, please e-mail melissa.weber@earthscope.org or Kristine Larson.

6.2 Citation for gnssrefl

- Larson K.M., gnssrefl: an open source python software package for environmental GNSS interferometric reflectometry applications, GPS Solutions, Volume 28, article number 165, 2024

6.3 Acknowledgements

- Kristine M. Larson Overall
- Kelly Enloe Jupyter Notebooks
- Tim Dittmann Access to Dockers
- Radon Rosborough helped with python/packaging questions, improved our docker distribution, and set up smoke tests.
- Naoya Kadota added the GSI data archive and helped find a bug in nmea2snr.
- Joakim Strandberg provided python RINEX translators and the EGM96 code.
- Johannes Boehm provided source code for the refraction correction.
- Makan Karegar added the NMEA capability.
- Dave Purnell provided his SNR inversion code.
- Carolyn Roesler helped with the original GNSS-IR Matlab codes.
- Felipe Nievinski and Simon Williams have provided significant advice for this project.
- Clara Chew and Eric Small developed the soil moisture algorithm; I ported it to python with Kelly's help.
- Sree Ram Radha Krishnan ported the rzones web app code.
- Dan Nowacki added Glonass to the NMEA reader
- Taylor Smith has worked on the NMEA reader and the refl_zones utility.
- Surui Xie and Thomas Nylen were instrumental in finding a bug in the newarcs version

- Peng Feng, Rudiger Haas, and Gunnar Elgered have helped us improve refraction models.

6.4 2023 Short Course on GNSS-IR

- overview
- videos/lectures

6.5 2024 Short Course on GNSS-IR for Water Level Measurements

- index
- videos are on Kristine's youtube page.

6.6 How you can help improve this code

- Archives frequently change their file transfer protocols. If you find one in gnssrefl that doesn't work anymore, please fix it and let us know. Please test that it works for both older and newer data.
- If you would like to add an archive, please do so. Use the existing code as a starting point.
- Check the issues section of the repository and look for "help wanted."
- Write up a new use case
- Investigate surface related biases.

6.7 How to get help with your gnssrefl questions

If you are new to the software, you should consider watching the videos about GNSS-IR

Before you ask for help - you should check to see if you are running the current software. Please go to the install page for help on how to update your latest docker/jupyter installs. For github/pypi, we recommend doing a clean download and new install.

You are encouraged to submit your concerns as an **Issue** to the github repository.

Please include

- the exact command or section of code you were running that prompted your question.
- details such as the error message or behavior you are getting. Please copy and paste (this is preferred over a screenshot) the error string. If the string is long - please post the error string in a thread response to your question.
- the operating system of your computer.

We are not able to answer any questions about Jupyter Notebooks. These were developed by Earthscope and unfortunately they no longer help maintain them.

Old news section from before we moved to readthedocs

I have removed the publications section.

Kristine M. Larson

CHAPTER

SEVEN

WHAT IS A GOOD GNSS REFLECTIONS SITE?

A good GNSS reflection site has:

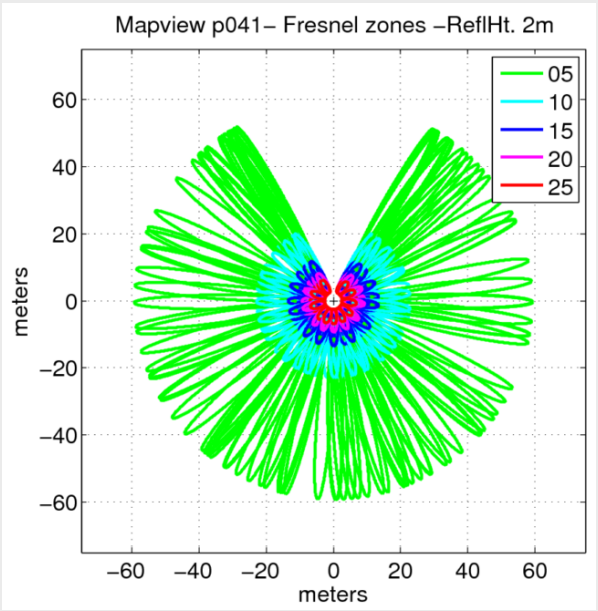
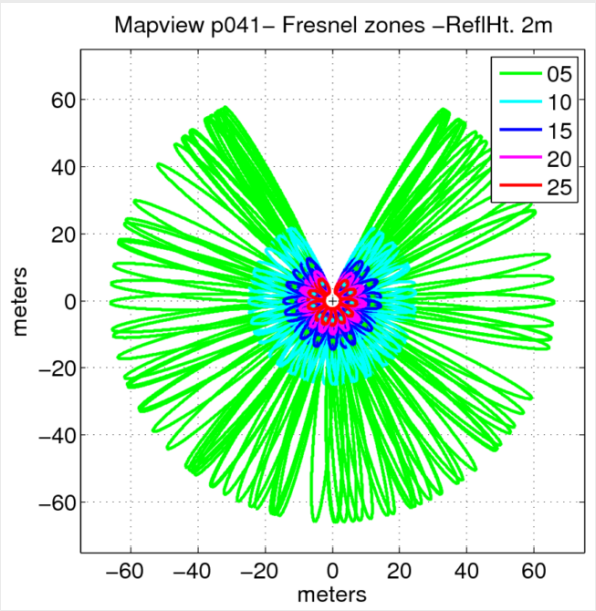
- A reflection zone that extends to a wide range of azimuths
- A good receiver tracking multiple GNSS signals and modern (L2C,L5) GPS signals
- A sampling rate that is commensurate with what you are trying to measure (i.e. 30 second sampling rate won't work for stations that are more than 8-9 meters above the reflecting surface). You should find out the proper sampling rate for your station before you install it! Use max_resolve_RH
- RINEX files with positions in the header and (preferably float) SNR data
- There is no elevation mask on the receiver

7.1 Reflection Zones

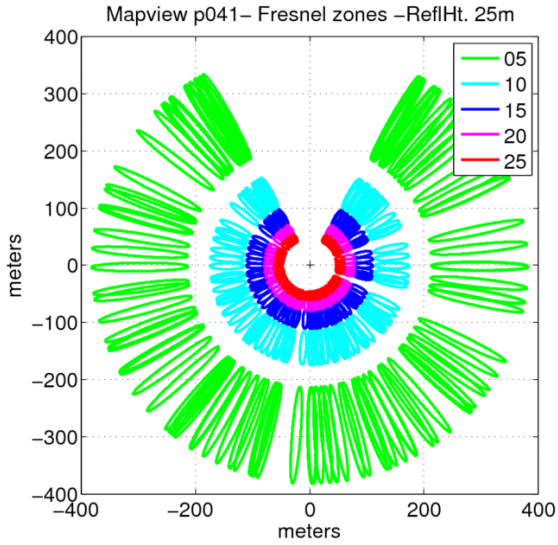
The only inputs needed to calculate your reflection zones are:

- the approximate position of the GNSS site
- the positions of the GNSS satellites
- the height of the antenna above the reflecting surface
- the GNSS signal wavelength (~0.19 or 0.244 meters for L1 vs. L2)

The equations you need for a Fresnel zone are given in the appendix for Larson and Nievinski (2013). Here are static examples for a 2 meter reflector height for L1 and L2.

L1	L2
	

Compare with a 25 meter reflector height:



25m reflector height Fresnel zone

Similarly, the sampling rate you need to use is not unknown – you just need to understand how the Nyquist frequency is defined for the SNR observations.

7.2 Designing a good GNSS Reflections Site:

- Sampling interval should be commensurate with your reflection target area. You can generally get away with 30 sec for surfaces that are < 10 meters below the antenna, but I urge you to use **15 sec**. For reflectors larger than 50 meters, I recommend 1 sec sampling. The bare minimum sampling rate numbers you need can be calculated using the code in Roesler and Larson (2018). This code can also be run from the GNSS-IR web app
- Make sure your antenna is surrounded by natural planar surfaces. No crashing waves. No outlet glaciers. No large ships coming and going.
- Use the reflection zone app or the python utility `refl_zones` to make sure that you can sense the surface you want to measure. This is extremely important for water levels, as many groups think seeing the water in a photo means you can measure it. All you need to check this is the position of your site. The app will calculate the geoid correction for the ellipsoidal height. If you are trying to measure an interior water body (where mean sea level is not relevant), there is a manual override.
- If you have flexibility, take into account that sites at mid-latitudes have holes in their sensing zone. In CONUS, don't face your GNSS receiver to the north. In southern Africa, South America, and Australia, don't try to use GNSS-IR to measure water levels to the south.
- If you are trying to measure snow accumulation in polar regions, you should ensure that your antenna is always at least 1 meters above the highest snow level. This may mean you need to revisit your site to reset the pole vertically.

7.3 Operating a good GNSS reflections site:

- Always remove the elevation mask on the receiver!
- Set the sampling interval by evaluating reflection surfaces. The standard GNSS sampling interval of thirty seconds was selected over thirty years ago before the internet existed! Collect (and archive) more data.
- Take photographs of your site.
- If you plan to put your GNSS antenna on a roof, pick the corner that gives you the best view of natural surfaces.
- Track all GPS signals! (L1 and L1C, L2P and L2C, L5). If you can track GLONASS, Galileo, Beidou without costing a lot of money, I strongly recommend it.
- It doesn't matter if you turn on multipath suppression algorithms or buy a fancy antenna. They don't stop multipath.
- Put SNR data in your RINEX file. RINEX 3 is generally preferred because it makes it easy to include all signals, but RINEX 2.11 is fine as long as you make sure the file has L2C and L5 in it.

7.4 Further reading

- Site guidelines for multi-purpose GNSS reflectometry stations

CHAPTER

EIGHT

API DOCUMENTATION

Information on specific functions, classes, and methods.

8.1 gnssrefl package

8.1.1 Submodules

gnssrefl.EGM96 module

class gnssrefl.EGM96.**EGM96geoid**

Bases: object

Class for EGM96 geoid corrections

Example

```
>>> egm = EGM06geoid()
>>> egm.heights(lat=10, lon=30)
-5.32
```

height(*lat: float*, *lon: float*)

gnssrefl.advanced_vegetation_correction module

Advanced vegetation correction model for VWC estimation.

Reference: DOI 10.1007/s10291-015-0462-4

Ported to gnssrefl from original MATLAB in October 2025.

gnssrefl.advanced_vegetation_correction.**advanced_vegetation_filter**(*station*, *vxyz*, *extension=''*, *bin_hours=24*, *bin_offset=0*, *pltit=True*, *fr=20*, *minvalperbin=10*, *save_tracks=False*)

Advanced vegetation model (model 2)

This function applies the advanced vegetation correction filter to a vxyz array input.

Parameters

- **station** (*str*) – 4-char GNSS station name

- **vxyz** (*numpy array*) – Full track-level data from vwc (16 columns)
- **extension** (*str*) – Extension used in the analysis json (default: '')
- **bin_hours** (*int*) – Time bin size for future subdaily support (default: 24)
- **bin_offset** (*int*) – Bin timing offset for future subdaily support (default: 0)
- **pltit** (*bool*) – Whether plots come to the screen
- **fr** (*int*) – Frequency code (1=L1, 5=L5, 20=L2C, default: 20)
- **minvalperbin** (*int*) – Minimum values required per time bin (default: 10)
- **save_tracks** (*bool*) – Save individual track VWC data to files (default: False)

Returns

Dictionary containing: - 'mjd': list of Modified Julian Day values - 'vwc': list of VWC values (percentage units, not leveled) - 'datetime': list of datetime objects for plotting - 'bin_starts': list of bin start hours (subdaily only) or empty list

Return type

dict

gnssrefl.advanced_vegetation_correction.**apply_vegetation_model**(*station*, *vxyz*, *normmet*, *sgolnum*, *sgolply*, *padlen*, *pltit*, *fr*, *bin_hours*, *bin_offset*, *extension*, *minvalperbin*, *save_tracks=False*)

Apply Clara's vegetation filter to compute soil moisture

Parameters

- **station** (*str*) – Station name
- **vxyz** (*numpy array*) – Original vwc data
- **normmet** (*numpy array*) – Normalized metrics from norm_zero_vxyz
- **sgolnum** (*int*) – Savgol filter length
- **sgolply** (*int*) – Savgol polynomial order
- **padlen** (*int*) – Padding length
- **pltit** (*bool*) – Show plots
- **fr** (*int*) – Frequency code
- **bin_hours** (*int*) – Time bin size in hours
- **bin_offset** (*int*) – Bin timing offset in hours
- **extension** (*str*) – Extension used in the analysis json
- **minvalperbin** (*int*) – Minimum values required per time bin
- **save_tracks** (*bool*) – Save individual track VWC data to files

Returns

- **final_mjd** (*list*) – MJD values for time-binned averages
- **final_vwc** (*list*) – VWC values for time-binned averages (percentage units, not leveled)
- **final_binstarts** (*list*) – Bin start hours for subdaily data (empty for daily)

gnssrefl.advanced_vegetation_correction.**load_clara_model**()

Load Clara's model from the organized model_data directory

Returns

- **amp_lsp** (*numpy array*) – LSP amplitude features
- **amp_dsnr** (*numpy array*) – DSNR amplitude features
- **delta_heff** (*numpy array*) – Change in effective reflector height
- **veg_correction** (*numpy array*) – Vegetation phase corrections
- **slope_correction** (*numpy array*) – Slope sensitivity corrections

gnssrefl.advanced_vegetation_correction.**norm_zero_vxyz**(*station*, *vxyz*, *remoutli*, *acc_rhdrift*, *baseperc*, *zphival*, *ampvarday*, *ampvarlimit*)

Normalize metrics and remove outliers from vxyz data

This processes the full track-level data from vwc to prepare it for the KNN lookup.

Parameters

- **station** (*str*) – Station name
- **vxyz** (*numpy array*) – Full track data (18 columns; see column list inside the function)
- **remoutli** (*int*) – Remove outliers flag
- **acc_rhdrift** (*int*) – Apply RH drift correction
- **baseperc** (*float*) – Percentage for amplitude normalization
- **zphival** (*float*) – Fraction for phase zeroing
- **ampvarday** (*int*) – Days for variance calculation
- **ampvarlimit** (*float*) – Amplitude variance threshold

Returns

- **metrics_all** (*numpy array*) – Normalized metrics [D_amplsp, D_amp, D_phi, D_RH]
- **vegmast** (*numpy array*) – Vegetation mask

gnssrefl.advanced_vegetation_correction.**padClara**(*obs*, *Ntrack*, *men*, *padlen*)

Pad arrays before smoothing (Clara's method)

Parameters

- **obs** (*numpy array*) – Observations to pad
- **Ntrack** (*int*) – Number of values in array
- **men** (*int*) – Number of values to use for calculating mean at each end
- **padlen** (*int*) – Padding length on each end

Returns
padded_obs – Padded version of obs

Return type
numpy array

gnssrefl.advanced_vegetation_correction.**rolling_window**(*a*, *window*)

Create rolling window view of array for variance calculation

From: https://stackoverflow.com/questions/6811183/rolling-window-for-1d-arrays-in-numpy

gnssrefl.check_rinex_file module

gnssrefl.check_rinex_file.**check_l2**(*i*, *savebase*, *nsat_line1*, *lines*, *nlin*, *l2index*, *year*, *doy*, *screenstats*)

Parameters

- **i** (*int*) – line number index in RINEX file
- **savebase** (*str*) – list of satellites at first epoch
- **nsat_line1** (*int*) – number of satellites in first epoch
- **nlin** (*int*) – number of lines per satellite allocated in RINEX file
- **l2index** (*int*) – index of L2 observable
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **screenstats** (*bool*) – print extra info to the screen

gnssrefl.check_rinex_file.**check_rinex_file**(*rinexfile*, *screenstats*)

commandline tool to look at header information in a RINEX file tries to look for existence of L2C data

Example

check_rinex_file p0311520.21o

Parameters

- **rinexfile** (*str*) – name of the RINEX 2.11 file
- **screenstats** (*bool*) – extra info sent to the screen

gnssrefl.check_rinex_file.**main**()

gnssrefl.computemp1mp2 module

gnssrefl.computemp1mp2.**ReadRecAnt**(*teqclog*)

prints out Receiver and Antenna name from a teqc log

Parameters
teqclog (*str*) – the name of a teqc log

gnssrefl.computemp1mp2.**check_directories**(*station*, *year*)

checks that directories exist for teqc logs

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – full year

gnssrefl.computemp1mp2.**get_files**(*station*, *year*, *doy*, *look*)

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

- **look** (*bool*) – whether you should try to get the file from unavco if it does not exist locally

Returns

- **navfile** (*str*) – navigation/orbit file
- **rinexfile** (*str*) – name of the obs file
- **foutname** (*str*) – full name of the teqc log output
- **mpdir** (*str*) – directory for MP results
- **goahead** (*boolean*) – whether you should go ahead and run teqc

gnssrefl.computemp1mp2.**main**()

computes MP1 MP2 stats using teqc or reads existing log

gnssrefl.computemp1mp2.**readoutmp**(*teqcfile*, *rcvtype*)

teqcfile input is the full name of a teqc log rcvtype is a string that includes the name of the receiver you are searching for. it does not have to be exact (so NETR would work for NETRS) returns MP1, MP2 (both in meters), and a boolean as to whether the values were found if rcvtype is set to NONE, it will return data without restriction

gnssrefl.computemp1mp2.**run_teqc**(*teqc*, *navfile*, *rinexfile*, *foutname*, *mpdir*)

run teqcs and stores the output

Parameters

- **teqc** (*str*) – location of the teqc executable
- **navfile** (*str*) – name of the RINEX nav file
- **rinexfile** (*string*) – name of the RINEX observation file
- **foutname** (*str*) – name of the output file
- **mpdir** (*str*) – location of the multipath directory on your system

gnssrefl.computemp1mp2.**sfilename**(*station*, *year*, *doy*)

Finds mp1 filename on your system

Parameters

- **station** (*string*) – 4 character station name
- **year** (*integer*) –
- **doy** (*integer*) – day of year

Returns
xfile – the full SNR filename on your local system

Return type
string

gnssrefl.computemp1mp2.**vegplt**(*station*, *tv*, *winter*)

makes a plot of MP1 multipath metric. Sends to the screen

Parameters

- **station** (*str*) – 4 ch station name
- **tv** (*np array*) – (year, doy, mp1, mp2)
- **winter** (*bool*) – whether to throw out ~jan-apr and ~oct-dec

gnssrefl.daily_avg module

gnssrefl.daily_avg.**daily_avg_stat_plots**(*obstimes*, *meanRH*, *meanAmp*, *station*, *txtdir*, *tv*, *ngps*, *nglo*, *ngal*, *nbei*, *test*)

plots of results for the daily avg code

daily average amplitude is now done elsewhere

Parameters

- **obstimes** (*datetime object*) –
- **meanRH** (*numpy array*) – daily averaged Reflector Height values in meters
- **meanAmp** (*numpy array*) – daily average RH amplitude
- **station** (*str*) – 4 character station name
- **txtdir** (*str*) – directory for the results
- **tv** – is the variable of daily results
- **ngps** (*numpy array*) – number of gps satellites each day
- **nglo** (*numpy array*) – number of glonass satellites each day
- **ngal** (*numpy array*) – number of galileo satellites each day
- **nbei** (*numpy array*) – number of beidou satellites each day
- **test** (*bool*) –

gnssrefl.daily_avg.**fbias_daily_avg**(*station*)

reads QC-RH values and the daily averages computes residuals and estimate the frequency bias for all available frequencies which is printed to the screen

Parameters
station (*str*) – station name - 4char - lowercase

gnssrefl.daily_avg.**multi_freq_amp**(*station*, *inputf*, *pngname*)

creates a time series plot of daily average LSP amplitude by frequency.

Parameters

- **station** (*str*) – 4 char staiton name
- **inputf** (*str*) – location of plain text file to be plotted
- **pngname** (*str*) – location of png file created

gnssrefl.daily_avg.**quick_raw**(*alldatafile2*, *xdir*, *station*, *subdir*)

quick plot of the raw RH data. No QC

Parameters

- **alldatafile2** (*str*) – name of the raw file to be read
- **xdir** (*str*) – code environ variable (I think)
- **station** (*str*) – 4 ch station name
- **subdir** (*str*) – subdirectory name for results in xdir/Files

gnssrefl.daily_avg.**readin_plot_daily**(*station*, *extension*, *year1*, *year2*, *fr*, *alldatafile*, *csvformat*, *howBig*, *ReqTracks*, *azim1*, *azim2*, *test*, *subdir*, *plot_limits*, ***kwargs*)

worker code for daily_avg_cl.py

It reads in RH files created by gnssir. Applies median filter and saves average results for further analysis

if there is only one RH on a given day - there is no median value and thus nothing will be saved for that day.

Gosh it would be nice if someone would clean this up. It was written before I understood python.

Parameters

- **station** (*str*) – station name, 4 ch, lowercase
- **extension** (*str*) – folder extension - usually empty string
- **year1** (*int*) – first year
- **year2** (*int*) – last year
- **fr** (*integer*) – 0 for all frequencies. otherwise, it must be a legal frequency (101 for Glonass L1)
- **alldatafile** (*str*) – name of the output filename
- **csvformat** (*boolean*) – whether you want output as csv format
- **howBig** (*float*) – criterion for the median filter, i.e. how far in meters can a RH be from the median for that day, in meters
- **ReqTracks** (*integer*) – is the number of retrievals required per day
- **azim1** (*integer*) – minimum azimuth, degrees
- **azim2** (*integer*) – maximum azimuth, degrees
- **test** (*bool*) –
- **subdir** (*bool*) – subdirectory for output files
- **subdir** – whether plot limits for the median filter are shown

Returns

- **tv** (*numpy array*) – with these values [year, doy, meanRHtoday, len(rh), month, day, stdRH, averageAmplitude] len(rh) is the number of RH on a given day stdRH is the standard deviation of the RH values (meters) averageAmplitude is in volts/volts
- **obstimes** (*list of datetime objects*) – observation times

gnssrefl.daily_avg.**write_out_RH_file**(*obstimes*, *tv*, *outfile*, *csvformat*, *station*, *extension*)

write out a file with the daily average RH values in it

Parameters

- **obstimes** (*datetime object*) – time of observation
- **tv** (*numpy array*) – these values [year, doy, meanRHtoday, len(rh), month, day, stdRH, averageAmplitude] where meanRHtoday is the mean (with QC), averageAmplitude is the average of the LSP amplitudes. stdRH is probably the standard deviation of RH
- **outfile** (*string*) – full name of output file
- **csvformat** (*boolean*) – true if you want csv format output
- **station** (*str*) – 4 ch station name
- **extension** (*str, optional*) – analysis extension name

gnssrefl.daily_avg.**write_out_all**(*allrh*, *csvformat*, *NG*, *yr*, *doy*, *d*, *good*, *gazim*, *gfreq*, *gsat*, *gamp*, *gpeak2noise*, *gutcTime*, *tvall*)

writing out all the RH retrievals to a single file: file ID is allrh) tvall had everything in it, but it was slowing everything down, so i do not do anything with it.

Parameters

- **allrh** (*fileID for writing*) –
- **csvformat** (*bool*) – whether you are writing to csv file
- **NG** (*int*) – number of lines of results
- **yr** (*int*) – year
- **doy** (*int*) – day of year
- **d** (*datetime object*) –
- **good** (*float*) – reflector height - I think
- **gazim** (*numpy array of floats*) – azimuths
- **gfreq** (*numpy array of int*) – frequencies
- **gsat** (*numpy array of int*) – satellite numbers
- **gamp** (*numpy array of floats*) – amplitudes of periodograms
- **gpeak2noise** (*numpy array of floats*) – peak 2 noise for periodograms
- **gutcTime** (*numpy array of floats*) – time of day in hours
- **tvall** –

Returns

tvall

Return type

??

gnssrefl.daily_avg_cl module

gnssrefl.daily_avg_cl.**daily_avg**(*station: str*, *medfilter: float*, *ReqTracks: int*, *txtfile: str | None = None*, *plt: bool = True*, *extension: str = ''*, *year1: int | None = None*, *year2: int | None = None*, *fr: int = 0*, *csv: bool = False*, *azim1: int = 0*, *azim2: int = 360*, *test: bool = False*, *subdir: str | None = None*, *plot_limits: bool = False*, *date1: str | None = None*, *date2: str | None = None*)

The goal of this code is to consolidate individual RH results into a single file consisting of daily averaged RH without outliers. These daily average values are nominally associated with the time of 12 hours UTC.

There are two required parameters - medfilter and ReqTracks. These are quality control parameters. They are applied in two steps. The code first calculates the median value each day - and keeps only the RH that are within medfilter (meters) of this median value. If there are at least "ReqTracks" number of RH left after that step, a daily average is computed for that day.

As of version 3.1.3 users may store the required input parameters to daily_avg in the json used by gnssir. The names of these parameters are: daily_avg_reqtracks and daily_avg_medfilter. For those making a new json, the parameters will be set to None if you don't choose a value on the command line. You can also hand edit or add it. This would be helpful in not having to rerun gnssir_input and risk losing some of your other specialized selections. Because median filter and required tracks are REQUIRED inputs, you still have to tell the code

something, even if you overwrite it in the json. Set to 0 and 0 to trigger the json inputs. I will work on getting a better way to do this.

If you are unfamiliar with what a median filter does in this code, please see https://gnssrefl.readthedocs.io/en/latest/pages/README_dailyavg.html

The outputs are stored in $REFL_CODE/Files/station by default. If you want to specify a new subdirectory, I believe that is an allowed option. You can also specify specific years to analyze and apply fairly simple azimuth constraints.

In summary, three text files are created

1. individual RH values with no QC applied
2. individual RH values with QC applied
3. daily average RH

Examples

daily_avg p041 0.25 10
consolidates results for p041 with median filter of 0.25 meters and at least 10 solutions per day

daily_avg p041 0.25 10 -plot_limits T
the same as above but with plot_limits to help you see where the median filter is applied

daily_avg p041 0.25 10 -year1 2015 -year2 2020
consolidates results for p041 with median filter of 0.25 meters and at least 10 solutions per day and restricts it to years between 2015 and 2020

daily_avg p041 0.25 10 -year1 2015 -year2 2020 -azim1 0 -azim2 180
consolidates results for p041 with median filter of 0.25 meters and at least 10 solutions per day and restricts it to years between 2015 and 2020 and azimuths between 0 and 180 degrees

daily_avg p041 0.25 10 -extension NV
consolidates results which were created using the extension NV when you ran gnssir.

daily_avg p041 0 0
this will use median filter and required tracks values from within the json. The parameter names are slightly different, daily_avg_medianfilter and dailyavg_reqtracks.

daily_avg p041 0.5 50 -date1 20200501 -date2 20210601
only uses data beteween May 1, 2020 and June 1, 2021

Parameters

- **station** (*str*) – 4 ch station name, generally lowercase
- **medfilter** (*float*) – Median filter for daily reflector height (m). Start with 0.25 for surfaces where you expect no significant subdaily change (snow/lakes).
- **ReqTracks** (*int*) – Required number of daily satellite tracks to save the daily average value.
- **txtfile** (*str, optional*) – Use this parameter to set your own output filename. default is to let the code choose.
- **plt** (*bool, optional*) – whether to print plots to screen or not. default is True.
- **extension** (*str, optional*) – extension for solution names. default is ‘’. (empty string)
- **year1** (*int, optional*) – restrict to years starting with. default is 2005 (set below).
- **year2** (*int, optional*) – restrict to years ending with. default is 2035 (set below).

- **fr** (*int, optional*) – GNSS frequency. If none input, all are used. Value options:

  1 : GPS L1

  2 : GPS L2

  20 : GPS L2C

  5 : GPS L5

  101 : GLONASS L1

  102 : GLONASS L2

  201 : GALILEO E1

  205 : GALILEO E5a

  206 : GALILEO E6

  207 : GALILEO E5b

  208 : GALILEO E5

  302 : BEIDOU B1

  306 : BEIDOU B3

  307 : BEIDOU B2

- **csv** (*boolean, optional*) – Whether you want csv instead of a plain text file. default is False.
- **azim1** (*int, optional*) – minimum azimuth, degrees note: should be modified to allow negative azimuth
- **azim2** (*int, optional*) – maximum azimuth, degrees
- **test** (*bool, optional*) – not sure what this does anymore.
- **subdir** (*str, optional*) – non-default subdirectory for Files output
- **plot_limits** (*bool, optional*) – adds the median value and median filter limits to the plot. default is False
- **date1** (*str, optional*) – you only want data starting from this date, format yyyymmdd this will supercede year1
- **date2** (*str, optional*) – you only want data ending from this date, format yyyymmdd this will supercede year2

gnssrefl.daily_avg_cl.**main**()

gnssrefl.daily_avg_cl.**parse_arguments**()

gnssrefl.download_ioc module

gnssrefl.download_ioc.**download_ioc**(*station: str*, *date1: str*, *date2: str*, *output: str | None = None*, *plt: bool = False*, *outliers: bool = False*, *sensor=None*, *subdir: str | None = None*)

Downloads and saves IOC tide gauge files

Parameters

- **station** (*str*) – IOC station name
- **date1** (*str*) – begin date in yyyymmdd. Example value: 20150101
- **date2** (*str*) – end date in yyyymmdd. Example value: 20150101
- **output** (*str*) – Optional output filename default is None The file will be written to REFL_CODE/Files
- **plt** (*bool, optional*) – plot comes to the screen default is None
- **outliers** (*bool, optional*) – tried to remove outliers, but it doesn't work as yet default is No
- **sensor** (*str, optional*) – type of sensor, prs(for pressure), rad (for radar), flt (for float) default is None, which means it will print out what is there. if there is more than one sensor you should specifically ask for the one you want

gnssrefl.download_ioc.**find_start_stop**(*year*, *m*)

finds the start and stop times for each month of the IOC download

Parameters

- **year** (*int*) – full year
- **m** (*int*) – month number

Returns

- **d1** (*str*) – yyyymmdd for first day of requested month
- **d2** (*str*) – yyyymmdd for last day of requested month

gnssrefl.download_noaa module

gnssrefl.download_noaa.**download_noaa**(*station: str*, *date1: str*, *date2: str*, *output: str | None = None*, *plt: bool = False*, *datum: str = 'mllw'*, *subdir: str | None = None*)

Downloads NOAA tide gauge files and stores locally If you ask for 31 days of data or less, it will download exactly what you ask for. But if you want a longer time series, this code needs to query the NOAA API every month. To make the code easier to write, I start with the first day of the first month you ask for and end with last day in the last month.

Output is written to REFL_CODE/Files/ unless subdir optional input is set Plot is sent to the screen if requested.

Parameters

- **station** (*str*) – 7 character ID of the station.
- **date1** (*str*) – start date. Example value: 20150101
- **date2** (*str*) – end date. Example value: 20150110
- **output** (*string, optional*) – Optional output filename default is None

- **plt** (*boolean, optional*) – plot comes to the screen default is None
- **datum** (*string, optional*) – set to lwd for lakes? default is mllw
- **subdir** (*str, optional*) – subdirectory for output in the $REFL_CODE/Files area

gnssrefl.download_noaa.**download_qld**(*station*, *year*, *plt*)

Parameters

- **station** (*str*) – tide gauge station name
- **year** (*int*) – calendar year
- **plt** (*bool*) – whether you want a plot to the screen

gnssrefl.download_noaa.**multimonthdownload**(*station*, *datum*, *fout*, *year1*, *year2*, *month1*, *month2*, *csv*)

downloads NOAA water level measurements > one month

Parameters

- **station** (*str*) – NOAA station name
- **datum** (*str*) – definition of water level datum
- **results** (*fout - fileID for writing*) –
- **year1** (*int*) – year when first measurements will be downloaded
- **month1** (*integer*) – month when first measurements will be downloaded
- **year2** (*integer*) – last year when measurements will be downloaded
- **month2** (*integer*) – last month when measurements will be downloaded
- **csv** (*boolean*) – whether output file is csv format

Returns

- **tt** (*list of times*) – modified julian day
- **obstimes** (*list of datetime objects*)
- **slevel** (*list or is it numpy ??*) – water level in meters

gnssrefl.download_noaa.**noaa2me**(*date1*)

converts NOAA type of date string to simple integers

Parameters

date1 (*string*) – time in format YYYYMMDD for year month and day

Returns

- **year1** (*integer*) – full year
- **month1** (*integer*) – month
- **day1** (*integer*) – day of the month
- **doy** (*integer*) – day of year
- **modjulday** (*float*) – modified julian date

gnssrefl.download_noaa.**noaa_command**(*station*, *fout*, *year*, *month1*, *month2*, *datum*, *metadata*, *tt*, *obstimes*, *slevel*, *csv*)

downloads/writes NOAA tidegauge data for one month

Parameters

- **station** (*str*) – station name
- **year** (*int*) – full year
- **month1** (*int*) – starting month
- **month2** (*int*) – ending month
- **datum** (*str*) – water datum
- **metadata** (*bool*) – whether you want the metadata printed to the screen
- **tt** (*numpy array*) – modified julian date for water measurements
- **obstimes** (*numpy array of datetimes*) – time of the measurements
- **slevel** (*numpy array of floats*) – water level in meters
- **csv** (*bool*) – True if csv output wanted (default is False)

Returns

- **tt** (*numpy array*) – modified julian date for water measurements
- **obstimes** (*numpy array*) – datetime format, updated with new data
- **slevel** (*numpy array*) – sea level (m) updated with new data

gnssrefl.download_noaa.**pickup_from_noaa**(*station*, *date1*, *date2*, *datum*, *printmeta*)

pickup up NOAA data between date1 and date2, which can be longer than one month (NOAA API restriction)

Parameters

- **station** (*str*) – station name
- **date1** (*str*) – beginning time, 20120101 is January 1, 2012
- **date2** (*str*) – end time , same format
- **datum** (*str*) – what kind of datum is requested
- **printmeta** (*bool*) – print metadata to screen

Returns

- **data** (*dictionary in NOAA format*)
- **error** (*bool*)

gnssrefl.download_noaa.**write_out_data**(*data*, *fout*, *tt*, *obstimes*, *slevel*, *csv*)

writes out the NOAA water level data to a file 20213-mar-27 using new format

Parameters

- **data** (*dictionary from NOAA API*) –
- **fout** (*file ID*) – for output
- **tt** –
- **obstimes** (*list of datetimes*) – times of water level measurements
- **slevel** (*numpy array of floats*) – water level in meters
- **csv** (*boolean*) – whether csv format or not

Returns

- **tt** (*same as input, but larger*)

- **obstimes** (*list of datetimes*) – times for waterlevels
- **slevel** (*list of floats*) – water levels in meters

gnssrefl.download_orbits module

gnssrefl.download_orbits.**download_orbits**(*orbit: str*, *year: int*, *month: int*, *day: int*, *doy_end: int | None = None*, *hour: int = 0*)

command line interface for download_orbits. If day is zero, then it is assumed that the month record is day or year

Examples

download_orbits nav 2020 50 0
downloads broadcast orbits for day of year 50 in the year 2020

download_orbits nav 2020 1 1
downloads broadcast orbits for January 1, 2020

download_orbits gnss 2023 1 1
multi-GNSS orbits from GFZ

download_orbits rapid 2023 1 1
rapid multi-GNSS orbits from GFZ

download_orbits rapid 2023 1 0 -doy_end 10
rapid multi-GNSS orbits from GFZ for days of year 1 thru 10 in 2023

Parameters

- **orbit** (*string*) – value options:

  gps (default) : uses GPS broadcast orbit

  gps+glo : will use JAXA orbits which have GPS and Glonass (usually available in 48 hours)

  gnss : will use GFZ orbits, which is multi-GNSS (available in 3-4 days). but taken from CDDIS archive

  nav : GPS broadcast, adequate for reflectometry. Searches various places

  nav-sopac : GPS broadcast file from SOPAC, adequate for reflectometry.

  nav-esa : GPS broadcast file from ESA, adequate for reflectometry.

  nav-cddis : GPS broadcast file from CDDIS, very slow to download

  igs : IGS precise, GPS only

  igr : IGS rapid, GPS only

  jax : JAXA, GPS + Glonass, within a few days, missing block III GPS satellites

  gbm : GFZ Potsdam, multi-GNSS, not rapid

  grg : French group, GPS, Galileo and Glonass, not rapid

  esa : ESA, multi-GNSS

  gfr : GFZ rapid, GPS, Galileo and Glonass, since May 17 2021

  wum : (disabled) Wuhan, multi-GNSS, not rapid

gnss2 : multi-GNSS, but uses IGN instead of CDDIS. does not work

gnss3 : multi-GNSS, but uses GFZ archive instead of CDDIS. same as gnss-gfz

ultra : ultra orbits directly from GFZ

rapid : rapid orbits directly from GFZ

- **year** (*integer*) – full year
- **month** (*integer*) – calendar month
- **day** (*integer*) – day of the month
- **doy_end** (*integer*) – optional, allows multiple day download
- **hour** (*int*) – optional hour for ultrarapid orbit , default is zero

gnssrefl.download_orbits.**main**()

gnssrefl.download_orbits.**parse_arguments**()

gnssrefl.download_psmsl module

gnssrefl.download_psmsl.**download_psmsl**(*station: str*, *output: str | None = None*, *plt: bool = False*)

Downloads PSMSL tide gauge files created by Simon Williams in json format, converts it to plain txt or csv format

Parameters

- **station** (*str*) – 4 ch station name
- **output** (*str, optional*) – Optional output filename default is None
- **plt** (*bool, optional*) – plot comes to the screen

gnssrefl.download_rinex module

downloads RINEX files

gnssrefl.download_rinex.**download_rinex**(*station: str*, *year: int*, *month: int*, *day: int*, *rate: str = 'low'*, *archive: str = 'all'*, *version: int = 2*, *strip: bool = False*, *doy_end: int | None = None*, *stream: str = 'R'*, *samplerate: int = 30*, *screenstats: bool = False*, *dec: int = 1*, *save_crx: bool = False*, *delete_hourly: bool = True*)

Command line interface for downloading RINEX files from global archives. Required inputs are station, year, month, and day. If you want to use day of year, call it as station, year, doy, 0.

decimate does not seem to do anything, at least not for RINEX 2.11 files

bkg option is changed. now must specify bkg-igs or bkg-euref

Examples

download_rinex mfle 2015 1 1
downloads January 1, 2015

download_rinex mfle 2015 52 0
Using day of year instead of month/day:

download_rinex p101 2015 52 0 -archive sopac
checks only sopac archive

Parameters

- **station** (*str*) – 4 or 9 character ID of the station.
- **year** (*int*) – full Year
- **month** (*int*) – month
- **day** (*int*) – day of month
- **rate** (*str, optional*) – sample rate. value options:

  low (default) : standard rate data

  high : high rate data

- **archive** (*str, optional*) – Select which archive to get the files from. Default is redirected to all, as defined below. Value options:

  cddis : NASA

  bev : Austria Federal Office of Metrology and Surveying

  bkg-igs : igs folder of BKG German Agency for Cartography and Geodesy

  bkg-euref : Euref folder of BKG German Agency for Cartography and Geodesy

  bfg : German Agency for water research, only Rinex 3 (no longer works)

  ga : Geoscience Australia

  gnet : Greenland Network

  gfz : GFZ

  jp : Japan-GSI

  jeff : Jeff Freymueller

  nrcan : Natural Resources Canada

  ngs : National Geodetic Survey

  nz : GNS, New Zealand

  sonel : (?)

  sopac : Scripps Orbit and Permanent Array Center

  special : reflectometry Rinex 2.11 files maintained by unavco

  unavco : now earthscope

  ngs-hourly: NGS, hourly files will be merged if they exist

  all : unavco, sopac, and sonel in that order

- **version** (*int, optional*) – Version of Rinex file. Default is 2. Value options 2 or 3

- **strip** (*bool, optional*) – Whether to strip only SNR observables. Uses teqc or gfzrnx. Default is False.
- **doy_end** (*int, optional*) – End day of year to be downloaded. Default is None. (meaning only a single day using the doy parameter)
- **stream** (*str, optional*) – Receiver or stream file, for RINEX3 only Default is 'R' but you can set to 'S' to get streamed version
- **samplerate** (*int, optional*) – Sample rate in seconds for RINEX3 only. Default is 30.
- **screenstats** (*bool, optional*) – provides screen output helpful for debugging Default is False
- **dec** (*int, optional*) – some highrate file downloads allow decimation. Default is 1 sec, i.e. no decimation
- **save_crx** (*bool, option*) – saves crx version for Rinex3 downloads. Otherwise they are deleted.

gnssrefl.download_rinex.**main**()

gnssrefl.download_rinex.**parse_arguments**()

gnssrefl.download_teqc module

download a year of teqc logs from unavco can do multiple years as well 2022 september 15, updated to https access

gnssrefl.download_teqc.**download_teqc**(*station: str*, *year: int*, *year_end: int | None = None*)

Download teqc logs from UNAVCO for one (or more) year.

Parameters

- **station** (*string*) – 4 character ID of the station
- **year** (*integer*) – Year
- **year_end** (*int, optional*) – end year.

gnssrefl.download_teqc.**main**()

gnssrefl.download_teqc.**mpfile_unavco**(*station*, *year*, *doy*)

picks up teqc log from unavco if it exists stores it in $REFL_CODE / year / mp / station directory does not check that directory exists. Assumes you previously ran check_directories from the veg library

Parameters

- **station** (*string*) – four character station name
- **year** (*integer*) –
- **doy** (*integer*) – day of year

gnssrefl.download_teqc.**parse_arguments**()

gnssrefl.download_tides module

gnssrefl.download_tides.**download_tides**(*station: str*, *network: str*, *date1: str | None = None*, *date2: str | None = None*, *output: str | None = None*, *plt: bool = False*, *datum: str = 'mllw'*, *subdir: str | None = None*, *year: int | None = None*)

Downloads tide gauge data from four different networks (see below)

Output is written to REFL_CODE/Files/ unless subdir optional input is set. Plot is sent to the screen if requested.

Examples

download_tides 8768094 noaa 20210101 20210131
NOAA station 876094

download_tides thul ioc 20210101 20210131
IOC station thul

download_tides 5970026 wsv
WSV station 5970026

download_tides 10313 psmsl
PSMSL station 10313 (downloads one file)

Parameters

- **station** (*str*) – station name
- **network** (*str*) – name of tide network. Options:

  noaa : US NOAA

  ioc : UNESCO

  wsv : Germany, Wasserstrassen-und Schifffahrtsverwaltung

  psmsl : Permanent Service Mean Sea Level
- **date1** (*str, optional*) – start date, 20150101, needed for NOAA/IOC
- **date2** (*str,optional*) – end date, 20150110, needed for NOAA/IOC
- **output** (*str, optional*) – Optional output filename
- **plt** (*bool, optional*) – plot comes to the screen
- **datum** (*str, optional*) – NOAA input, default is mllw
- **sensor** (*str, optional*) – setting for IOC
- **subdir** (*str, optional*) – subdirectory for output in the $REFL_CODE/Files area

gnssrefl.download_tides.**main**()

gnssrefl.download_tides.**parse_arguments**()

gnssrefl.download_unr module

gnssrefl.download_unr.**download_unr**(*station: str*)

Command line interface for downloading time series from the University of Nevada Reno website

This code is not actively maintained.

Examples

download_unr p041

download_unr sc02

Parameters
station (*str*) – 4 character ID of the station name

gnssrefl.download_unr.**main**()

gnssrefl.download_unr.**parse_arguments**()

gnssrefl.download_wsv module

gnssrefl.download_wsv.**download_wsv**(*station: str*, *plt: bool = True*, *output: str | None = None*)

Downloads and saves WSV (Germany) tide gauge files

Parameters

- **station** (*str*) – station name
- **plt** (*bool, optional*) – plot comes to the screen default is None
- **output** (*str, optional*) – output filename which is stored in $REFL_CODE/Files if not set, it uses station.txt

gnssrefl.download_wsv.**main**()

gnssrefl.download_wsv.**parse_arguments**()

gnssrefl.extract_arcs module

extract_arcs.py - Standalone module for extracting satellite arcs from SNR data.

This module provides a clean API for detecting and extracting satellite arcs from Signal-to-Noise Ratio (SNR) data files. It refactors arc detection logic from gnssir_v2.py into reusable functions.

An "arc" represents a continuous satellite pass (rising or setting) across the sky. Arcs are split when: 1. Time gap > 600 seconds (10 minutes) 2. Elevation angle direction reverses (rising <-> setting)

gnssrefl.extract_arcs.**apply_refraction**(*snr_array*, *station_config*, *year*, *doy*, *verbose=True*)

Apply refraction correction to SNR elevation angles.

Returns a copy of *snr_array* with corrected elevations; rows where the correction is invalid (e.g. ele < 1.5 for NITE/MPF) are removed.

gnssrefl.extract_arcs.**attach_gnssir_processing_results**(*arcs*, *results*, *time_tolerance=0.17*)

Attach gnssir processing results to extracted arcs.

For each arc, finds the matching row in the gnssir result file based on satellite number, frequency, rise/set direction, and UTC time proximity. Sets `metadata['gnssir_processing_results']` to a dict or `None`.

gnssrefl.extract_arcs.**attach_phase_processing_results**(*arcs*, *results*, *time_tolerance=0.17*)

Attach phase processing results to extracted arcs.

For each arc, finds the matching row in the phase result file based on satellite number, frequency, and UTC time proximity. Sets `metadata['phase_processing_results']` to a dict or `None`.

gnssrefl.extract_arcs.**attach_vwc_track_results**(*arcs*, *station*, *year*, *doy*, *extension=''*, *az_tolerance=5.0*, *time_tolerance=0.25*)

Attach VWC track results to extracted arcs.

Matches track file rows to arcs by sat, azimuth, and hour. Track files live under vwc_outputs/<freq>/individual_tracks/, so the freq lookup is resolved by the per-arc metadata['freq'].

Sets `metadata['vwc_track_results']` to a dict or `None`.

gnssrefl.extract_arcs.**check_azimuth_compliance**(*az_min_ele: float*, *azlist: List[float]*) → bool

Check if azimuth is within allowed regions.

Parameters

- **az_min_ele** (*float*) – Azimuth angle (degrees) at the lowest elevation point of the arc
- **azlist** (*list of float*) – Azimuth regions as pairs [az1_start, az1_end, az2_start, az2_end, ...] e.g., [0, 90, 180, 270] means 0-90 and 180-270 degrees

Returns
True if azimuth is within any of the allowed regions

Return type
bool

gnssrefl.extract_arcs.**extract_arcs**(*snr_array: ndarray*, *freq: int | List[int] | None = None*, *e1: float = 5.0*, *e2: float = 25.0*, *ellist: List[float] | None = None*, *azlist: List[float] | None = None*, *sat_list: List[int] | None = None*, *min_pts: int = 20*, *polyV: int = 4*, *pele: List[float] | None = None*, *dbhz: bool = False*, *screenstats: bool = False*, *detrend: bool = True*, *split_arcs: bool = True*, *filter_to_day: bool = True*, *year: int | None = None*, *doy: int | None = None*, *dec: int = 1*) → List[Tuple[Dict[str, Any], Dict[str, ndarray]]]

Extract satellite arcs from SNR data array.

Parameters

- **snr_array** (*np.ndarray*) – 2D array with columns: [sat, ele, azi, seconds, edot, snr1, snr2, ...]
- **freq** (*int, list of int, or None*) – Frequency code(s). Default: None (auto-detect)
- **e1** (*float*) – Minimum elevation angle (degrees). Default: 5.0
- **e2** (*float*) – Maximum elevation angle (degrees). Default: 25.0
- **ellist** (*list of floats, optional*) – Multiple elevation angle ranges as pairs. Overrides e1/e2.
- **azlist** (*list of floats, optional*) – Azimuth regions as pairs. Default: [0, 360]

- **sat_list** (*list of int, optional*) – Specific satellites to process. Default: all satellites in data
- **min_pts** (*int*) – Minimum points required per arc. Default: 20
- **polyV** (*int*) – Polynomial order for DC removal. Default: 4
- **pele** (*list of float, optional*) – Elevation angle range [min, max] for polynomial fit. Default: [e1, e2]
- **dbhz** (*bool*) – If True, keep SNR in dB-Hz. Default: False
- **screenstats** (*bool*) – If True, print debug information. Default: False
- **detrend** (*bool*) – If True (default), remove DC component via polynomial fit.
- **split_arcs** (*bool*) – If True (default), split data into separate arcs.
- **filter_to_day** (*bool*) – If True (default), only return arcs within the principal day (0-24h).
- **year** (*int, optional*) – Year, used for L2C/L5 satellite list lookup.
- **doy** (*int, optional*) – Day of year, used with *year*.
- **dec** (*int*) – Decimation factor. Default: 1 (no decimation).

Returns

Each arc is represented as: - metadata: dict with keys: sat, freq, arc_num, arc_type, ele_start, ele_end,

az_min_ele, az_avg, time_start, time_end, arc_timestamp, num_pts, delT, edot_factor, cf

- data: dict with keys: ele, azi, snr, seconds, edot (all np.ndarray)

Return type
list of (metadata, data) tuples

gnssrefl.extract_arcs.**extract_arcs_from_file**(*obsfile: str*, *freq: int | List[int] | None = None*, *buffer_hours: float = 2*, ***kwargs*) → List[Tuple[Dict[str, Any], Dict[str, ndarray]]]

Extract satellite arcs from an SNR file.

Loads the file with `read_snr()` and extracts arcs in one call.

Parameters

- **obsfile** (*str*) – Path to the SNR observation file.
- **freq** (*int, list of int, or None*) – Frequency code(s). Default: None (auto-detect)
- **buffer_hours** (*float*) – Hours of data from adjacent days. Default: 2
- ****kwargs** – Additional keyword arguments passed to `extract_arcs()`

Returns
See `extract_arcs()` for format details.

Return type
list of (metadata, data) tuples

Raises

- **FileNotFoundError** – If *obsfile* does not exist.
- **RuntimeError** – If `read_snr()` fails to load the file.

gnssrefl.extract_arcs.**extract_arcs_from_station**(*station: str*, *year: int*, *doy: int*, *freq: int | List[int] | None = None*, *snr_type: int = 66*, *buffer_hours: float = 2*, *attach_results: bool | List[str] = False*, *extension: str = ''*, *station_config: Dict[str, Any] | None = None*, *gzip: bool = True*, *track_file: str | Path | None = None*, *track_cache: Dict[str, Any] | None = None*, *tag_with_legacy_apriori: bool = False*, *refraction_verbose: bool = True*, ***kwargs*) → List[Tuple[Dict[str, Any], Dict[str, ndarray]]]

Extract satellite arcs for a station/year/day.

Resolves the SNR file path, loads SNR data, optionally applies refraction correction and decimation, extracts arcs, and optionally saves arc files and attaches processing results.

Parameters

- **station** (*str*) – Station name (4 characters, e.g. 'mchl')
- **year** (*int*) – Full year (e.g. 2025)
- **doy** (*int*) – Day of year (1-366)
- **freq** (*int, list of int, or None*) – Frequency code(s). Default: None (auto-detect)
- **snr_type** (*int*) – SNR file type (66, 77, 88, etc.). Default: 66
- **buffer_hours** (*float*) – Hours of data from adjacent days for midnight-crossing arcs. Default: 2
- **attach_results** (*bool*) – If True, attach gnssir/phase/vwc results to arc metadata. Default: False
- **extension** (*str*) – Strategy extension for result file paths. Default: ''
- **station_config** (*dict, optional*) – Station analysis parameters. When provided, enables refraction correction (if `station_config['refraction']`) and savearcs (if `station_config['savearcs']`).
- **gzip** (*bool*) – If True, gzip the SNR file after reading. Default: True
- **track_file** (*path-like, optional*) – Path to a tracks-shaped JSON file (tracks.json from build_tracks, or vwc_tracks.json from vwc_input). When supplied, each arc's metadata is tagged via tracks.attach_track_id with track_id, track_epoch, track_azim, and (if present in the epoch dict) apriori_RH. Arcs that don't match any track get -1/None.
- **track_cache** (*dict, optional*) – Shared dict for reusing the same tracks JSON across many calls. Pass the same dict on each call; the JSON is loaded and indexed on the first call and reused thereafter.
- **tag_with_legacy_apriori** (*bool*) – When True, tag arcs from the legacy GPS apriori_rh_{fr}.txt file via tracks.attach_legacy_apriori (sets apriori_RH / track_azim on each arc by (sat, azimuth-within-3 deg) matching). Mutually exclusive with track_file. Default: False.

  When neither track_file nor tag_with_legacy_apriori is provided, arcs are returned without track_id / track_epoch / track_azim / apriori_RH tagging.
- **refraction_verbose** (*bool*) – Forwarded as `verbose` to apply_refraction so batch callers can silence the per-day refraction prints. Default: True.
- ****kwargs** – Additional keyword arguments passed to `extract_arcs()`

Returns
See `extract_arcs()` for format details.

Return type
list of (metadata, data) tuples

Raises

- **FileNotFoundError** – If the SNR file does not exist and cannot be decompressed, or if `track_file` is supplied but does not exist.
- **ValueError** – If both `track_file` and `tag_with_legacy_apriori=True` are set.

gnssrefl.extract_arcs.**extract_arcs_from_tracks**(*tracks_json*)

Walk active-epoch days in tracks_json and return tagged (meta, data) arcs.

Robust SNR-walk entry for consumers that need the full per-arc SNR payload tagged against a (possibly QC-edited) in-memory tracks_json. Station and extension come from tracks_json['metadata']; tagging happens via a temp-file round-trip through extract_arcs_from_station's track_file kwarg. Arcs with no matching track are dropped.

Returns a flat list of (metadata, data) tuples in the standard extract_arcs format, concatenated across all active-epoch days.

For the fast summary-only path (results/ + failQC/ with no SNR walk), use load_gnssir_results_from_tracks instead.

gnssrefl.extract_arcs.**load_gnssir_results_from_tracks**(*tracks_json*)

Fast-path summary DataFrame from results/ + failQC/ artifacts.

Walks active-epoch days in tracks_json, reads the gnssir results file and its failQC sibling for each day via load_results_with_failqc, and tags each row against tracks_json via lookup_arc. Requires a prior gnssir run; missing failQC siblings raise FileNotFoundError. Rows with no matching track are dropped.

Returns a DataFrame with columns `mjd, azim, constellation, RH, match_T, track_id, track_epoch`. `match_T` is always NaN so tracks.fit_segment falls back to the constellation's default repeat interval via the constellation column.

For the robust SNR-walk that returns full (meta, data) tuples, use extract_arcs_from_tracks instead.

gnssrefl.extract_arcs.**load_results_with_failqc**(*station*, *year*, *doy*, *extension*, *require_failqc*)

Load the combined results+failQC ndarray for one day.

Reads results/{station}/[{extension}/]{doy:03d}.txt and the sibling failQC/ file written by retrieve_rh. Both files share the RESULT_COLUMNS layout; failQC rows have their RH column overwritten with NaN so that downstream consumers filtering on RH.notna() separate pass from fail without a second column.

Parameters

- **station** (*identifier tuple used by FileManagement.*) –
- **year** (*identifier tuple used by FileManagement.*) –
- **doy** (*identifier tuple used by FileManagement.*) –
- **extension** (*strategy extension string (" for the default strategy).*) –
- **require_failqc** (*bool*) – When True, raise FileNotFoundError if the results file has at least one row but the failQC sibling does not exist. Empty results files (zero-row, e.g. from days where the SNR file had no data) are tolerated because gnssir writes no failQC file in that case. The fast-path tracks loader sets this. When False, missing failQC is silently tolerated (used by the attach_results=['gnssir'] branch of extract_arcs_from_station).

Returns
Combined 2-D array with the same column layout as RESULT_COLUMNS, or None when neither file exists.

Return type
np.ndarray or None

gnssrefl.extract_arcs.**move_arc_to_failqc**(*meta*, *station*, *year*, *doy*, *extension=''*)

Move a saved arc file from arcs/ to arcs/failQC/.

gnssrefl.extract_arcs.**remove_dc_component**(*ele: ndarray*, *snr: ndarray*, *polyV: int*, *dbhz: bool*, *pele: List[float] | None = None*) → ndarray

Remove direct signal component via polynomial fit.

Parameters

- **ele** (*np.ndarray*) – Elevation angles (degrees)
- **snr** (*np.ndarray*) – Raw SNR values
- **polyV** (*int*) – Polynomial order for DC removal
- **dbhz** (*bool*) – If True, keep SNR in dB-Hz; if False, convert to linear units first
- **pele** (*list of float, optional*) – Elevation angle range [min, max] for polynomial fit. If provided, the polynomial is fit on data within this range but evaluated (and removed) over the full arc.

Returns
Detrended SNR data

Return type
np.ndarray

gnssrefl.extract_arcs.**save_arc**(*meta*, *data*, *sdir*, *station*, *year*, *doy*, *savearcs_format='txt'*)

Save a single arc file to sdir.

gnssrefl.extract_arcs.**setup_arcs_directory**(*station*, *year*, *doy*, *extension=''*, *nooverwrite=False*)

Create arcs directory, optionally clearing old contents.

gnssrefl.filesizes module

gnssrefl.filesizes.**main**()

very simple code to pick up all the file sizes for SNR files in a given year only checks for snr66 files.

this is too slow - instead of using numpy array - use a list - and then change to numpy array at the end

It makes a plot - not very useful now that files are nominal gzipped.

Parameters

- **station** (*str*) – 4 character station name
- **year1** (*int, optional*) – beginning year
- **year2** (*int, optional*) – ending year
- **doy1** (*int, optional*) – beginning day of year
- **doy2** (*int, optional*) – ending day of year
- **gz** (*bool, optional*) – say T or True to search for gzipped files

gnssrefl.fundy module

gnssrefl.fundy.**apply_new_azim_mask**(*station*, *file1*, *predictf*, *timeOffset*)

Parameters

- **station** (*str*) – 4 ch station name
- **file1** (*str*) – observation file created by first part of subdaily
- **predictf** (*str*) – file of predicts downloaded from Canadian tide gauge site
- **timeOffset** (*float*) – how much time (in hours) from low tide triggers data deletion

Returns

fout – name of edited obseration file to be used by second part of subdaily

Return type

str

gnssrefl.fundy.**better_high_low_tide**(*mjd*, *water_level*)

gnssrefl.fundy.**dumb_high_low_tide**(*mjd*, *tide*)

goofy way i was originally getting daily high/low tides . . .

gnssrefl.fundy.**read_predicts**(*f*)

reads canadian government csv file and returns mjd and tide

Parameters

f (*str*) – filename

Returns

- **mjd** (*numpy array of floats*) – modified julian day
- **tide** (*numpy array of floats*) – water level in meters

gnssrefl.gnss_frequencies module

Central registry of GNSS frequency and constellation metadata.

All frequency codes, wavelengths, satellite ranges, display labels, and SNR column mappings are defined here. Other modules should use the accessor functions rather than maintaining their own hardcoded lists.

This module has NO dependencies on other gnssrefl modules at import time, so it can be safely imported from anywhere without circular import issues.

gnssrefl.gnss_frequencies.**all_default_frequencies**()

All-constellation default frequency list for gnssir_input -allfreq.

GPS L2 P-code (code 2) is excluded; we prefer L2C (code 20).

gnssrefl.gnss_frequencies.**all_frequencies**()

Return sorted list of all valid frequency codes.

gnssrefl.gnss_frequencies.**get_constellation**(*f*)

Return constellation name for a frequency code.

gnssrefl.gnss_frequencies.**get_display_label**(*f*)

Return display label like 'GPS L2C' for plot titles.

gnssrefl.gnss_frequencies.**get_file_suffix**(*f*)

Return file naming suffix like '_G_L1', '_E_L7', '_C_L5'.

Format is '_<constellation_char>_<signal_label>' using the RINEX 3 constellation chars (G/R/E/C) and band labels (L1/L2/L2C/L5/L6/L7/L8).

gnssrefl.gnss_frequencies.**get_glonass_channel**(*prn*)

Return the FDMA channel number for a GLONASS satellite, or None if unknown.

gnssrefl.gnss_frequencies.**get_glonass_wavelength**(*f*, *prn*)

Return GLONASS wavelength in meters for frequency code f and satellite prn.

GLONASS uses FDMA so each satellite transmits on a slightly different carrier determined by its channel number. f is 101 (G1) or 102 (G2). Raises ValueError if the satellite slot has no known channel assignment.

gnssrefl.gnss_frequencies.**get_sat_list**(*f*)

Return numpy array of satellite PRNs for a frequency's constellation.

gnssrefl.gnss_frequencies.**get_sat_range**(*f*)

Return (start, stop) tuple for np.arange to build a satellite list.

gnssrefl.gnss_frequencies.**get_scale_factor**(*f*, *sat=None*)

Return wavelength/2 (the LSP scale factor cf).

gnssrefl.gnss_frequencies.**get_signal_label**(*f*)

Return signal label like 'L1', 'L2C', 'L5'.

gnssrefl.gnss_frequencies.**get_snr_column**(*f*)

Return 1-based SNR file column index for a frequency code.

gnssrefl.gnss_frequencies.**get_wavelength**(*f*, *sat=None*)

Return wavelength in meters. For GLONASS, sat number is required.

gnssrefl.gnss_frequencies.**gps_default_frequencies**()

Default GPS-only frequency list for gnssir_input.

gnssrefl.gnss_frequencies.**is_valid_frequency**(*f*)

Check whether a frequency code is recognized.

gnssrefl.gnss_frequencies.**signal_label_to_freq**(*constellation_char*, *signal_label*)

Return the frequency code for a RINEX constellation char and signal label.

Parameters

- **constellation_char** (*str*) – Single RINEX char: 'G' (GPS), 'R' (GLONASS), 'E' (Galileo), 'C' (BeiDou)
- **signal_label** (*str*) – Signal label like 'L1', 'L2', 'L5', 'L6', 'L7', 'L8', 'L2C'

Returns

Frequency code (e.g. 1, 20, 205, 302)

Return type

int

Raises

KeyError – If the (constellation_char, signal_label) pair is not recognized

gnssrefl.gnss_frequencies.**wl**(*freq_mhz*)

Wavelength in meters from frequency in MHz.

gnssrefl.gnssir_cl module

gnssrefl.gnssir_cl.**count_result_arcs**(*result_path*)

Count non-comment lines in a result file.

gnssrefl.gnssir_cl.**gnssir**(*station: str*, *year: int*, *doy: int*, *snr: int = 66*, *plt: bool = False*, *fr: list = []*, *ampl: float | None = None*, *sat: int | None = None*, *doy_end: int | None = None*, *year_end: int | None = None*, *azim1: int = 0*, *azim2: int = 360*, *nooverwrite: bool = False*, *extension: str = ''*, *compress: bool = False*, *screenstats: bool = True*, *delTmax: int | None = None*, *e1: float | None = None*, *e2: float | None = None*, *mmdd: bool = False*, *gzip: bool | None = None*, *dec: int = 1*, *savearcs: bool = False*, *savearcs_format: str = 'txt'*, *par: int | None = None*, *debug: bool = False*, *midnite: bool = True*, *dbhz: bool = False*, *lsp_method: str = 'fast'*)

gnssir is the main driver for estimating reflector heights. The user is required to have set up an analysis strategy using gnssir_input.

screenstats is always True now - and the information is written to a file. I have kept the optional parameter for backwards compatability, but it does not do anything.

To improve screen output when your job crashes, try -debug T

"secret limits" : arcs must be at least one degree long (in elevation angle). I can change that - but just a warning.

Parallel processing is now available. If you set -par to an integer between 2 and 10, it should substantially speed up your processing. Big thank you to AaryanRampal for getting this up and running. If you are using the docker, you will need to experiment about how to use this - as they have requirements for multiple processes that I do not know about.

As of v3.6. there is a way to save individual rising and setting arcs to an external file. You can then use them as you wish. The default is plain text but only has elevation angles and deltaSNR (SNR with direct signal removed). You can also save more information in a pickle file. Just say -savearcs_format pickle. Both require -savearcs T to set this option. The location of the files is printed to the screen. If an arc does not pass QC, it is saved, but in a separate directory with the name failQC added to it.

If you are using the non-standard snr files (i.e. not 66), you have been required to provide an online parameter every time you run gnssir. As of v 3.6.6, you can now save a parameter called snr when you use gnssir_input. So that would automate it for you. If you haven't done that then you should use snr on the command line and set it to the appropriate value.

Examples

gnssir p041 2021 15
analyzes the data for station p041, year 2021 and day of year 15.

gnssir p041 2021 15 -savearcs T
prints out individual arcs to $REFL_CODE/2021/arcs/p041/015

gnssir p041 2021 1 -doy_end 100 -par 10
analyze 100 days of data - but spawn 10 processes at a time. Big cpu time savings.

gnssir p041 2021 15 -snr 99
uses SNR files with a 99 suffix

gnssir p041 2021 15 -plt T
plots of SNR data and periodograms come to the screen. Each frequency gets its own plot.

gnssir p041 2021 15 -screenstats T
sends more information to the screen

gnssir p041 2021 15 -nooverwrite T
only runs gnssir if there isn't a previous solution

gnssir p041 2021 15 -extension strategy1
runs gnssir using json file called p041.strategy1.json

gnssir p041 2021 15 -doy_end 20
Analyzes data from day of year 15 to day of year 20

gnssir p041 2021 15 -dec 5
before computing periodograms, decimates the SNR file contents to 5 seconds

gnssir p041 2021 15 -gzip T
gzips the SNR file after you run the code. Big space saver (now the default)

gnssir p041 2021 15 -fr 1 101
ignore frequency list in your json and use frequencies 1 and 101

Parameters

- **station** (*str*) – lowercase 4 character ID of the station
- **year** (*int*) – full year
- **doy** (*integer*) – Day of year
- **snr** (*int, optional*) – SNR format. This tells the code what elevation angles to save data for. Input is the snr file ending. Value options:

  66 (default) : saves all data with elevation angles less than 30 degress

  99 : saves all data with elevation angles between 5 and 30 degrees

  88 : saves all data

  50 : saves all data with elevation angles less than 10 degrees
- **plt** (*bool, optional*) – Send plots to screen or not. Default is False.
- **fr** (*int, optional*) – As of version 3.5.9 you are allowed to enter more than one GNSS frequency. It overrides whatever was stored in your json. This is entered into a list so do not use commas, i.e. 1 101 102 allowed frequency names:

  1,2,20,5 : GPS L1, L2, L2C, L5

  101,102 : GLONASS L1, L2

  201, 205,206,207,208 : GALILEO E1, E5a,E6,E5b,E5

  301,302,305,306,307,308 : BEIDOU (See RINEX 3 format description for details)
- **ampl** (*float, optional*) – minimum spectral peak amplitude. default is None
- **sat** (*int, optional*) – satellite number to only look at that single satellite. default is None.
- **doy_end** (*int, optional*) – end day of year. This is to create a range from doy to doy_end of days. If year_end parameter is used - then day_end will end in the day of the year_end. Default is None. (meaning only a single day using the doy parameter)
- **year_end** (*int, optional*) – end year. This is to create a range from year to year_end to get the snr files for more than one year. doy_end will be for year_end. Default is None.
- **azim1** (*int, optional*) – lower limit azimuth. If the azimuth angles are changed in the json (using 'azval2' key) and not here, then the json overrides these. If changed here, then it overrides what you requested in the json. default is 0.

- **azim2** (*int, optional*) – upper limit azimuth. If the azimuth angles are changed in the json (using 'azval2' key) and not changed here, then the json overrides these. If changed here, then it overrides what you requested in the json. default is 360.
- **nooverwrite** (*bool, optional*) – Use to overwrite lomb scargle result files or not. Default is False, i.e., it will overwrite.
- **extension** (*string, optional*) – extension for result file, useful for testing strategies. default is empty string
- **compress** (*boolean, optional*) – xz compress SNR files after use. default is False.
- **screenstats** (*bool, optional*) – whether to print stats to the screen or not. default is True.
- **delTmax** (*int, optional*) – maximum satellite arc length in minutes. found in the json
- **e1** (*float, optional*) – use to override the minimum elevation angle.
- **e2** (*float, optional*) – use to override the maximum elevation angle.
- **mmdd** (*boolean, optional*) – adds columns in results for month, day, hour, and minute. default is False.
- **gzip** (*boolean, optional*) – gzip compress SNR files after use. default is True (as of 2023 Sep 17).
- **dec** (*int, optional*) – decimate SNR file to this sampling period before the periodograms are computed. 1 sec is default (i.e. no decimating)
- **savearcs** (*bool, optional*) – save arcs in individual files (elevation angle and deltaSNR)
- **savearcs_format** (*str, optional*) – format of saved arc files, txt or pickle. default is txt
- **par** (*int, optional*) – number of parallel processing jobs.
- **debug** (*bool, optional*) – remove the primary call from try/except so that you have a better idea of why the code might be crashing. No parallel processing in this mode
- **midnite** (*bool*) – whether arcs can cross midnite
- **dbhz** (*bool*) – whether to keep SNR data in db-hz. default (false) is to convert to linear scale

gnssrefl.gnssir_cl.**main**()

gnssrefl.gnssir_cl.**parse_arguments**()

gnssrefl.gnssir_cl.**process_day_worker**(*worker_args*)

Worker for parallel gnssir processing. Returns arc count for progress bar.

gnssrefl.gnssir_cl.**process_year**(*year*, *year_end*, *doy*, *doy_end*, *args*)

Sequential processing for gnssir.

gnssrefl.gnssir_input module

gnssrefl.gnssir_input.**main**()

gnssrefl.gnssir_input.**make_gnssir_input**(*station: str*, *lat: float = 0*, *lon: float = 0*, *height: float = 0*, *e1: float = 5.0*, *e2: float = 25.0*, *h1: float = 0.5*, *h2: float = 8.0*, *nr1: float | None = None*, *nr2: float | None = None*, *peak2noise: float = 2.8*, *ampl: float = 5.0*, *allfreq: bool = False*, *l1: bool = False*, *l2c: bool = False*, *l5: bool = False*, *xyz: bool = False*, *refraction: bool = True*, *extension: str = ''*, *ediff: float = 2.0*, *delTmax: float = 75.0*, *frlist: list = []*, *azlist2: list = [0, 360]*, *ellist: list = []*, *refr_model: str = '1'*, *apriori_rh: float | None = None*, *Hortho: list | None = None*, *pele: list = [5, 30]*, *polyV: int = 4*, *daily_avg_reqtracks: int | None = None*, *daily_avg_medfilter: float | None = None*, *subdaily_alt_sigma: bool | None = None*, *subdaily_ampl: float | None = None*, *subdaily_delta_out: float | None = None*, *subdaily_knots: int | None = None*, *subdaily_sigma: float | None = None*, *subdaily_subdir: str | None = None*, *subdaily_spline_outlier1: float | None = None*, *subdaily_spline_outlier2: float | None = None*, *snr: int | None = None*, *stream: str | None = None*, *samplerate: int | None = None*, *dec: int | None = None*, *orb: str | None = None*, *archive: str | None = None*, *Hdates: str | None = None*, *gzip: bool = True*)

This new script sets the Lomb Scargle analysis strategy you will use in gnssir. It saves your inputs to a json file which by default is saved in REFL_CODE/<station>.json. This code replaces make_json_input.

This version no longer requires you to have azimuth regions of 90-100 degrees. You can set a single set of azimuths in the command line variable azlist2, i.e. -azlist2 0 270 would accommodate all rising and setting arcs between 0 and 270 degrees. If you have multiple distinct regions, that is also acceptable, i.e. -azlist2 0 150 180 360 would use all azimuths between 0 and 360 except for 150 to 180

Your first azimuth constraint can be negative, i.e. -azlist2 -90 90, is allowed.

Note: you can keep using your old json files - you just need to add this new -azlist2 setting manually.

Hortho can be added to your json so that you can save orthometric height used in subdaily. Multiple Hortho values can be input if you moved your site. You will need to associates times with them, which can be done using Hdates, in this format, 2024-01-01 15:29. subdaily will read the json to get these values.

Latitude, longitude, and height are assumed to be stored in either the UNR database we provide with gnssrefl or in your local coordinate file. See the instructions in the file formats section of gnssrefl for information about the format, name, and location of that local coordinate file.

Originally we had refraction as a boolean, i.e. on or off. This was stored in the gnssir analysis description json. The code however, uses a 1 for a simple non-time-varying Bennett correction and 0 for no correction.

From version 1.8.4 we begin to implement more refraction models. Model 1 (Bennett) will continue to be the default. The model number is written (as an integer) to the LSP results file so that people can keep track easily of whether they are inadvertently mixing files with different strategies. And that is why it is an integer, because all results in the LSP results files are numbers. Going forward, we are adding a time-varying capability.

Model 1: Bennett, non-time-varying

Model 2: Bennett, time-varying

Model 3: Ulich, non-time-varying

Model 4: Ulich, time-varying

Model 5: NITE, Feng et al. 2023 DOI: 10.1109/TGRS.2023.3332422, time-varying

Model 6: MPF, Williams and Nievinski, 2017, DOI: 10.1002/2016JB013612, time-varying

We allow users to input the model names NITE and MPF (nite and mpf also allowed). The other models do not allow that. If you want model 2, you have to ask for model 2. We thank Peng Feng for providing python code to be used for some of these models.

If you want to test the effect of different refraction models, you are encouraged to create two json files using the extension option. You can then run gnssir using those two extensions. In general I think the refraction default is fine for soil moisture and snow accumulation. If you are going to look at tall sites, you most definitely need the refraction correction. If you plan to look at very tall sites, you should pick the best one.

Examples

gnssir_input p041
uses only GPS frequencies and all azimuths and the coordinates in the UNR database

gnssir_input p041 -azlist2 0 180 -fr 1 101
uses UNR coordinates, GPS L1 and Glonass L1 frequencies, and azimuths between 0 and 180.

gnssir_input p041 -lat 39.9494 -lon -105.19426 -height 1728.85 -l2c T -e1 5 -e2 15
uses only L2C GPS data between elevation angles of 5 and 15 degrees. user input lat/long/height. The lat/long/height can also be entered into a local coordinate file. See documentation in the file formats section.

gnssir_input p041 -h1 0.5 -h2 10 -e1 5 -e2 25
uses UNR database, only GPS data between elevation angles of 5-25 degrees and reflector heights of 0.5-10 meters

gnssir_input p041 -ediff 1
uses UNR database, only GPS data, default station coordinates, enforces elevation angles to be within 1 degrees of default elevation angle limits (5-25)

gnssir_input sc02 -ellist 5 10 7 12
let's say you want to compute smaller arcs than just a single set of elevation angles. you can use this to set this up, so instead of 5 and 12, you could set it up to do two arcs, one for 5-10 degrees and the other for 7-12. WARNING: you need to pay attention to QC metrics (amplitude and peak2noise). You likely need to lower them since your periodogram for fewer data will be less robust than with the longer elevation angle region. WARNING: these are pairs. Don't give the code an odd number of values.

Parameters

- **station** (*str*) – 4 character station ID.
- **lat** (*float, optional*) – latitude in degrees.
- **lon** (*float, optional*) – longitude in degrees.
- **height** (*float, optional*) – ellipsoidal height in meters.
- **e1** (*float, optional*) – elevation angle lower limit in degrees. default is 5.
- **e2** (*float, optional*) – elevation angle upper limit in degrees. default is 25.
- **h1** (*float, optional*) – reflector height lower limit in meters. default is 0.5.
- **h2** (*float, optional*) – reflector height upper limit in meters. default is 8.
- **nr1** (*float, optional*) – noise region lower limit for QC in meters. default is None.
- **nr2** (*float, optional*) – noise region upper limit for QC in meters. default is None.

- **peak2noise** (*float, optional*) – peak to noise ratio used for QC. default is 2.7 (just a starting point for water - should be 3 or 3.5 for snow or soil. . .)
- **ampl** (*float, optional*) – spectral peak amplitude for QC. default is 6.0 this is receiver and elevation angle region dependent - so you need to change it based on your site
- **allfreq** (*bool, optional*) – True requests all GNSS frequencies. default is False (defaults to use GPS frequencies).
- **l1** (*bool, optional*) – set to True to use only GPS L1 frequency. default is False.
- **l2c** (*bool, optional*) – set to use only GPS L2C frequency. default is False.
- **xyz** (*bool, optional*) – set to True if using Cartesian coordinates instead of Lat/Long/Ht. default is False.
- **refraction** (*bool, optional*) – set to False to turn off refraction correction. default is True.
- **extension** (*str, optional*) – provide extension name so you can try different strategies. Results will then go into $REFL_CODE/YYYY/results/ssss/extension Default is ‘’
- **ediff** (*float, optional*) – quality control parameter (Degrees) Allowed min/max elevation angle diff from requested min/max elev angle default is 2
- **delTmax** (*float, optional*) – maximum allowed arc length (minutes) default is 75, which can be a bit long for tides
- **frlist** (*list of integers*) – avoids all the booleans - if you know the frequencies, enter them. e.g. 1 2 or 1 20 5 or 1 20 101 102
- **azlist2** (*list of floats*) – Default is 0 to 360. list of azimuth limits as subquadrants are no longer required.
- **ellist** (*list of floats*) – min and max elevation angles to be used with the azimuth regions you listed, i.e. [5 10 6 11 7 12 8 13] would allow overlapping regions - all five degrees long Default is empty list.
- **refr_model** (*str*) – refraction model. While defined as a string (so that people can specify names of models) we convert this to an integer for book-keeping. 1 is the default refraction model (it corrects elevation angles using standard bending models). 0 is no refraction correction. The other models are defined in the summary section of this code.
- **apriori_rh** (*float*) – apriori reflector height (meters). only used in NITE model
- **Hortho** (*list*) – station orthometric height, in meters. Currently used in subdaily and daily_avg. If not provided on the command line, it will use ellipsoidal height and EGM96 to compute. Advanced users can set more than one Hortho if they provide a Hdates list (see below).
- **pele** (*float*) – min and max elevation angles in direct signal removal, i.e. 3 40. Default is 5 30.
- **polyV** (*int*) – polynomial order used in direct signal removal. The default is 4, but you can set it to something different for your specific antenna and elevation angle range.
- **daily_avg_reqtracks** (*int, optional*) – number of tracks required for daily_avg code
- **daily_avg_medfilter** (*float, optional*) – median filter value required for daily_avg code (meters)
- **subdaily_alt_sigma** (*bool, optional*) – use Nievinski sigma definition
- **subdaily_ampl** (*float, optional*) – override the required LSP amplitude

- **subdaily_delta_out** (*int, optional*) – spacing for final subdaily spline output
- **subdaily_knots** (*int, optional*) – number of knots per day for subdaily spline fits
- **subdaily_sigma** (*float, optional*) – how many standard deviations for outliers in subdaily code setting
- **subdaily_subdir** (*str, optional*) – non-standard location for subdaily outputs
- **subdaily_spline_outlier1** (*float, optional*) – alternate setting for outlier detection in part1
- **subdaily_spline_outlier2** (*float, optional*) – alternate setting for outlier detection in part2
- **snr** (*int*) – This denotes the kind of SNR file when either creating or using SNR files. If using the default (66), there is no reason to set this. If you are going to use non-defaults (i.e. 88) throughout, it would be helpful to set this here and then the value will be used when using gnssir. If you do set it, it will also be used by rinex2snr, which again can be useful.
- **stream** (*str, optional*) – for RINEX3 translation only, R or S naming parameter set to R
- **samplerate** (*int , optional*) – for RINEX3 translation only, file sample rate to be used set to None for now
- **orb** (*str, optional*) – for SNR file creation. If nothing is provided, nothing is written to the json. Can be useful if you want to use a specific orbit source (that the code recognizes)
- **archive** (*str, optional*) – for SNR file creation. If nothing is provided, nothing is written to the json. Can be useful if you forget which archive has which station files.
- **Hdates** (*str, optional*) – This variable is used to allow time information for different Hortho values, as might happen if you moved your GNSS antenna vertical between installations. Uses format of 2024-11-01 15:22 You must include all of these values in this format (i.e. you cannot leave off HH:MM)

gnssrefl.gnssir_input.**parse_arguments**()

gnssrefl.gnssir_v2 module

gnssrefl.gnssir_v2.**check_azim_compliance**(*az_min_ele*, *azlist*)

Check to see if your arc is in one of the requested regions

Parameters

- **az_min_ele** (*float*) – azimuth of selected arc (deg)
- **azlist** (*list of floats*) – list of acceptable azimuth regions

Returns

keeparc – whether the arc is in a selected azimuth range

Return type

bool

gnssrefl.gnssir_v2.**convert_Hdates_mjd**(*Hdates*, *remove_hhmm*)

takes a list of dates in format yyyy-mm-dd hh:mm and turns them into a list of mjd

Parameters

- **Hdates** (*list of str*) – date strings in the format yyyy-mm-dd hh:mm

- **remove_hhmm** (*bool*) – whether you want to ignore hh:mm

Returns
mjd_Hortho – modified julian dates of character string dates

Return type
list of floats

gnssrefl.gnssir_v2.**find_mgnss_satlist**(*f*, *year*, *doy*)

find satellite list for a given frequency and date

Parameters

- **f** (*integer*) – frequency
- **snrExist** (*numpy array, bool*) – tells you if a signal is (potentially) legal
- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns
satlist – satellites to use

Return type
numpy list of integers

gnssrefl.gnssir_v2.**gnssir_guts_v2**(*station*, *year*, *doy*, *snr_type*, *extension*, *station_config*, *debug*)

Computes lomb scargle periodograms for a given station, year, day of year etc.

Arcs are determined differently than in the first version of the code, which was quadrant based. This identifies arcs and applies azimuth constraints after the fact.

2023-aug-02 trying to fix the issue with azimuth print out being different than azimuth at lowest elevation angle

if screenstats is True, it prints to a log file now, directory $REFL_CODE/logs/ssss

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snr_type** (*int*) – snr file type
- **extension** (*str*) – optional subdirectory to save results
- **station_config** (*dictionary*) –

  e1
  [float] min elev angle, deg

  e2
  [float] max elev angle, deg

  freqs: list of int
  frequencies to use

  minH
  [float] min reflector height, m

  maxH
  [float] max reflector height, m

NReg
[list of floats] noise region for RH peak2noise , meters

azval2
[list of floats] new pairs of azimuth regions, i.e. [0 180 270 360]

delTmax
[float] max allowed arc length in minutes

pele: list of floats
min and max elev angle in DC removal

PkNoise
[float] peak to noise value for QC

ediff
[float] elev angle difference for arc length, QC

reqAmp
[list of floats] list of required periodogram amplitude for QC for each frequency

ellist: list of floats
added 23jun16, allow multiple elevation angle regions

apriori_rh
[float] a priori reflector height, used in NITE, meters

savearcs
[bool] if true, elevation angle and detrended SNR data are saved for each arc default is False

savearcs_format
[str] if arcs are to be saved, will they be txt or pickle format

midnite
[bool] whether midnite arcs are alloweed

dbhz
[bool] whether db-hz (True) or volts/volts (False) are used for SNR data

- **debug** (*bool*) – debugging value to help track down bugs

gnssrefl.gnssir_v2.**gzip_migration**(*station_config*, *station*, *extension=''*)

Temporary migration (remove after 2027-01-01): old JSON configs had gzip defaulting to False due to a bug in gnssir_input. This function overrides gzip to True, updates the JSON on disk, and warns the user.

Call this after read_json_file() and before using station_config['gzip']. Pass -gzip F on the command line to opt out.

gnssrefl.gnssir_v2.**local_update_plot**(*x*, *y*, *px*, *pz*, *ax1*, *ax2*, *failure*)

updates optional result plot for SNR data and Lomb Scargle periodograms

Parameters

- **x** (*numpy array*) – elevation angle (deg)
- **y** (*numpy array*) – SNR (volt/volt)
- **px** (*numpy array*) – reflector height (m)
- **pz** (*numpy array*) – spectral amplitude (volt/volt)
- **ax1** (*matplotlib figure control*) – top plot
- **ax2** (*matplotlib figure control*) – bottom plot

- **failure** (*boolean*) – whether periodogram fails QC

gnssrefl.gnssir_v2.**make_parallel_proc_lists**(*year*, *doy1*, *doy2*, *nproc*)

make lists of dates for parallel processing to spawn multiple jobs

Parameters

- **year** (*int*) – year of processing
- **doy1** (*int*) – start day of year
- **2** (*doy*) – end day of year

Returns

- **datelist** (*dict*) – list of dates formatted as year doy1 doy2
- **numproc** (*int*) – number of datelists, thus number of processes to be used

gnssrefl.gnssir_v2.**make_parallel_proc_lists_mjd**(*year*, *doy*, *year_end*, *doy_end*, *nproc*)

make lists of dates for parallel processing to spawn multiple jobs

Parameters

- **year** (*int*) – year processing begins
- **doy** (*int*) – start day of year
- **year_end** (*int*) – year end of processing
- **doy_end** (*int*) – end day of year
- **nproc** (*int*) – requested number of processes to spawn

Returns

- **datelist** (*dict*) – list of MJD
- **numproc** (*int*) – number of datelists, thus number of processes to be used

gnssrefl.gnssir_v2.**new_rise_set**(*elv*, *azm*, *dates*, *e1*, *e2*, *ediff*, *sat*, *screenstats*)

This provides a list of rising and setting arcs for a given satellite in a SNR file based on using changes in elevation angle

Parameters

- **elv** (*numpy array of floats*) – elevation angles from SNR file
- **azm** (*numpy array of floats*) – azimuth angles from SNR file
- **dates** (*numpy array of floats*) – seconds of the day from SNR file
- **e1** (*float*) – min elevation angle
- **e2** (*float*) – max elevation angle
- **ediff** (*float*) – el angle difference Quality control parameter
- **sat** (*int*) – satellite number

Returns

tv – beginning and ending indices of the arc satellite number, arc number

Return type

numpy array

gnssrefl.gnssir_v2.**open_gnssir_logfile**(*station*, *year*, *doy*, *extension*)

opens a logfile when asking for screen output

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **extension** (*str*) – analysis extension name (for storage of results) if not set you should send empty string

Returns

fileid – I don't know the proper name of this - but what comes out when you open a file so you can keep writing to it

Return type

?

gnssrefl.gnssir_v2.**plot2screen**(*station*, *f*, *ax1*, *ax2*, *pltname*)

Add axis information and Send the plot to the screen. https://www.semicolonworld.com/question/57658/matplotlib-adding-an-axes-using-the-same-arguments-as-a-previous-axes

Parameters

station (*string*) – 4 character station ID

gnssrefl.gnssir_v2.**read_json_file**(*station*, *extension=''*, ***kwargs*)

picks up json instructions for calculation of lomb scargle periodogram This was originally meant to be used by gnssir, but is now read by other functions.

Uses new directory structure: - No extension: input/{station}/{station}.json with fallback to input/{station}.json - With extension: input/{station}/{extension}/{station}.json with fallback to input/{station}.{extension}.json

Parameters

- **station** (*str*) – 4 character station name
- **extension** (*str*) – subdirectory - default is ''

Returns

station_config

Return type

dictionary

gnssrefl.gnssir_v2.**retrieve_Hdates**(*a*)

Retrieves character strings of dates and attempts to QC them.

Parameters

a (*list of str*) – online input to gnssir_input for Hdates

Returns

Hdate – full dates (2024-10-11 15:12) of Hortho values

Return type

list of str

gnssrefl.gnssir_v2.**rewrite_azel**(*azval2*)

Trying to allow regions that cross zero degrees azimuth

Parameters
azval2 (*list of floats*) – input azimuth regions

Returns
azelout – azimuth regions without negative numbers …

Return type
list of floats

gnssrefl.gnssir_v2.**window_new**(*snrD*, *f*, *satNu*, *ncols*, *pfitV*, *e1*, *e2*, *azlist*, *screenstats*, *fileid*, *dbhz*, ***kwargs*)

retrieves SNR arcs for a given satellite. returns elevation angle and detrended linear SNR

2023-aug02 updated to improve azimuth calculation reported

2024-aug-15 testing out imposing pele values for DC removal. 2024-sep04 removed pele as input

Parameters

- **snrD** (*numpy array (multiD)*) – contents of the snr file, i.e. 0 column is satellite numbers, 1 column elevation angle …
- **f** (*int*) – frequency you want
- **satNu** (*int*) – requested satellite number
- **ncols** (*int*) – how many columns does the SNR file have
- **pfitV** (*float*) – polynomial order
- **e1** (*float*) – requested min elev angle (deg)
- **e2** (*float*) – requested max elev angle (deg)
- **azlist** (*list of floats (deg)*) – non-continguous azimuth regions, corrected for negative regions
- **screenstats** (*bool*) – whether you want debugging information printed to the screen
- **fileid** – log location
- **dbhz** (*bool*) – whether you want dbhz or linear SNR units

Returns

- **x** (*numpy array of floats*) – elevation angle, degrees
- **y** (*numpy array of floats*) – linear SNR with DC removed
- **Nvv** (*int*) – number of points in x/y array
- **cf** (*float*) – scale factor for requested frequency (used in LSP)
- **meanTime** (*float*) – UTC hour of the day (GPS time)
- **avgAzim** (*float*) – average azimuth of the arc (deg) ### this will not be entirely consistent with other metric
- **outFact1** (*float*) – kept for backwards compatibility. set to zero
- **outFact2** (*float*) – edot factor used in RH dot correction
- **delT** (*float*) – arc length in minutes
- **secxonds** (*numpy array of floats*) – hopefully seconds of the day

gnssrefl.gps module

gnssrefl.gps.**LSPresult_name**(*station*, *year*, *doy*, *extension*)

Makes filename for the Lomb Scargle output if extension is not being used you should send it empty string (‘’)

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **extension** (*str*) – name of subdirectory for results

Returns

- **filepath1** (*str*) – where Lomb Scargle output goes
- **fileexists** (*bool*) – whether output already exists

gnssrefl.gps.**UNR_highrate**(*station*, *year*, *doy*)

picks up the 5 minute time series from UNR website for a given station

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **filename** (*str*) – output filename
- **goodDownload** (*bool*) – whether your download was successful

gnssrefl.gps.**another_gfz_orbits**(*year*, *month*, *day*, *orbtype*, *hour*)

a function to access yet more differently named GFZ orbits …

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month
- **orbtype** (*str*) – kind of orbit, rapid or final
- **hour** (*int*) – only for ultra

Returns

- **littlename** (*str*) – name of the orbit file
- **fdir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**avoid_cddis**(*year*, *month*, *day*)

work around for people that can’t use CDDIS ftps this will get multi-GNSS files for GFZ from the IGN hopefully

Parameters

- **year** (*int*) – full year

- **month** (*int*) – month of the year
- **day** (*int*) – calendar day

Returns

- **filename** (*str*) – name of the orbit file
- **fdir** (*str*) – where the orbit file is stored
- **foundit** (*bool*) – whether it was found or not

gnssrefl.gps.**azimuth_angle**(*RecSat*, *East*, *North*)

Given cartesian Receiver-Satellite vectors, and East and North unit vectors, computes azimuth angle

Parameters

- **RecSat** (*3-vector*) – meters
- **East** (*3-vector*) – unit vector in east direction
- **North** (*3-vector*) – unit vector in north direction

Returns
azangle – azimuth angle in degrees

Return type
float

gnssrefl.gps.**back2thefuture**(*iyear*, *idoy*)

code checks that this is not a day in the future also rejects data before the year 2000

Parameters

- **iyear** (*int*) – full year
- **idoy** (*int*) – day of year

Returns
badDay – whether your day exists (yet)

Return type
bool

gnssrefl.gps.**bfg_data**(*fstation*, *year*, *doy*, *samplerate=30*, *debug=False*)

Picks up a RINEX3 file from BFG network

Parameters

- **fstation** (*string*) – 4 char station ID
- **year** (*integer*) – year
- **doy** (*integer*) – day of year
- **samplerate** (*integer*) – sample rate of the receiver (default is 30)
- **debug** (*boolean*) – directory file listing provided if true default is false

gnssrefl.gps.**bfg_password**(***kwargs*)

Picks up BFG userid and password that is stored in a pickle file in your REFL_CODE/Files/passwords area If it does not exist, it asks you to input the values and stores them for you.

Allows you to send another name for your password

Returns

- **userid** (*str*) – BFG username
- **passpord** (*str*) – password for archive

gnssrefl.gps.**big_Disk_in_DC_hourly**(*station*, *year*, *month*, *day*, *idtag*)

Picks up a one hour RINEX file from CORS. and gunzips it

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*integer*) – day of the month. if zero, it means month is really day of year
- **idtag** (*str*) – small case letter from a to x; tells the code which hour it is

gnssrefl.gps.**big_Disk_work_hard**(*station*, *year*, *month*, *day*, *delete_hourly*)

Attempts to pick up subdaily RINEX 2.11 files from the NGS archive creates a single RINEX file

If day is 0, then month is presumed to be the day of year

Requires gfzrnx for merging.

Parameters

- **station** (*str*) – 4 char station name
- **year** (*int*) – year
- **month** (*int*) – month
- **day** (*integer*) – day
- **delete_hourly** (*bool*) – whether hourly files are deleted

gnssrefl.gps.**binary**(*string*)

changes python string to bytes for use in fortran code using f2py via numpy input is a string, output is bytes with null at the end

gnssrefl.gps.**cdate2nums**(*col1*)

returns fractional year from ch date, e.g. 2012-02-15 if time is blank, return 3000

Parameters
col1 (*str*) – date in yyyyy-mm-dd, 2012-02-15

Returns
t – fractional date, year + doy/365.25

Return type
float

gnssrefl.gps.**cdate2ydoy**(*col1*)

returns year and day of year from character date, e.g. '2012-02-15'

Parameters
col1 (*str*) – date in yyyyy-mm-dd, 2012-02-15

Returns

- **year** (*int*) – full year
- **doy** (*int*) – day of year

gnssrefl.gps.**cddis_download_2022B**(*filename*, *directory*)

Nth iteration of download code for CDDIS

Parameters

- **filename** (*str*) – name of the rinex file or orbit file
- **directory** (*str*) – where the file lives at CDDIS

gnssrefl.gps.**cddis_download_2022B_new**(*filename*, *directory*)

download code for CDDIS using https and password

Parameters

- **filename** (*str*) – name of the rinex file or orbit file
- **directory** (*str*) – where the file lives at CDDIS

gnssrefl.gps.**cddis_password**()

Picks up cddis userid and password that is stored in a pickle file in your REFL_CODE/Files/passwords area If it does not exist, it asks you to input the values and stores them for you.

Returns

- **userid** (*str*) – cddis username
- **password** (*str*) – cddis password

gnssrefl.gps.**cddis_restriction**(*iyear*, *idoy*, *archive*)

CDDIS has announced a restructuring of their archive. After 6 months files are tarred. It would be ok for the code to accommodate this change, but it will have to come from the community. If six months has passed since you ran the code, a warning will come to the screen and the code will exit.

Updated now that i realize BKG does the same thing

Parameters

- **iyear** (*int*) – year you want to download from CDDIS
- **idoy** (*int*) – day of year you want to download from CDDIS
- **archive** (*str*) – name of archive

Returns
bad_day – if bad_day is true, you cannot access high-rate data from CDDIS or BKG

Return type
bool

gnssrefl.gps.**char_month_converter**(*month*)

integer month to 3 character month

Parameters
month (*int*) – integer month (1-12)

Returns
month – three char month, uppercase

Return type
str

gnssrefl.gps.**checkEGM**()

Downloads and stores EGM96 file in REFL_CODE/Files for use in refl_zones

Returns
foundfile – whether EGM96 file was found (or installed) on your local machine

Return type
bool

gnssrefl.gps.**checkFiles**(*station*, *extension*)

apparently no one consistently checks for the Files directory existence. this is an attempt to fix that.

Parameters

- **station** (*str*) – 4 ch station ID
- **extension** (*str*) – subdirectory for results in $REFL_CODE/Files/station

gnssrefl.gps.**check_environ_variables**()

Checks to see if you have set the expected environment variables used in gnssrefl

gnssrefl.gps.**check_inputs**(*station*, *year*, *doy*, *snr_type*)

inputs to Lomb Scargle and Rinex translation codes are checked for sensibility. Returns true or false so code can exit.

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snr_type** (*int*) – snr file type (e.g. 66)

Returns
exitSys – whether fatal error is trigger by a bad choice

Return type
bool

gnssrefl.gps.**check_navexistence**(*year*, *month*, *day*)

Check to see if you already have the nav file. Uncompresses it if necessary

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or doy if day is zero
- **day** (*int*) – day of month or 0

Returns
foundit – whether nav file has been found

Return type
boolean

gnssrefl.gps.**confused_obstimes**(*tvd*)

this will be slow (and should be fixed)

Parameters
tvd (*numpy array*) – results of LSP results

Returns
modifiedjulian – modified julian date values

Return type
numpy array of floats

class gnssrefl.gps.**constants**

Bases: object

c = 299792458

mu = 398600500000000.0

omegaEarth = 7.2921151467e-05

gnssrefl.gps.**crx2rnx**(*crnx_filename*)

Parameters
crnx_filename (*str*) – name of the hatanaka compressed file. can be either gzipped or not

Returns
rnx_filename – name of the unzipped and hatanaka uncompressed file

Return type
str

gnssrefl.gps.**datestring_mjd**(*H*)

Parameters
H (*str*) – date is of form 2024-10-01 15:22 I think it will work with only 2024-10-01

Returns
mjd – modified julian day

Return type
float

gnssrefl.gps.**dec31**(*year*)

Calculates the day of year for December 31

Parameters
input (*int*) – year

Returns
doy – day of year for December 31

Return type
int

gnssrefl.gps.**define_logdir**(*station*, *year*, *doy*)

creates logfile name and directory (for rinex2snr) given a station, year and day of year

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

gnssrefl.gps.**define_quick_filename**(*station*, *year*, *doy*, *snr*)

defines SNR File name

Parameters

- **station** (*str*) – 4 ch station name

- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snr** (*int*) – snr file type (66,88, etc)

Returns
f – SNR filename

Return type
str

gnssrefl.gps.**diffraction_correction**(*el_deg*, *temp=20.0*, *press=1013.25*)

Computes and return the elevation correction for refraction in the atmosphere such that the elevation of the satellite plus the correction is the observed angle of incidence.

Based on an empirical model by G.G. Bennet. This code was provided by Chalmers Group, Joakim Strandberg and Thomas Hobiger Bennett, G. G. The calculation of astronomical refraction in marine navigation. Journal of Navigation 35.02 (1982): 255-259.

Parameters

- **el_deg** (*array_like*) – A vector of true satellite elevations in degrees for which the correction is calculated.
- **temp** (*float, optional*) – Air temperature at ground level in degrees celsius, default 20 C.
- **press** (*float, optional*) – Air pressure at ground level in hPa, default 1013.25 hPa.

Returns
corr_el_deg – The elevation correction in degrees.

Return type
1d-array

gnssrefl.gps.**doy2ymd**(*year*, *doy*)

Parameters

- **year** (*int*) –
- **doy** (*int*) – day of year

Returns
d

Return type
datetime object

gnssrefl.gps.**elev_angle**(*up*, *RecSat*)

computes satellite elevation angle

Parameters

- **up** (*3 vector float*) – unit vector in the up direction
- **RecSat** (*3 vector numpy*) – Cartesian vector pointing from receiver to satellite in meters

Returns
angle – elevation angle in radians

Return type
float

gnssrefl.gps.**fdoy2mjd**(*year*, *fdoy*)

calculates modified julian day from year and fractional day of year

Parameters

- **year** (*int*) – full year
- **fdoy** (*float*) – fractional day of year

Returns

mjd – modified julian day

Return type

float

gnssrefl.gps.**final_gfz_orbits**(*year*, *month*, *day*)

downloads gfz final orbit and stores in $ORBITS

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month

Returns

- **littlename** (*str*) – orbit filename, fdir, foundit
- **fdir** (*str*) – directory where the orbit file is stored locally
- **foundit** (*bool*) – whether the file was found

gnssrefl.gps.**findConstell**(*cc*)

determine constellation integer value

Parameters

cc (*string is one character (from rinex satellite line)*) –

constellation definition:

G : GPS R : Glonass E : Galileo C : Beidou

Returns

out – value added to satellite number for our system, 0 for GPS, 100 for Glonass, 200 for Galileo, 300 for everything else

Return type

integer

gnssrefl.gps.**find_satlist_wdate**(*f*, *snrExist*, *year*, *doy*)

find satellite list for a given frequency and date

Parameters

- **f** (*integer*) – frequency
- **snrExist** (*numpy array, bool*) – tells you if a signal is (potentially) legal
- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **satlist** (*numpy list of integers*) – satellites to use

- **june 24, 2021** (*updated for SVN78*)

gnssrefl.gps.**freq_out**(*x*, *ofac*, *hifac*)

Parameters

- **x** (*numpy array*) – sine(elevation angle)
- **ofac** (*float*) – oversamping factor
- **hifac** (*float*) – how far to calculate RH frequencies (in meters)

Returns
pd – frequencies

Return type
float numpy arrays

gnssrefl.gps.**ga_highrate**(*station9*, *year*, *doy*, *dec*, *deleteOld=True*)

Attempts to download and merge highrate RINEX 3 files from GA

Parameters

- **station9** (*str*) – nine character station name appropriate for rinex 3
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **dec** (*int*) – decimation value. 1 or 0 means no decimation
- **deleteOld** (*bool*) – whether to delete old rinex 3 files

Returns

- **rinex2** (*string*) – rinex2 filename created by merging 96 files!
- **fexist** (*boolean*) – whether a rinex2 file was successfully created

gnssrefl.gps.**gbm_orbits_direct**(*year*, *month*, *day*)

downloads gfz multi-gnss orbits, aka gbm orbits, directly from GFZ. thus avoids CDDIS. it first checks to see if you have the files online. both version of the long name.

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month number or day of year if day is set to zero
- **day** (*int*) – calendar day of month

gnssrefl.gps.**geoidCorrection**(*lat*, *lon*)

Calculates the EGM96 geoid correction

Parameters

- **lat** (*float*) – latitude, degrees
- **lon** (*float*) – longitude, degrees

Returns
geoidC – geoid correction in meters

Return type
float

gnssrefl.gps.**getMJD**(*year*, *month*, *day*, *fract_hour*)

Parameters

- **year** (*int*) –
- **month** (*int*) –
- **day** (*int*) –
- **fract_hour** (*float*) – hour (fractional)

Returns
mjd – modified julian day

Return type
float

gnssrefl.gps.**get_cddis_navfile**(*navfile*, *cyyyy*, *cyy*, *cdoy*)

Tries to download navigation file from CDDIS Renames it to my convention (auto0010.22n)

Parameters

- **navfile** (*str*) – name of GPS broadcast orbit file
- **cyyyy** (*str*) – 4 char year
- **cyy** (*str*) – 2 char year
- **cdoy** (*str*) – 3 char day of year

Returns
navfile – full path of the stored navigation file

Return type
str

gnssrefl.gps.**get_esa_navfile**(*cyyyy*, *cdoy*)

downloads GPS broadcast navigation file from ESA tries both Z and gz compressed

Parameters

- **cyyyy** (*str*) – 4 char year
- **cdoy** (*str*) – 3 char day of year

Returns
fstatus – whether file was found or not

Return type
bool

gnssrefl.gps.**get_noaa_obstimes**(*t*)

Needs to be be fixed for new file structure

Parameters
t (*list of integers*) – year, month, day, hour, minute, second

Returns
obstimes – datetime format

Return type
list

gnssrefl.gps.**get_noaa_obstimes_plus**(*t*, ***kwargs*)

given a list of time tags (y,m,d,h,m,s), it calculates datetime objects and modified julian days

Parameters
t (*numpy array*) – our water level format where year, month, day, hour, minute, second are in the first columns

Returns

- **obstimes** (*list of datetime obj*) – list of timetags
- **modjulian** (*list of floats*) – modified julian date

gnssrefl.gps.**get_obstimes**(*tvd*)

Calculates datetime objects for times associated with LSP results file contents, i.e. the variable created when you read in the results file.

Parameters
tvd (*numpy array*) – results of LSP results

Returns
obstimes – datetime objects

Return type
numpy array

gnssrefl.gps.**get_obstimes_plus**(*tvd*)

send a LSP results file, so the variable created when you read in the results file. return obstimes for matplotlib plotting purposes 2022jun10 - added MJD output

See get_obstimes 2024apr06 too slow because I was using np.append

Parameters
tvd (*numpy array*) – contents of Lomb Scargle data processing

Returns

- **obstimes** (*list of datetime objects*) – times of observations
- **modjulian** (*list of floats*) – modified julian days

gnssrefl.gps.**get_ofac_hifac**(*elevAngles*, *cf*, *maxH*, *desiredPrec*)

Computes two factors - ofac and hifac - that are inputs to the Lomb-Scargle Periodogram code. We follow the terminology and discussion from Press et al. (1992) in their LSP algorithm description.

Parameters

- **elevAngles** (*numpy of floats*) – vector of satellite elevation angles in degrees
- **cf** (*float*) – (L-band wavelength/2) in meters
- **maxH** (*int*) – maximum LSP grid frequency in meters
- **desiredPrec** (*float*) – the LSP frequency grid spacing in meters i.e. how precise you want he LSP reflector height to be estimated

Returns

- **ofac** (*float*) – oversampling factor
- **hifac** (*float*) – high-frequency factor

gnssrefl.gps.**get_orbits_setexe**(*year*, *month*, *day*, *orbtype*)

picks up and stores orbits as needed.

why does this work if it does not include the hour for ultra?

modified 2025 jul 5 to use new GFZ website

Parameters

- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **orbtype** (*str*) – orbit source, e.g. nav, gps…

Returns

- **foundit** (*bool*) – whether orbit file was found
- **f** (*str*) – name of the orbit file
- **orbdir** (*str*) – location of the orbit file

gnssrefl.gps.**get_sopac_navfile**(*navfile*, *cyyyy*, *cyy*, *cdoy*)

downloads navigation file from SOPAC

Parameters

- **navfile** (*string*) – name of GPS broadcast orbit file
- **cyyyy** (*string*) – 4 char year
- **cyy** (*string*) – 2 char year
- **cdoy** (*string*) – 3 char day of year

Returns
navfile – should be the same name as input. not logical! I have no idea why i did it this way.

Return type
string

gnssrefl.gps.**get_sopac_navfile_cron**(*yyyy*, *doy*)

downloads navigation file from SOPAC to be used in a cron job

Parameters

- **yyyy** (*int*) – full year
- **doy** (*int*) – day of year

Returns
filefound – whether file is found

Return type
bool

gnssrefl.gps.**get_wuhan_orbits**(*year: int*, *month: int*, *day: int*, *hour: int*) → [<class 'str'>, <class 'str'>, <class 'bool'>]

Downloads ultra-rapid Wuhan sp3 file and stores them in $ORBITS

modified 2024 jul 8 to download files with NRT names

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year
- **day** (*int*) – day or if set to 0, then month is really day of year
- **hour** (*int*) – specific hour of ultrarapid orbit

Returns

- **unzipped_filename** (*str*) – name of the sp3 orbit file
- **orbit_dir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**getnavfile**(*year*, *month*, *day*)

picks up nav file and stores it in the ORBITS directory

Parameters

- **year** (*int*) – full year
- **month** (*int*) – if day is zero, the month value is really the day of year
- **day** (*int*) – day of the month

Returns

- **navname** (*str*) – name of navigation file
- **navdir** (*str*) – location of where the nav file should be stored
- **foundit** (*bool*) – whether the file was found

gnssrefl.gps.**getnavfile_archive**(*year*, *month*, *day*, *archive*)

picks up nav file from a specific archive and stores it in the ORBITS directory

Parameters

- **year** (*integer*) – full year
- **month** (*int*) – calendar month, or day of year
- **day** (*int*) – day of the month, or zero
- **archive** (*str*) – name of the GNSS archive. currently allow sopac and esa

Returns

- **navname** (*str*) – name of navigation file (should always be auto???0.yyn, so unclear to me why it is sent)
- **navdir** (*str*) – location of where the file has been stored
- **foundit** (*bool*) – whether the file was found

gnssrefl.gps.**getseries**(*site*)

originally from brendan crowell. picks up two UNR time series - stores in subdirectory called tseries input is station name (four character, lower case)

gnssrefl.gps.**getsp3file**(*year*, *month*, *day*)

retrieves IGS sp3 precise orbit file from CDDIS

Parameters

- **year** (*integer*) – full year

- **month** (*integer*) – calendar month
- **day** (*integer*) – calendar day

Returns

- **name** (*str*) – filename for the orbits
- **fdir** (*str*) – directory for the orbits

gnssrefl.gps.**getsp3file_flex**(*year*, *month*, *day*, *pCtr*)

retrieves sp3files only gets the old-style filenames

Parameters

- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **pCtr** (*str*) – 3 character orbit processing center, i.e. gfz

Returns

- **name** (*str*) – filename for the orbits
- **fdir** (*str*) – directory for the orbits
- **fexist** (*bool*) – whether the orbit file was successfully found

gnssrefl.gps.**getsp3file_mgex**(*year*, *month*, *day*, *pCtr*)

retrieves MGEX sp3 orbit files

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month number
- **day** (*int*) – day of month
- **pCtr** (*str*) – name of the orbit center, i.e. gfz

Returns

- **name** (*str*) – orbit filename
- **fdir** (*str*) – file directory
- **foundit** (*bool*) – whether orbit file was found

gnssrefl.gps.**gfz_version**()

Finds location of the gfzrnx executable

Returns
gfzv – name/location of gfzrnx executable

Return type
str

gnssrefl.gps.**greenland_rinex3**(*station*, *year*, *doy*, *stream*, *samplerate*)

downloads RINEX3 files from GNET. Returns hatanaka and gzipped file. Could uncompress and convert, but downstream code expects *crx.gz

It requires you have a GNET password. The first time you call this function it will ask for the username and password and it will store it locally.

You must have installed lftp on your own. (gnssrefl will not do it for you).

stream and samplerate parameters REQUIRED

Only allows one day files

Parameters

- **station** (*str*) – long station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **stream** (*str*) – stream ID
- **samplerate** (*int*) – seconds

Returns

- **filename** (*str*) – rinex3 filename(hatanaka compressed and gzipped)
- **found** (*bool*) – whether file was found

gnssrefl.gps.**hatanaka_version**()

Finds the Hatanaka decompression executable

Returns
hatanakav – name/location of hatanaka executable

Return type
str

gnssrefl.gps.**hatanaka_warning**()

Return type
warning about missing Hatanaka executable

gnssrefl.gps.**highrate_nz**(*station*, *year*, *month*, *day*)

NO LONGER SUPPORTED picks up a high-rate RINEX 2.11 file from GNS New zealand requires teqc to convert/merge the files

Parameters

- **station** (*str*) – station name
- **year** (*int*) – full year
- **month** (*int*) – month or day of year
- **day** (*int*) – day or zero

gnssrefl.gps.**ign_orbits**(*filename*, *directory*, *year*)

Downloads sp3 files from the IGN archive

Parameters

- **filename** (*str*) – name of the sp3 file
- **directory** (*str*) – location of orbits at the IGN
- **year** (*int*) – full year

Returns
foundit – whether sp3 file was found

Return type
bool

gnssrefl.gps.**ign_rinex3**(*station9ch*, *year*, *doy*, *srate*)

Downloads a RINEX 3 file from IGN

Parameters

- **station9ch** (*str*) – 9 character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **srate** (*int*) – sample rate

Returns
fexist – whether file was downloaded

Return type
bool

gnssrefl.gps.**igsname**(*year*, *month*, *day*)

returns the name of an IGS orbit file

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month number
- **day** (*int*) – calendar day number

Returns

- **name** (*str*) – IGS orbit file name
- **clockname** (*str*) – COD clockname

gnssrefl.gps.**inout**(*c3gz*)

Takes a Hatanaka rinex3 file that has been gzipped gunzips it and decompresses it

Parameters
c3gz (*string*) – name of a gzipped hatanaka compressed RINEX 3 filename

Returns

- **translated** (*boolean*) – whether file was successfully translated or not
- **rnx** (*string*) – filename of the uncompressed and de-Hatanaka'ed RINEX file

gnssrefl.gps.**kgpsweek**(*year*, *month*, *day*, *hour*, *minute*, *second*)

Calculates GPS week and GPS second of the week There is another version that works on character string. I think (kgpsweekC)

Examples

kgpsweek(2023,1,1,0,0,0)
returns 2243 and 0

Parameters

- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day

- **hour** (*int*) – hour of the Day (gps time)
- **minute** (*int*) – minutes
- **second** (*int*) – seconds

Returns

- **GPS_wk** (*int*) – GPS week
- **GPS_sec_wk** (*int*) – GPS second of the week

gnssrefl.gps.**kgpsweekC**(*z*)

converts RINEX timetag line into integers/float

Parameters
z (*str*) – timetag from rinex file (YY MM DD MM SS.SSSS)

Returns

- **gpsw** (*integer*) – GPS week
- **gpss** (*integer*) – GPS seconds

gnssrefl.gps.**l1c_list**(*year*, *doy*)

Creates a satellite list of L1C transmitting satellites for a given year/doy

L1C is only transmitted by GPS Block III satellites launched since December 2018.

Parameters

- **year** (*int*) – full year
- **doy** (*integer*) – day of year

Returns
l1csatlist – satellites possibly transmitting L1C signal

Return type
numpy array (int)

gnssrefl.gps.**l2c_l5_list**(*year*, *doy*)

Creates a satellite list of L2C and L5 transmitting satellites for a given year/doy

Parameters

- **year** (*int*) – full year
- **doy** (*integer*) – day of year

Returns

- **l2csatlist** (*numpy array (int)*) – satellites possibly transmittingL2C
- **l5satlist** (*numpy array (int)*) – satellites possibly transmitting L5

gnssrefl.gps.**llh2xyz**(*lat*, *lon*, *height*)

converts llh to Cartesian values

Parameters

- **lat** (*float*) – latitude in degrees
- **lon** (*float*) – longitude in degrees
- **height** (*float*) – ellipsoidal height in meters

Returns

- **x** (*float*) – X coordinate (m)
- **y** (*float*) – Y coordinate (m)
- **z** (*float*) – Z coordinate (m)
- **Ref** (*Decker, B. L., World Geodetic System 1984,*)
- *Defense Mapping Agency Aerospace Center.*
- *modified from matlab version kindly provided by CCAR*

gnssrefl.gps.**lsp_header**(*station*, ***kwargs*)

makes a header for a LSP result file that can be used with np.savetxt

Parameters
station (*str*) – name of station

Returns
all – multi-line string for LSP header file

Return type
str

gnssrefl.gps.**make_azim_choices**(*alist*)

not used yet

Parameters

- **alist** (*list of floats*) – azimuth pairs - must be even number of values, i.e. [amin1, amax1, amin2, amax2]
- **azval** (*list of floats*) – azimuth regions for lomb scargle periodograms

gnssrefl.gps.**make_nav_dirs**(*yyyy*)

input year and it makes sure output directories are created for orbits

Parameters
yyyy (*int*) – year

gnssrefl.gps.**make_snrdir**(*year*, *station*)

makes various directories needed for SNR file/analysis outputs

Parameters

- **year** (*int*) – full year
- **station** (*str*) – 4 ch station name

gnssrefl.gps.**mjd**(*y*, *m*, *d*, *hour*, *minute*, *second*)

calculate the integer part of MJD and the fractional part.

Parameters

- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **hour** (*int*) – hour of day
- **minute** (*int*) – minute of the day
- **second** (*int*) – second of the day

Returns

- **mjd** (*float*) – modified julian day of y-m-d
- **fracDay** (*float*) – fractional day
- **using information from http** (*//infohost.nmt.edu/~shipman/soft/sidereal/ims/web/MJD-fromDatetime.html*)

gnssrefl.gps.**mjd_more**(*mmjd*)

This is not working yet.

Parameters
mmjd (*float*) – mod julian date

Returns

- **year** (*int*) – full year
- **mm** (*int*) – month
- **dd** (*int*) – day
- **doy** (*int*) – day of year

gnssrefl.gps.**mjd_to_date**(*jd*)

https://gist.github.com/jiffyclub/1294443

Converts Modified Julian Day to y,m,d

Algorithm from Practical Astronomy with your Calculator or Spreadsheet
4th ed., Duffet-Smith and Zwart, 2011.

Parameters
jd (*float*) – Julian Day

Returns

- **year** (*int*) – Year as integer. Years preceding 1 A.D. should be 0 or negative. The year before 1 A.D. is 0, 10 B.C. is year -9.
- **month** (*int*) – Month as integer, Jan = 1, Feb. = 2, etc.
- **day** (*float*) – Day, may contain fractional part.

gnssrefl.gps.**mjd_to_datetime**(*mjd*)

Parameters
mjd (*float*) – modified julian date

Returns
dt

Return type
datetime object

gnssrefl.gps.**modjul_to_ydoy**(*MJD*)

yet another data translation function. when will it end? Modified Julian Day to Year, Doy

Parameter

MJD: float
modified julian day

returns

- **year** (*int*) – full year
- **doy** (*int*) – day of year

gnssrefl.gps.**month_converter**(*month*)

brendan gave this to me - give it a 3 char month, returns integer

gnssrefl.gps.**more_confused_obstimes**(*tvd*)

too slow

Parameters
tvd (*numpy array of floats*) – lsp results from a loadtxt command

Returns
modifiedjulian – mjd values

Return type
numpy array of floats

gnssrefl.gps.**myfavoritegpsobs**()

returns list of GPS only SNR obs needed for gfzrnx.

gnssrefl.gps.**myfavoriteobs**()

returns list of SNR obs needed for gfzrnx.

gnssrefl.gps.**myfindephem**(*week*, *sweek*, *ephem*, *prn*)

inputs are gps week, seconds of week # ephemerides and PRN number # returns the closest ephemeris block after the epoch # if one does not exist, returns the first one

gnssrefl.gps.**myreadnav**(*file*)

Reads a GPS nav message. Not currently used in gnssrefl. For historical purposes I am leaving it here.

Parameters

- **file** (*str*) – nav filename
- **blocks** (*output is complicated - broadcast ephemeris*) –

gnssrefl.gps.**myscan**(*rinexfile*)

stripping the header code came from pyrinex. data are stored into a variable called table columns 0,1,2 are PRN, GPS week, GPS seconds, and observables rows are the different observations. these should be stored properly - this is a kluge

gnssrefl.gps.**nav_name**(*year*, *month*, *day*)

returns the name and location of the navigation file

Parameters

- **year** (*integer*) –
- **month** (*integer*) –
- **day** (*integer*) –

Returns

- **navfilename** (*str*) – name of the navigation file
- **navfiledir** (*str*) – local directory where navigation file will be stored

gnssrefl.gps.**navfile_retrieve**(*navfile*, *cyyyy*, *cyy*, *cdoy*)

retrieves navfile from either SOPAC or CDDIS

Parameters

- **navfile** (*str*) – name of the broadcast orbit file
- **cyyyy** (*str*) – 4 character yaer
- **cyy** (*string*) – 2 character year
- **cdoy** (*str*) – 3 character day of year

Returns
FileExists – whether the file was found

Return type
bool

gnssrefl.gps.**new_rinex3_rinex2**(*r3_filename*, *r2_filename*, *dec*, *gpsonly*, *log*, ***kwargs*)

This code translates a RINEX 3 file into a RINEX 2.11 file. It is assumed that the gfzrnx exists and that the RINEX 3 file is Hatanaka uncompressed or compressed. (ending in rnx or crx)

Parameters

- **r3_filename** (*str*) – RINEX 3 format filename. Either Hatanaka compressed or uncompressed allowed, but not if it is gzipped. The file extensions are crx or rnx.
- **r2_filename** (*str*) – RINEX 2.11 file
- **dec** (*integer*) – decimation factor. If 0 or 1, no decimation is done.
- **gpsonly** (*bool*) – whether you want only GPS signals. Default is false
- **log** (*fileiD*) – this must have been defined before you call this code

Returns
fexists – whether the RINEX 2.11 file was created and exists

Return type
bool

gnssrefl.gps.**new_ultra_gfz_orbits**(*year*, *month*, *day*, *hour*)

downloads gfz ultra rapid orbit and stores in $ORBITS locally this is for the two day version with long file names that became the only access point in early June 2024

to use this code properly for day N you should call it for day N-1 need because GFZ creates a two day file, and the file is named for day N-1

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month
- **hour** (*int*) – hour for ultrarapid

Returns

- **littlename** (*str*) – name of the orbit file

- **fdir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**newish_gfz_orbits**(*year*, *month*, *day*, *orbtype*)

downloads "newish" gfz final and rapid orbits and stores in $ORBITS locally Uses isdcftp instead original GFZ ftp site final are 15 minute files and rapid are 5 minute uses different name than at CDDIS?

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month
- **orbtype** (*str*) – kind of orbit, rapid or final

Returns

- **littlename** (*str*) – name of the orbit file
- **fdir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**nextdoy**(*year*, *doy*)

given a year/doy returns the subsequent year/doy

Parameters

- **year** (*int*) – day of year
- **doy** (*int*) – day of year

Returns

- **nyear** (*int*) – next year
- **ndoy** (*int*) – next day of year

gnssrefl.gps.**nicerTime**(*UTCtime*)

Converts fractional time (hours) to HH:MM

This only works properly for positive numbers. Took out the screen warning.

Parameters
UTCtime (*float*) – fractional hours of the day

Returns
T – output as HH:MM

Return type
str

gnssrefl.gps.**noaa2me**(*date1*)

converts NOAA type of date string to simple integers

Parameters
date1 (*string*) – time in format YYYYMMDD for year month and day

Returns

- **year1** (*integer*) – full year
- **month1** (*integer*) – month

- **day1** (*integer*) – day of the month
- **doy** (*integer*) – day of year
- **modjulday** (*float*) – modified julian date

gnssrefl.gps.**noaatime_to_obstime**(*noaa*)

stitching together something to convert string with year, month, day and returning obstime

Parameters
noaa (*str*) – year, month, day (e.g. 20220115)

Returns
obstime – time in datetime format

Return type
datetime

gnssrefl.gps.**norm**(*vect*)

calculates magnitude of a vector

Parameters
vect (*float*) – vector

Returns
nv – norm of vect

Return type
float

gnssrefl.gps.**one_gfz_archive_to_rule_them_all**(*year*, *month*, *day*, *orbtype*, *hour*)

access to latest GFZ archive structure for orbits

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month
- **orbtype** (*str*) – kind of orbit: rapid, final, ultra
- **hour** (*int*) – hour of orbit, only used for ultra

Returns

- **filename** (*str*) – name of the orbit file
- **fdir** (*str*) – name of the file directory where orbit is stored by gnssrefl
- **foundit** (*bool*) – whether orbit file was found

gnssrefl.gps.**open_outputfile**(*station*, *year*, *doy*, *extension*)

opens an output file in $REFL_CODE/year/results/station/extension directory for Lomb Scargle periodogram results. This really should go in a gnssir library.

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **extension** (*str*) – analysis extension name (for storage of results)

Returns

fileID – I don't know the proper name of this - but what comes out when you open a file so you can keep writing to it

Return type

?

gnssrefl.gps.**open_plot**(*plt_screen*)

is this used?

simple code to open a figure, called by gnssIR_lomb

gnssrefl.gps.**orbfile_cddis**(*name*, *year*, *secure_file*, *secure_dir*, *file2*)

tries to download a file from a directory at CDDIS which it then stores it the year directory (with a given name)

Parameters

- **name** (*str*) – the name of the orbit file you want to download from CDDIS
- **year** (*int*) – full year
- **secure_file** (*str*) – name of the file at CDDIS
- **secure_dir** (*str*) – where the file lives at CDDIS
- **file2** (*str*) – name without the compression???

Returns

- **foundit** (*bool*) – whether the file was found
- *now checks that file size is not zero. allows old file name downloads*

gnssrefl.gps.**prevdoy**(*year*, *doy*)

Given year and doy, return previous year and doy

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **pyear** (*int*) – previous year
- **pdoy** (*int*) – previous day of year

gnssrefl.gps.**print_file_stats**(*ele*, *sat*, *s1*, *s2*, *s5*, *s6*, *s7*, *s8*, *e1*, *e2*)

inputs

ele

sat

s1

s2

gnssrefl.gps.**print_version_to_screen**()

what it sounds like

gnssrefl.gps.**propagate**(*week*, *sec_of_week*, *ephem*)

inputs are GPS week, seconds of the week, and the appropriate ephemeris block from the navigation message returns the x,y,z, coordinates of the satellite and relativity correction (also in meters), so you add, not subtract

gnssrefl.gps.**queryUNR_modern**(*station*)

Queries the UNR database for station coordinates that has been stored in sql. downloads the sql file and stores it locally if necessary

Also queries a local file if you have defined one. It should be located in $REFL_CODE/input/llh_local.txt four columns should be 4 character station name lat lon height No commas between them and the station name should be four characters Hopefully lowercase, but I will try to put in a toggle that allows uppercase

gnssrefl.gps.**query_coordinate_file**(*station*)

Returns a priori latitude, longitude, and ellipsoidal height from a local file The file should be stored in $REFL_CODE/input/llh_local.txt It has a simple structure. Each value is separated by spaces

station latitude longitude height

The station name is four characters long and the units of the other three are degrees, degrees, and meters. Height is the ellipsoidal height. Comments are allowed in this file using a percent sign. If you use more or less than four columns per line the code will crash.

Parameters
station (*str*) – 4 character station name. checks both upper and lower case

Returns

- **foundit** (*bool*) – whether you found the coordinates
- **lat** (*float*) – latitude in degrees (zero if not found)
- **lon** (*float*) – longitude in degrees (zero if not found)
- **ht** (*float*) – ellipsoidal ht in meters (zero if not found)

gnssrefl.gps.**quick_plot**(*plt_screen*, *gj*, *station*, *pltname*, *f*)

inputs plt_screen variable (1 means go ahead) and integer variable gj which if > 0 there is something to plot also station name for the title pltname is png filename, if requested

gnssrefl.gps.**quickazel**(*gweek*, *gpss*, *sat*, *recv*, *ephemdata*, *localup*, *East*, *North*)

assumes you have read in the broadcast ephemeris, know where your receiver is and the time (gps week, second of week)

gnssrefl.gps.**quickp**(*station*, *t*, *sealevel*)

makes a quick plot of sea level prints the plot to the screen - it does not save it.

Parameters

- **station** (*str*) – station name
- **t** (*numpy array in datetime format*) – time of the sea level observations UTC
- **sealevel** (*list of floats*) – meters (unknown datum)

gnssrefl.gps.**random**() → x in the interval [0, 1).

gnssrefl.gps.**randomfilename**()

makes a string -length 9 - using random number generator. useful for filenames

Returns
rname – filename with nine random numerical characters

Return type
str

gnssrefl.gps.**rapid_gfz_orbits**(*year*, *month*, *day*)

downloads gfz rapid orbit and stores in $ORBITS locally

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year if day is set to zero
- **day** (*int*) – day of month

Returns

- **littlename** (*str*) – name of the orbit file
- **fdir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**read_files**(*year*, *month*, *day*, *station*)

who wrote this and who is it for? I do not believe it is for reflectometry and should be removed.

gnssrefl.gps.**read_leapsecond_file**(*mjd*)

reads leap second file and tries to figure out the UTC-GPS time offset needed for NMEA file users for the given MJD value

It will download and store the leap second file in REFL_CODE/Files if you don't already have it.

Parameters
mjd (*float*) – Modified Julian Day for when you want to know the leap seconds since GPS began

Returns
offset – UTC-GPS time offset in seconds. This should be added to UTC to get GPS

Return type
int

gnssrefl.gps.**read_simon_williams**(*filename*, *outfilename*)

Reads a PSMSL file and creates a new file in the standard format I use for tide gauge data in gnssrefl

Parameters

- **filename** (*str*) – datafile of GNSS based water level measurements from the archive at PSMSL created by Simon Williams
- **outfilename** (*str*) – where the rewritten data will go

Returns

- **outobstimes** (*datetime array*) – time of observations
- **outmjd** (*numpy array of floats*) – modified julian day
- **outsealevel** (*numpy array of floats*) – sea level, meters
- **prn** (*numpy array of integers*) – satellite numbers
- **fr** (*numpy array of integers*) – frequency
- **az** (*numpy array of floats*) – azimuth (degrees)

gnssrefl.gps.**read_sp3**(*file*)

borrowed from Ryan Hardy, who got it from David Wiese … can't really say more about it

gnssrefl.gps.**read_sp3file**(*file_path*)

input: file_path is the sp3file name this code is from Joakim Strandberg I believe. It is for the python only version of the translator, which should be deprecated

Returns

- **sp3** (*ndarray*)
- *colums are satnum, gpsweek, gps_sow, x,y,z*
- *x,y,z are in meters*
- *satnum has 0, 100, 200, 300 added for gps, glonass, galileo,beidou,*
- *respectively. all other satellites are ignored*

gnssrefl.gps.**removeDC**(*dat*, *satNu*, *sat*, *ele*, *pele*, *azi*, *az1*, *az2*, *edot*, *seconds*)

remove direct signal using given elevation angle (pele) and azimuth (az1,az2) constraints, return x,y as primary used data and windowed azimuth, time, and edot removed zero points, which 10^0 have value 1. used 5 to be sure?

Parameters

- **dat** (*numpy array of floats*) – SNR data
- **satNu** (*float*) – requested satellite number
- **sat** (*numpy array of floats*) – satellite numbers
- **ele** (*numpy array of floats*) – elevation angles, deg
- **pele** (*list of floats*) – min and max elevation angles (deg)
- **azi** (*numpy array of floats*) – azimuth angle, deg
- **az1** (*float*) – minimum azimuth angle (deg)
- **az2** (*float*) – maximum azimuth angle (deg)
- **edot** (*numpy array of floats*) – derivative elevation angle (deg/sec)
- **seconds** (*numpy array of floatas*) – seconds of the day

Returns

- **x** (*numpy array of floats*) – sine of elevation angle (i believed)
- **y** (*numpy array of floats*) – SNR data in lineer units with DC component removed
- **sat** (*??*) – not sure why this is sent and returned
- **azi** (*numpy array of flaots*) – azimuth angles
- **seconds** (*numpy array of flaots*) – seconds of the day
- **edot** (*numpy array of floats*) – derivative of elevation angle

gnssrefl.gps.**replace_wget**(*url*, *f*, ***kwargs*)

use requests instead of wget to download files this cannot be used for ftp addresses.

added optional input parameter timeout

Parameters

- **url** (*str*) – full path to file
- **f** (*str*) – filename
- **timeout** (*int*) – optional timeout parameter, seconds

Returns
success – whether file was found or not

Return type
bool

gnssrefl.gps.**result_directories**(*station*, *year*, *extension*)

Creates directories for results

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **extension** (*str*) – subdirectory for results (used for analysis strategy)

gnssrefl.gps.**rinex3_nav**(*year*, *month*, *day*)

not sure what this does!

gnssrefl.gps.**rinex_ga_highrate**(*station*, *year*, *month*, *day*)

no longer supported -

Parameters

- **station** (*str*) – 4 character station ID, lowercase
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*integer*) – day of the month

gnssrefl.gps.**rinex_jp**(*station*, *year*, *month*, *day*)

Picks up RINEX file from Japanese GSI GeoNet archive URL : https://www.gsi.go.jp/ENGLISH/index.html

Parameters

- **station** (*str*) – station name
- **year** (*int*) – full year
- **month** (*int*) – month or day of year
- **day** (*int*) – day of month or zero

gnssrefl.gps.**rinex_name**(*station*, *year*, *month*, *day*)

Defines rinex 2.11 file name

Parameters

- **station** (*str*) – 4 character station ID
- **year** (*int*) – full year
- **month** (*int*) –
- **day** (*int*) –

Returns

- **fnameo** (*str*) – RINEX 2.11 name
- *fnamed* – RINEX 2.11 name, Hatanaka compressed

gnssrefl.gps.**rinex_nrcan_highrate**(*station*, *year*, *month*, *day*)

picks up 1-Hz RINEX 2.11 files from NRCAN requires gfzrnx or teqc to merge the 15 minute files

Parameters

- **station** (*string*) – 4 character station name
- **year** (*integer*) – year
- **month** (*integer*) – month or day of year if day is set to zero
- **day** (*integer*) – day

gnssrefl.gps.**rinex_unavco**(*station*, *year*, *month*, *day*)

This is being used by the vegetation code picks up a RINEX 2.11 file from unavco low-rate area requires Hatanaka code

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*integer*) – full year
- **month** (*integer*) – month or day of year
- **day** (*integer*) – day of month or zero

gnssrefl.gps.**rinex_unavco_highrate**(*station*, *year*, *month*, *day*)

picks up a 1-Hz RINEX 2.11 file from unavco.

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **month** (*int*) – month number
- **day** (*int*) – calendar day number

gnssrefl.gps.**rot3**(*vector*, *angle*)

Parameters

- **vector** (*3 vector*) – float
- **angle** (*float*) – radians

Returns

vector2 – float, original vector rotated by angle

Return type

3 vector

gnssrefl.gps.**save_plot**(*plotname*)

save plot and send location info to the screen

Parameters

plotname (*str*) – name of output figure file

gnssrefl.gps.**set_subdir**(*subdir*)

make sure that subdirectory exists for output files should return the directory name ...

Parameters

subdir (*str*) – subdirectory in $REFL_CODE/Files

gnssrefl.gps.**simpleTime**(*mjd*)

Parameters
mjd (*float*) – modified julian day

Returns

- **year** (*int*) – full year
- **month** (*int*) – calendar year
- **day** (*int*) – calendar day
- **hour** (*int*) – hour of the day
- **minute** (*int*) – minute
- **second** (*int*) – second of the day
- **doy** (*int*) – day of year

gnssrefl.gps.**snr_exist**(*station*, *year*, *doy*, *snrEnd*)

check to see if the SNR file already exists uncompresses if necessary (gz or xz)

Parameters

- **station** (*str*) – four character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snrEnd** (*str*) – 2 character snr type, i.e. 66, 99

Returns
snre – whether SNR file exists.

Return type
boolean

gnssrefl.gps.**snr_name**(*station*, *year*, *month*, *day*, *option*)

Defines SNR filename

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) –
- **month** (*int*) –
- **day** (*int*) –
- **option** (*int*) – snr filename delimiter, i.e. 66

Returns
fname – snr filename

Return type
string

gnssrefl.gps.**sp3_name**(*year*, *month*, *day*, *pCtr*)

defines old-style sp3 name

Parameters

- **year** (*int*) –

- **month** (*int*) –
- **day** (*int*) –
- **pCtr** (*str*) – Orbit processing center

Returns

- **sp3name** (*str*) – old-style (lowercase) IGS name for sp3 file
- **sp3dir** (*str*) – where file is stored locally

gnssrefl.gps.**store_orbitfile**(*filename*, *year*, *orbtype*)

Stores orbit files locally

Parameters

- **filename** (*str*) – orbit filename
- **year** (*int*) – full year
- **orbtype** (*str*) – kind of orbit (nav or sp3)

Returns

xdir – local directory where the orbit belongs

Return type

str

gnssrefl.gps.**store_snrfile**(*filename*, *year*, *station*)

move an snr file to the right place

Parameters

- **filename** (*str*) – name of SNR file
- **year** (*int*) – full year
- **station** (*str*) – 4 ch station name

gnssrefl.gps.**strip_compute**(*x*, *y*, *cf*, *maxH*, *desiredP*, *minH*, *lsp_method='fast'*)

strips snr data

Parameters

- **x** (*numpy array*) – elevation angles in degrees
- **y** (*numpy array*) – SNR data
- **cf** (*float*) – scale factor for given frequency
- **maxH** (*float*) – maximum reflector height in meters
- **desiredP** (*float*) – precision of Lomb Scargle in meters
- **minH** (*float*) – minimum reflector height in meters
- **lsp_method** (*str*) – 'fast' for AstroPy NFFT (default), 'scipy' for original SciPy

Returns

- **maxF** (*float*) – maximum Reflector height (meters)
- **maxAmp** (*float*) – amplitude of periodogram
- **eminObs** (*float*) – minimum observed elevation angle in degrees
- **emaxObs** (*float*) – maximum observed elevation angle in degrees

- **riseSet** (*integer*) – 1 for rise and -1 for set
- **px** (*numpy array*) – periodogram, x-axis, RH, meters
- **pz** (*numpy array*) – periodogram, y-axis, volts/volts

gnssrefl.gps.**teqc_version**()

Finds location of the teqc executable

Returns
gpse – location of teqc executable

Return type
str

gnssrefl.gps.**translate_dates**(*year*, *month*, *day*)

i do not think this is used

gnssrefl.gps.**trignet**(*station*, *year*, *doy*)

tries to pick up 30 sec RINEX 2.11 data from TRIGNET

Parameters

- **station** (*str*) – station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns
fexist – whether you succeeded

Return type
bool

gnssrefl.gps.**ultra_gfz_orbits**(*year*, *month*, *day*, *hour*)

downloads rapid GFZ sp3 file and stores them in $ORBITS is this correct? or is the regular file?

started to changed file directory source and name in may 2024 right now they are segregated

I am not sure this works anymore. You should use new_ultra instead

Parameters

- **year** (*int*) – full year
- **month** (*int*) – month or day of year
- **day** (*int*) – day or if set to 0, then month is really day of year
- **hour** (*int*) – hour of the day

Returns

- **littlename** (*str*) – name of the orbit file
- **fdir** (*str*) – name of the file directory where orbit is stored
- **foundit** (*bool*) – whether file was found

gnssrefl.gps.**unr_database**(*file1*, *file2*, *database_file*)

Checks to see if the database lives in either file1 or file2 locations. Stem of the filename is third input, database_file

Parameters

- **file1** (*str*) – full name of database in $REFL_CODE/Files

- **file2** (*str*) – full name of database in local gnssrefl directory
- **database_file** (*str*) – database file name

Returns

- **exists_now** (*bool*) – whether you were successful in finding the database
- **database_location** (*str*) – full name of the database you found and want to use going on

gnssrefl.gps.**up**(*lat*, *lon*)

returns the up unit vector, and local east and north unit vectors needed for azimuth calc.

Parameters

- **latitude** (*float*) – radians
- **longitude** (*float*) – radians

Returns

- **East** (*three vector*) – local transformation unit vector
- **North** (*three vector*) – local transformation unit vector

gnssrefl.gps.**update_plot**(*plt_screen*, *x*, *y*, *px*, *pz*)

is this used?

plt_screen
[int] 1 means update the plot

x
[numpy array of floats] elevation angles (deg)

y
[numpy array of floats] SNR data, volts/volts

px
[numpy array of floats] periodogram, x-axis (meters)

pz
[numpy array of floats] periodogram, y-axis

gnssrefl.gps.**update_quick_plot**(*station*, *f*)

updates plot in quickLook

Parameters

- **station** (*str*) – 4 ch name
- **f** (*int*) – frequency

class gnssrefl.gps.**wgs84**

Bases: object

wgs84 parameters for Earth radius and flattening

a = 6378137.0

e = np.float64(0.08181919084262149)

f = 0.0033528106647474805

gnssrefl.gps.**what_is_today**()

figures out year and doy for the current date on your computer

Returns

- *year*
- *doy*

gnssrefl.gps.**window_data**(*s1*, *s2*, *s5*, *s6*, *s7*, *s8*, *sat*, *ele*, *azi*, *seconds*, *edot*, *f*, *az1*, *az2*, *e1*, *e2*, *satNu*, *pfitV*, *pele*, *screenstats*)

window the SNR data for a given satellite azimuth and elevation angle range

also calculates the scale factor for various GNSS frequencies. currently returns meanTime in UTC hours and mean azimuth in degrees cf, which is the wavelength/2 currently works for GPS, GLONASS, GALILEO, and Beidou new: pele are the elevation angle limits for the polynomial fit. these are appplied before you start windowing the data

Parameters

- **s1** (*numpy array*) – SNR L1 data, floats
- **s2** (*numpy array*) – SNR L2 data, floats
- **s5** (*numpy array*) – SNR L5 data
- **s6** (*numpy array floats*) – SNR L6 data
- **s7** (*numpy array floats*) – SNR L7 data
- **s8** (*numpy array floats*) – SNR L8 data
- **sat** (*numpy array*) – satellite number
- **ele** (*numpy array*) – elevation angle (Degrees)
- **azi** (*numpy array*) – azimuth angle (Degrees)
- **seconds** (*numpy array*) – seconds of the day (GPS time)
- **edot** (*numpy array*) – elev angle time rate of change (units?)
- **f** (*int*) – requested frequency
- **az1** (*float*) – minimum azimuth limit, degrees
- **az2** (*float*) – maximum azimuth limit, degrees
- **e1** (*float*) – minimum elevation angle limit, degrees
- **e2** (*float*) – maximum elevation angle limit, degrees
- **satNu** (*int*) – requested satellite number
- **pfitV** (*int*) – polynomial order for DC fit
- **screenstats** (*bool*) – Whether statistics come to the screen

Returns

- **x** (*numpy array of floats*) – elevation angle, deg
- **y** (*numpy array of floats*) – SNR, db-Hz
- **Nvv** (*int*) – number of points in x
- **cf** (*float*) – refl scale factor (lambda/2)
- **meanTime** (*float*) – UTC hour of the arc

- **avgAzim** (*float*) – average azimuth of the track (degrees)
- **outFact1** (*float*) – tan(elev)/elevdot, hours, from SNR file
- **outFact2** (*float*) – tan(elev)/elevdot, hours, from linear fit
- **delT** (*float*) – track length in minutes

gnssrefl.gps.**write_QC_fails**(*delT*, *delTmax*, *eminObs*, *emaxObs*, *e1*, *e2*, *ediff*, *maxAmp*, *Noise*, *PkNoise*, *reqamp*, *tooclose2edge*, *fileid*)

prints out various QC fails to the screen

Parameters

- **delT** (*float*) – how long the satellite arc is (minutes)
- **delTmax** (*float*) – max satellite arc allowed (minutes)
- **eminObs** (*float*) – minimum observed elev angle (deg)
- **emaxObs** (*float*) – maximum observed elev angle (deg)
- **e1** (*float*) – minimum allowed elev angle (deg)
- **e2** (*float*) – maximum allowed elev angle (deg)
- **ediff** (*float*) – allowed min/max elevation diff from obs min/max elev angle (deg)
- **maxAmp** (*float*) – measured peak LSP
- **Noise** (*float*) – measured noise value for the periodogram
- **PkNoise** (*float*) – required peak to noise
- **reqamp** (*float*) – require peak LSP
- **tooclose2edge** (*bool*) – wehther peak value is too close to begining or ending of the RH constraints
- **fileid** – file identifier

gnssrefl.gps.**xyz2llh**(*xyz*, *tol*)

Computes latitude, longitude and height from XYZ (meters)

Parameters

- **xyz** (*list or np array*) – X,Y,Z in meters
- **tol** (*float*) – tolerance in meters for the calculation (1E-8 is good enough)

Returns

- **lat** (*float*) – latitude in radians
- **lon** (*float*) – longitude in radians
- **h** (*float*) – ellipsoidal height in WGS84 in meters

gnssrefl.gps.**xyz2llhd**(*xyz*)

Converts cartesian coordinates to latitude,longitude,height

Parameters

xyz (*three vector of floats*) – Cartesian position in meters

Returns

- **lat** (*float*) – latitude in degrees
- **lon** (*float*) – longitude in degrees

- **h** (*float*) – ellipsoidal height in WGS84 in meters

gnssrefl.gps.**ydoy2datetime**(*y*, *doy*)

translates year/day of year numpy array into datetimes for plotting

this was running slow for large datasets. changing to use lists and then asarray to numpy

Parameters

- **y** (*numpy array of floats*) – full year
- **doy** (*numpy array of floats*) – day of year

Returns
bigT – datetime objects

Return type
numpy array

gnssrefl.gps.**ydoy2mjd**(*year*, *doy*)

calculates modified julian day from year and day of year

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns
mjd – modified julian day

Return type
float

gnssrefl.gps.**ydoy2useful**(*year*, *doy*)

Calculates various dates

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*integer*) – calendar day
- **cyyyy** (*str*) – four character year
- **cdoy** (*str*) – three character day of year
- **YMD** (*str*) – date as in '19-12-01' for December 1, 2019

gnssrefl.gps.**ydoy2ymd**(*year*, *doy*)

inputs: year and day of year (doy) returns: year, month, day

gnssrefl.gps.**ydoych**(*year*, *doy*)

Converts year and doy to various character strings (two char year, 4 char year, 3 char day of year)

Parameters

- **year** (*int*) – full year

- **doy** (*int*) – day of year

Returns

- **cyyyy** (*str*) – 4 character year
- **cyy** (*str*) – 2 character year
- **cdoy** (*str*) – 3 character day of year

gnssrefl.gps.**ymd2ch**(*year*, *month*, *day*)

returns doy and character versions of year,month,day if day is zero, it assumes doy is in the month input

Parameters

- **year** (*int*) – full year
- **month** (*int*) – if day is zero this is day of yaer
- **day** (*int*) – day of month or zero

Returns

- **month** (*int*) – numerical month of the year
- **day** (*int*) – day of the month
- **doy** (*int*) – day of year
- **cyyyy** (*str*) – 4 ch year
- **cyy** (*str*) – 2 ch year
- **cdoy** (*str*) – 4 ch year

gnssrefl.gps.**ymd2doy**(*year*, *month*, *day*)

Calculates day of year and other date strings

Parameters

- **year** (*integer*) – full year
- **month** (*integer*) – month
- **day** (*integer*) – day of the month

Returns

- **doy** (*int*) – day of year
- **cdoy** (*str*) – three character day of year
- **cyyyy** (*str*) – four character year
- **cyy** (*str*) – two character year

gnssrefl.gps.**ymd_hhmmss**(*year*, *doy*, *utc*, *dtime*)

translates year, day of year and UTC hours into various other time parameters

this code should really be removed. all time translations should be wrt to MJD.

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – full year
- **UTC** (*float*) – fractional hours
- **dtime** (*bool*) – whether you want datetime object

Returns

- **bigT** (*datetime object*)
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **hour** (*int*) – hour of the day
- **minute** (*int*) – minutes of the day
- **second** (*int*) – seconds

gnssrefl.gps.**zenithdelay**(*h*)

a very simple zenith troposphere delay in meters this is NOT to be used for precise geodetic applications

Parameters
h (*float*) – ellipsoidal (height) in meters

Returns
zd – simple zenith delay for the troposphere in meters

Return type
float

gnssrefl.gpsweek module

gnssrefl.gpsweek.**main**()

Calculates GPS week information and prints it to the screen

Parameters

- **year** (*integer*) – full year
- **month** (*integer*) – calendar month
- **day** (*integer*) – day of the month

Returns

- **wk** (*int*) – GPS week
- **dayofthewk** (*int*) – day of the week (from 0-6) used by the IGS and others for orbit filenames

gnssrefl.highrate module

gnssrefl.highrate.**bkg_highrate**(*station*, *year*, *month*, *day*, *stream*, *dec_rate*, *bkg*, ***kwargs*)

picks up a highrate RINEX 3 file from BKG, merges and decimates it. requires gfzrnx

Parameters

- **station** (*str*) – 9 ch station name
- **year** (*int*) – full year
- **month** (*integer*) – month or day of year if day set to 0
- **day** (*int*) – day of the month
- **stream** (*str*) – R or S

- **dec_rate** (*int*) – decimation rate in seconds
- **bkg** (*str*) – file directory at BKG (igs or euref)

Returns

- **file_name24** (*str*) – name of merged rinex file
- **fexist** (*boolean*) – whether file exists

gnssrefl.highrate.**bkg_highrate_tar**(*station*, *year*, *month*, *day*, *stream*, *dec_rate*, *bkg*)

picks up a highrate RINEX 3 file from BKG, merges and decimates it. requires gfzrnx

Parameters

- **station** (*str*) – 9 ch station name
- **year** (*int*) – full year
- **month** (*integer*) – month or day of year if day set to 0
- **day** (*int*) – day of the month
- **stream** (*str*) – R or S
- **dec_rate** (*int*) – decimation rate in seconds
- **bkg** (*str*) – file directory at BKG (igs or euref)

Returns

- **file_name24** (*str*) – name of merged rinex file
- **fexist** (*boolean*) – whether file exists

gnssrefl.highrate.**cddis_highrate**(*station*, *year*, *month*, *day*, *stream*, *dec_rate*)

picks up highrate RINEX files from CDDIS and merges them

Parameters

- **station** (*str*) – 4 char or 9 char station name Rinex 2.11 for the first and rinex 3 for the latter
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **stream** (*str*) – rinex3 ID, S or R
- **dec_rate** (*int*) – decimation rate, seconds

Returns

- **rinexname** (*str*) – name of the merged/uncompressed outputfile
- **fexist** (*bool*) – whether the Rinex file was successfully retrieved
- *requires hatanaka code and gfzrnx*

gnssrefl.highrate.**cddis_highrate_tar**(*station*, *year*, *month*, *day*, *stream*, *dec_rate*)

picks up a tar'ed highrate RINEX files from CDDIS, untars it, and merges.

added a try so it doesn't die if the file does not exist

Parameters

- **station** (*str*) – 4 char or 9 char station name Rinex 2.11 for the first and rinex 3 for the latter
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **stream** (*str*) – rinex3 ID, S or R
- **dec_rate** (*int*) – decimation rate, seconds

Returns

- **rinexname** (*str*) – name of the merged/uncompressed outputfile
- **fexist** (*bool*) – whether the Rinex file was successfully retrieved
- *requires hatanaka code and gfzrnx*

gnssrefl.highrate.**esp_highrate**(*station*, *year*, *month*, *day*, *stream*, *dec_rate*)

picks up a highrate RINEX 3 file from Spanish Geodeic Center, merges and decimates it. requires gfzrnx

Parameters

- **inputs** (*string*) – 9 ch station name
- **year** (*integer*) – full year
- **month** (*integer*) – month or day of year if day set to 0
- **day** (*integer*) – day of the month
- **stream** (*str*) – R or S
- **dec_rate** (*integer*) – decimation rate in seconds

Returns

- **file_name24** (*str*) – name of merged rinex file
- **fexist** (*boolean*) – whether file exists

gnssrefl.highrate.**kadaster_highrate**(*station*, *year*, *doy*, *stream*, *dec_rate*)

picks up a highrate RINEX 3 file from Dutch archive, merges and decimates it. requires gfzrnx. Someone should add regular 30 second downloads - but it is not gonna be me.

Parameters

- **station** (*str*) – 9 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **stream** (*str*) – R or S
- **dec_rate** (*int*) – decimation rate in seconds

Returns

- **file_name24** (*str*) – name of merged rinex file
- **fexist** (*boolean*) – whether the new merged file exists

gnssrefl.highrate.**variableArchives**(*station*, *year*, *doy*, *cyyyy*, *cyy*, *cdoy*, *chh*, *cmm*)

creates rinex3 compliant file names and finds executables needed to manipulate those files

Parameters

- **station** (*str*) – 9 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **cyyyy** (*str*) – 4 ch year
- **cyy** (*str*) – two ch year
- **cdoy** (*str*) – 3 ch day of year
- **chh** (*str*) – 2 ch hour
- **cmm** (*str*) – 2 ch minutes

Returns

- **file_name** (*str*) – first filename to look for
- **crnx_name** (*str*) – first hatanaka name
- **file_name2** (*str*) – second filename to look for
- **crnx_name2** (*str*) – second hatanaka compressed name
- **exe1** (*str*) – uncompression executable to use for file_name
- **exe2** (*str*) – uncompression executable to use for file_name2

gnssrefl.installexe_cl module

gnssrefl.installexe_cl.**checkexist**(*exe*)

check to see if an executable exists

Parameters

exe (*str*) – executable name to check

gnssrefl.installexe_cl.**download_chmod_move**(*url*, *savename*, *exedir*)

download an executable, chmod it, and store it locally

Parameters

- **url** (*str*) – external location of the executable
- **savename** (*str*) – name of the executable
- **exedir** (*str*) – name of local executable directory (EXE environment variable)

gnssrefl.installexe_cl.**installexe**(*opsys: str*)

Command line interface to install non-python executables, specifically CRX2RNX and gfzrnx.

https://stackoverflow.com/questions/12791997/how-do-you-do-a-simple-chmod-x-from-within-python

Parameters

opsys (*str*) – operating system. Allowed values are linux64, macos, and mac-newchip PC users should use the docker, where these executables come pre-installed

gnssrefl.installexe_cl.**main**()

command line code that downloads helper GNSS codes: Hatanaka and gfzrnx

Parameters
opsys (*str*) – operating system. Allowed values are linux64, macos, and mac-newchip PC users should use the docker, where these executables come pre-installed

gnssrefl.installexe_cl.**newchip_gfzrnx**(*exedir*)

installs the gfzrnx executable and stores in the EXE directory

Parameters
exedir (*str*) – location of the executable directory

gnssrefl.installexe_cl.**newchip_hatanaka**(*exedir*)

compiles hatanaka code if an existing executable is not there stores in EXE directory

Parameters
exedir (*str*) – location of the executable directory

gnssrefl.installexe_cl.**parse_arguments**()

gnssrefl.invsnr_cl module

gnssrefl.invsnr_cl.**invsnr**(*station: str*, *year: int*, *doy: int*, *signal: str*, *peak2noise: float = 2.5*, *constel: str | None = None*, *screenstats: bool = False*, *dec: int = 1*, *polydeg: int = 2*, *snrfit: bool = True*, *plt: bool = True*, *doy_end: int | None = None*, *lspfigs: bool = False*, *snrfigs: bool = False*, *knot_space: int = 3*, *rough_in: float = 0.1*, *risky: bool = False*, *snr: int = 66*, *outfile_type: str = 'txt'*, *outfile_name: str = ''*, *outlier_limit: float = 0.5*, *no_dots: bool = False*, *delta_out: int = 300*, *refraction: bool = True*, *json_override: bool = False*, *lastday_seconds: int = 0*)

You must have run invsnr_input before using this code. This is the wrapper code that does the invsnr modelling. Note: outfile_name and outfile_type are unnecessary. Consolidate them.

In an earlier version of the code the pk2nlim was set to 4. Later the definition of the metric was changed, and this made the default setting far too stringent. In short, no arcs were being found. As of 2023/10/28 it is set to 2.5. This may not be optimal, but it is not as bad as 4. Please set it yourself as you prefer. Note: it will not the same as gnssir as this code was written separately and for a different purpose.

pktnlim is now known externally as peak2noise

Examples

invsnr sc02 2023 15 L1+L2+L5
would analyze day of year 15 and the L1, L2, and L5 signals

invsnr sc02 2023 15 ALL
would analyze day of year 15 and all signals

invsnr sc02 2023 15 L1+L2
would analyze day of year 15 and just L1 and L2

invsnr sc02 2023 15 L1+L2 -pk2nlim 2
would analyze day of year 15 and just L1 and L2, lower peak to noise limit ratio

invsnr sc02 2023 15 L1+L2 -doy_end 18
would analyze day of years 15 through 18 and L1 and L2 signals

Parameters

- **station** (*str*) – four character ID
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **signal** (*str*) – signal to use, L1 L2 L5 L6 L7 L1+L2 L1+L2+L5 L1+L5 ALL
- **peak2noise** (*float, optional*) – Peak2noise ratio limit for Quality Control. Default is 2.5
- **constel** (*str, optional*) – Only a single constellation. Default is gps, glonass, and galileo. value options:

  G : GPS

  E : Galileo

  R : Glonass

  C : Beidou

  withBeidou : adds Beidou to the default.
- **screenstats** (*bool, optional*) – Whether to print out stats to the screen. Default is False
- **dec** (*int, optional*) – SNR file decimator (seconds) Default is 1 (everything)
- **polydeg** (*integer, optional*) – polynomial degree for direct signal removal Default is 2
- **snrfit** (*bool, optional*) – Whether to do the inversion or not Default is True
- **plt** (*bool, optional*) – Whether to plot to the screen or not Default is True
- **doy_end** (*int, optional*) – day of year to end analysis. Default is None.
- **lspfigs** (*bool, optional*) – Whether or not to make LSP plots Note: Don't turn these on unless you really need plots because it is slow to make one per satellite arc. Default is False
- **snrfigs** (*boolean, optional*) – Whether or not to make SNR plots Don't turn these on unless you really need plots because it is slow to make one per satellite arc. Default is False
- **knot_space** (*float, optional*) – Knot spacing in hours Default is 3
- **rough_in** (*float, optional*) – Roughness Default is 0.1
- **risky** (*bool, optional*) – Risky taker related to gaps/knot spacing Default is False
- **snr** (*int, optional*) – SNR file ending. Default is 66
- **outfile_type** (*string, optional*) – output file type, txt or csv Default is txt
- **outfile_name** (*string, optional*) – output file name. Default is ??
- **outlier_limit** (*float, optional*) – Outliers plotted. (meters) Default is 0.5
- **no_dots** (*bool, optional*) – To plot lombscargle or not. Default is False
- **delta_out** (*int, optional*) – Output increment, in seconds. Default is 300
- **refraction** (*bool, optional*) – Default is True
- **json_override** (*bool, optional*) – Override json file name Default is False

- **lastday_seconds** (*int, optional*) – last time point (seconds of the day) should really be read from the file - if you don't provide this the fit blows up

gnssrefl.invsnr_cl.**main**()

gnssrefl.invsnr_cl.**parse_arguments**()

gnssrefl.invsnr_input module

gnssrefl.invsnr_input.**invsnr_input**(*station: str*, *h1: float = 1*, *h2: float = 8*, *e1: float = 5*, *e2: float = 15*, *azim1: float = 0*, *azim2: float = 360*, *lat: float | None = None*, *lon: float | None = None*, *height: float | None = None*, *peak2noise: float = 3*)

Sets some of the analysis parameters for invnsr. Values are stored in a json in $REFL_CODE/input Note: this code was written independently of gnssrefl. The Quality Control parametesr are thus quite different from how gnssrefl is done. The LSP RH retrievals are not - and should not - be the same.

Examples

invsnr_input sc02 -h1 3 -h2 12 -e1 5 -e2 13 -azim1 40 -azim2 220
general Friday Harbor inputs

invsnr_input at01 -h1 9 -h2 14 -e1 5 -e2 13 -azim1 20 -azim2 22
St Michael inputs

Parameters

- **station** (*str*) – four ch ID of the station
- **h1** (*float, optional*) – Lower limit reflector height (m)
- **h2** (*float, optional*) – Upper limit reflector height (m)
- **e1** (*float, optional*) – Lower limit elev. angle (deg)
- **e2** (*float*) – Upper limit elev. angle (deg)
- **azim1** (*float*) – Lower limit azimuth angle (deg)
- **azim2** (*float*) – Upper limit azimuth angle (deg)
- **lat** (*float, optional*) – Latitude (degrees)
- **lon** (*float, optional*) – Longitude (degrees)
- **height** (*float, optional*) – Ellipsoidal height (meters)
- **peak2noise** (*float, optional*) – peak to noise

gnssrefl.invsnr_input.**main**()

gnssrefl.invsnr_input.**parse_arguments**()

gnssrefl.karnak_libraries module

gnssrefl.karnak_libraries.**filename_plus**(*station9ch*, *year*, *doy*, *srate*, *stream*)

function to create RINEX 3 filenames for one day files.

Parameters

- **station9ch** (*str*) – 9 character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **srate** (*int*) – receiver sample rate
- **stream** (*str*) – R or S ; latter means the file was streamed.
- **output** (*str*) – compliant filename with crx.gz on the end as this is how the files are stored at GNSS archives as far as I know.

Returns

- **file_name** (*str*) – filename of the RINEX 3 file
- **cyyyy** (*str*) – 4ch year
- **cdoy** (*str*) – 3ch day of year

gnssrefl.karnak_libraries.**ga_stuff**(*station*, *year*, *doy*, *rinexv=3*)

GA API requirements to download a Rinex 3 file

Parameters

- **station** (*str*) – 9 character station name
- **year** (*integer*) – full year
- **doy** (*int*) – day of year
- **rinexv** (*int*) – rinex version

Returns

- **QUERY_PARAMS** (*dict*) – I think
- **headers** (*dict*) – I think

gnssrefl.karnak_libraries.**ga_stuff_highrate**(*station*, *year*, *doy*, *rinexv=3*)

This should be merged with existing ga_stuff code

Parameters

- **station** (*str*) – 4 or 9 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **rinexv** (*int*) – rinex version

Returns

- **QUERY_PARAMS** (*json*) – i think
- **headers** (*json*) – i think

gnssrefl.karnak_libraries.**gogetit**(*dir1*, *filename*, *ext*)

The purpose of this function is to to download RINEX 2 files. The code will try to get the file and check to see if it was successful.

Parameters

- **dir1** (*str*) – the main https directory address
- **filename** (*str*) – name of the GNSS files
- **ext** (*str*) – kind of ending to the filename, (Z, gz etc)

Returns

- **foundit** (*bool*) – whether file was found
- **f** (*str*) – name of the file

gnssrefl.karnak_libraries.**gsi_data**(*station*, *year*, *doy*)

get data from GSI, Japan

Parameters

- **station** (*str*) – 6 char station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

gnssrefl.karnak_libraries.**just_bkg**(*cyyyy*, *cdoy*, *file_name*)

looks for RINEX files at BKG in two directories

Parameters

- **cyyyy** (*str*) – four character year
- **cdoy** (*str*) – three character day of year
- **file_name** (*str*) – expected RINEX filename

gnssrefl.karnak_libraries.**make_rinex2_ofiles**(*file_name*)

take a rinex2 file, gunzip or uncompress it, and then Hatanaka decompress it

Parameters
file_name (*str*) – rinex2 filename

Returns

- **new_name** (*str*) – filename after multiple decompression processes
- **fexist** (*bool*) – whether file was successfully created

gnssrefl.karnak_libraries.**rinex2_highrate**(*station*, *year*, *doy*, *archive*, *strip_snr*)

kluge to download highrate data since i have revamped the rinex2 code strip_snr is boolean as to whether you want to strip out the non-SNR data it can be slow with highrate data. it requires gfzrnx

this no longer allows the all archive … which should have been stopped at rinex2snr_cl

Parameters

- **station** (*str*) – 4 character station ID. lowercase
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **archive** (*str*) – name of GNSS archive

- **strip_snr** (*bool*) – whether you want to strip out the observables (leaving only SNR)

gnssrefl.karnak_libraries.**rinex2names**(*station*, *year*, *doy*)

Creates the expected filename for rinex2 version files

Parameters

- **station** (*str*) – four character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **Results** –
- **-------** –
- **f1** (*str*) – hatanaka rinex filename
- **f2** (*str*) – regular rinex filename
- **cyyyy** (*str*) – four character year
- **cdoy** (*str*) – three character day of year

gnssrefl.karnak_libraries.**serial_cddis_files**(*dname*, *cyyyy*, *cdoy*)

Looks for rinex files in the hatanaka decompression section of cddis

Parameters

- **dname** (*str*) – rinex2 filename without compression extension
- **cyyyy** (*str*) – character string of full year
- **cdoy** (*str*) – character string (3) of day of year

Returns

- **foundit** (*bool*) – whether file was found
- **f** (*str*) – filename

gnssrefl.karnak_libraries.**strip_rinexfile**(*rinexfile*)

uses either teqc or gfzrnx to reduce observables, i.e. only SNR data.

Parameters
rinexfile (*str*) – name of the rinex2 file

gnssrefl.karnak_libraries.**swapRS**(*stream*)

function that swaps R to S and vice versa for RINEX 3 files

Parameters
stream (*str*) – RINEX 3 filename streaming acronym (S or R)

Returns
newstream – the opposite of what was in stream

Return type
str

gnssrefl.karnak_libraries.**universal**(*station9ch*, *year*, *doy*, *archive*, *srate*, *stream*, *debug=False*)

main code for seamless archive for RINEX 3 files …

Parameters

- **station9ch** (*str*) – nine character station name

- **year** (*int*) – year
- **doy** (*int*) – day of year
- **archive** (*str*) – archive name
- **srate** (*int*) – receiver samplerate
- **stream** (*str*) – one character: R or S
- **debug** (*bool*) – whether debugging statements printed

Returns

- **file_name** (*str*) – name of rinexfile (?? with gz or not? crx or rnx??)
- **foundit** (*boolean*) – whether file was found

gnssrefl.karnak_libraries.**universal_all**(*station9ch*, *year*, *doy*, *srate*, *stream*, *screenstats*)

check multiple archives for RINEX 3 data

Parameters

- **station9ch** (*str*) – 9 character station name
- **year** (*int*) – full year
- **doy** (*int*) – doy of year
- **srate** (*int*) – receiver sample rate
- **stream** (*str*) – R or S
- **screenstats** (*bool*) – print statements
- **Peturns** –
- **-------** –
- **file_name** (*str*) – rinex filename
- **foundit** (*bool*) – whether rinex file was found

gnssrefl.karnak_libraries.**universal_rinex2**(*station*, *year*, *doy*, *archive*, *screenstats*)

The long-awaited seamless archive for rinex 2.11 files …

Parameters

- **station** (*str*) – four character station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **archive** (*str*) – name of the GNSS archive
- **screenstats** (*bool*) – whether print statements come to scree

Returns

- **file_name** (*str*) – RINEX filename that was downloaded
- **foundit** (*bool*) – whether file was found

gnssrefl.kelly module

gnssrefl.kelly.**the_kelly_simple_way**(*url*, *filename*)

new way to access rinex files at unavco using earthscope-sdk downloads file - does not translate or uncompress

Updated 2023 august 20 to place and expect the token in REFL_CODE

Parameters

- **url** (*string*) – path to the file
- **filename** (*string*) – rinexfilename you are downloading. Could be hatanaka or not

Returns
foundit – whether file was found

Return type
bool

gnssrefl.llh2xyz module

gnssrefl.llh2xyz.**main**()

Command line tool that converts latitude, longitude, and ellipsoidal ht to Cartesian coordinates and prints to the screen

Examples

llh2xyz 39.949492042 -105.194266387 1728.856
returns -1283634.1616 -4726427.8934 4074798.0432

Parameters

- **lat** (*float*) – latitude in degrees
- **lon** (*float*) – longitude in degrees
- **height** (*float*) – ellipsoidal height in meters

Returns
XYZ – Cartesian coordinates to the screen (m)

Return type
float

gnssrefl.make_meta module

gnssrefl.make_meta.**check_offsets**(*station*, *meta_dict*)

check GAGE processing offset file for sources of possible gnssrefl timeseries offsets currently only relevant for ~3k stations processed by GAGE

Parameters

- **station** (*str*) – 4 character station ID.
- **meta_dict** (*dict*) – dictionary of metadata; keys 'dates','current','previous'.

Returns
meta_dict – dictionary of metadata; keys 'dates','current','previous'.

Return type
dict

gnssrefl.make_meta.**get_coords**(*station*, *lat*, *lon*, *height*)

initializes metadata dictionary with lat lon ht keys, either from UNR database (default) or user entered

Parameters

- **station** (*str*) – 4 character station ID.
- **lat** (*float, optional, default is None*) – latitude in deg
- **lon** (*float, optional, default is None*) – longitude in deg
- **height** (*float, optional, default is None*) – ellipsoidal height in m

Returns
comp_dict – dictionary of metadata; keys 'lat','lon','ht' 'meta'.

Return type
dict

gnssrefl.make_meta.**get_es_sdk_headers**()

checks for earthscope-sdk authentication token and refreshes repurposed from gnssrefl/kelly.py

Returns
header – dictionary of es-sdk token

Return type
dict

gnssrefl.make_meta.**main**()

gnssrefl.make_meta.**make_meta**(*station: str*, *lat: float | None = None*, *lon: float | None = None*, *height: float | None = None*, *man_input: bool = True*, *read_offset: bool = False*, *overwrite: bool = False*)

Make a json file that includes equipment metadata information. It saves your inputs to a json file saved in REFL_CODE/input/<station>_meta.json.

If station is in the UNR database, those lat/lon/ht values are used. You may override those values with the optional inputs.

The default is for the user to enter these values; multiple calls will append entries to the metadata array, unless the user sets overwrite to True. The user can attempt to extract some of this information from the GAGE offset file. GAGE file used is https://data.unavco.org/archive/gnss/products/offset/cwu.kalts_nam14.off (requires ES auth) [A caveat: this file is comprehensive for antennas changed by stations included in GAGE processing (n=~3k). Receivers are included, but incomplete.]

Examples

make_meta p038
makes json meta file for p038; uses UNR coords and will request user manually populate meta info. If meta json already exists for p038, will append to existing file.

make_meta test -lat 39.7417583 -lon -105.0706972 -height 1655
makes json meta file for test; uses manually input coords and will request ueser manually populate meta info

make_meta p038 -man_input False -read_offset True -overwrite True
makes json meta file for p038; uses UNR coords, will not request user manually populate meta info but will extract available meta info from GAGE offset file. Will overwrite any existing meta json file for p038

Parameters

- **station** (*str*) – 4 character station ID.
- **lat** (*float, optional*) – latitude in deg
- **lon** (*float, optional*) – longitude in deg
- **height** (*float, optional*) – ellipsoidal height in m
- **read_offset** (*bool, optional*) – set to True to parse GAGE offset file. default is False.
- **man_input** (*bool, optional*) – set to True to manually input equipment metadata. default is True.
- **overwrite** (*bool, optional*) – set to True to overwrite existing metadata file. default is False.

gnssrefl.make_meta.**meta_man_input**(*meta_dict*)

allows user to manually enter metadata information [Rx, Antenna, Dome, FW] at a given YYYY-mm-dd

Parameters
meta_dict (*dict*) – dictionary of metadata; keys 'dates','current','previous'.

Returns
meta_dict – dictionary of metadata; keys 'dates','current','previous'.

Return type
dict

gnssrefl.make_meta.**parse_arguments**()

gnssrefl.max_resolve_RH_cl module

gnssrefl.max_resolve_RH_cl.**main**()

gnssrefl.max_resolve_RH_cl.**max_resolve_RH**(*station: str*, *lat: float | None = None*, *lon: float | None = None*, *el_height: float | None = None*, *e1: float = 5*, *e2: float = 25*, *samplerate: float = 30*, *system: str = 'gps'*, *hires_figs: bool = False*)

Calculates the Maximum Resolvable Reflector Height. This is analogous to a Nyquist frequency for GNSS-IR. It creates a plot and makes a plain txt file in case you want to look at the numbers.

Examples

max_resolve_RH sc02 -e1 5 -e2 15
typical case for sites over water (5-15 degrees) but otherwise using defaults (GPS, 30 seconds)

max_resolve_RH sc02 -samplerate 15 -system galileo
Assume receiver sampling rate of 15 seconds and the Galileo constellation

max_resolve_RH xxxx -lat 40 -lon 100 -el_height 20
test your own site (xxxx) by inputting coordinates

Parameters

- **station** (*str*) – 4 ch station name
- **lat** (*float, optional*) – latitude in deg

- **lon** (*float, optional*) – longitude in deg
- **el_height** (*float, optional*) – ellipsoidal height in m
- **e1** (*float*) – min elevation angles (deg)
- **e2** (*float*) – max elevation angles (deg)
- **samplerate** (*float*) – receiver sampling rate
- **system** (*str, optional*) – name of constellation (gps,glonass,galileo, beidou allowed) default is gps
- **hires_figs** (*bool*) – whether you want eps files instead of png

Return type
Creates a figure file, stored in $REFL_CODE/Files/station

gnssrefl.max_resolve_RH_cl.**parse_arguments**()

gnssrefl.mjd module

gnssrefl.mjd.**main**()

converts MJD to year, month, day, hour, minute, second and prints that to the screen

Parameters
mjd (*float*) – modified julian date

gnssrefl.nmea2snr module

gnssrefl.nmea2snr.**angle_range_positive**(*ang*)

This code lacks documentation

Parameters
ang –

gnssrefl.nmea2snr.**azimuth_diff**(*azim1*, *azim2*)

Someone that is not me should document this

Parameters

- **azim1** –
- **azim2** –

Return type

???

gnssrefl.nmea2snr.**azimuth_diff1**(*azim*)

Someone that is not me should document this

Parameters
azim –

gnssrefl.nmea2snr.**azimuth_diff2**(*azim1*, *azim2*)

Someone that is not me should document this

gnssrefl.nmea2snr.**azimuth_mean**(*azim1*, *azim2*)

Someone that is not me should document this

Parameters

- **azim1** (*list of floats ?*) – azimuth degrees
- **azim2** (*list of floats*) – azimuth degrees

Returns

azim – azimuths in degrees

Return type

list of floats ?

gnssrefl.nmea2snr.**fix_angle_azimuth**(*time*, *angle*, *azimuth*)

interpolate elevation angles and azimuth to retrieve decimal values thru time this is for NMEA files. This is not used if sp3 orbit file used.

Parameters

- **time** (*list of floats*) – GPS seconds of the week
- **angle** (*list of floats*) – elevation angles (degrees)
- **azimuth** (*list of floats*) – azimuth angles (degrees)

Returns

- **angle_fixed** (*list of floats*) – interpolated elevation angles
- **azim_fixed** (*list of floats*) – interpolated azimuth angles

gnssrefl.nmea2snr.**nmea_apriori_coords**(*station*, *llh*, *sp3*)

gnssrefl.nmea2snr.**nmea_sp3_azel**(*recv*, *year*, *month*, *day*, *tod*, *prn*, *s1*, *s2*, *s5*, *s6*, *s7*, *s8*, *orbfile*, *snrfile*, *csnr*)

Compute azimuth/elevation from SP3 orbits for NMEA observations and write SNR file.

Parameters

- **recv** (*list of float*) – Cartesian receiver coordinates [X, Y, Z] in meters
- **year** (*int*) – calendar date
- **month** (*int*) – calendar date
- **day** (*int*) – calendar date
- **tod** (*ndarray*) – time of day in seconds (with leap-second offset applied)
- **prn** (*ndarray of int*) – satellite PRN numbers (gnssrefl convention: GPS 1-32, GLONASS 101-132, etc.)
- **s1** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **s2** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **s5** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **s6** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **s7** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **s8** (*ndarray of float*) – SNR values on L1, L2, L5, L6, L7, L8
- **orbfile** (*str*) – path to SP3 orbit file
- **snrfile** (*str*) – path to output SNR file

- **csnr** (*str*) – SNR option ('66', '99', etc.)

gnssrefl.nmea2snr.**nmea_translate**(*locdir*, *fname*, *snrfile*, *csnr*, *dec*, *year*, *doy*, *recv*, *sp3*, *gzip*, *orb*, *hour*)

Reads and translates a NMEA file stored in locdir + fname. The naming convention assumed for the NMEA file is SSSS1520.23.A where SSSS is station name, day of year is 152 and year is 2023 locdir is generally $REFL_CODE/nmea/SSSS/yyyy where yyyy is the year number and SSSS is the station name

Note from KL: I believe lowercase is also allowed (and preferred), but the A at the end is still set to be upper case (I believe) The SNR files are stored with upper case if given upper case, lower case if given lower case.

At request of Felipe Nievinski, the ultra option (which points to GFZ ultrarapid orbits) will first look for the orbit on (requested doy, hour 0), then (doy-1, hour 0), and then (doy-1, hour 12). I do not think this first search is correct - but I don't think it makes the code crash so I will keep it there for now.

Parameters

- **locdir** (*str*) – directory where your NMEA files are kept
- **fname** (*str*) – NMEA filename
- **snrfile** (*str*) – name of output file for SNR data
- **csnr** (*str*) – snr option, i.e. '66' or '99'
- **dec** (*int*) – decimation value in seconds
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **recv** (*list of floats*) – a priori Cartesian station coordinates for people using high quality orbits
- **sp3** (*bool*) – whether you use multi-GNSS sp3 file to do azimuth elevation angle calculations
- **gzip** (*bool*) – gzip compress snrfiles. No idea if it is used here … as this compression should happen in the calling function, not here
- **orb** (*str*) – requested orbit source
- **hour** (*int*) – ultrarapid orbit hour

gnssrefl.nmea2snr.**quickname**(*station*, *year*, *cyy*, *cdoy*, *csnr*)

Creates a full name of the snr file name (i.e. including the path)

Checks that directories exist.

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **cyy** (*str*) – two character year
- **cdoy** (*str*) – three character day of year
- **csnr** (*str*) – snr type, e.g. '66'

Returns
fname – output filename

Return type
str

gnssrefl.nmea2snr.**read_nmea**(*fname*)

reads a NMEA file.

Reads timing from RMC sentences (and GGA when present)

Parameters
fname (*str*) – NMEA filename

Returns

- **t** (*list of int*) – timetags in GPS seconds
- **prn** (*list of int*) – GPS satellite numbers
- **az** (*list of floats ??*) – azimuth values (degrees)
- **elv** (*list of floats ??*) – elevation angles (degrees)
- **snr** (*list of floats*) – snr values
- **freq** (*list of ???*) – apparently frequency values -

gnssrefl.nmea2snr.**run_nmea2snr**(*station*, *year*, *doy*, *isnr*, *overwrite*, *dec*, *llh*, *recv*, *sp3*, *gzip*, *orb*, *hour*)

runs the nmea2snr conversion code

Looks for NMEA files in $REFL_CODE/nmea/ssss/2023 for station ssss and year 2023. I prefer lowercase station names, but I believe the code allows both upper and lower case.

Files are named: SSSS1520.23.A

where SSSS is station name, day of year 152 and the last two characters of the 2023 as the middle value.

The SNR files are stored with upper case if given upper case, lower case if given lower case.

Parameters

- **station** (*str*) – 4 ch name of station
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **isnr** (*int*) – snr file type
- **overwrite** (*bool*) – whether make a new SNR file even if one already exists
- **dec** (*int*) – decimation in seconds
- **llh** (*list of floats*) – lat and lon (deg) and ellipsoidal ht (m)
- **recv** (*list of floats*) – cartesian receiver coordinates (m)
- **sp3** (*bool*) – whether you want to use GFZ rapid sp3 file for the orbits
- **gzip** (*bool*) – whether snrfiles are gzipped after creation
- **orb** (*str*) – requested orbit source
- **hour** (*int*) – requested hour for ultrarapid orbit

gnssrefl.nmea2snr_cl module

gnssrefl.nmea2snr_cl.**main**()

gnssrefl.nmea2snr_cl.**nmea2snr**(*station: str*, *year: int*, *doy: int*, *snr: int = 66*, *year_end: int | None = None*, *doy_end: int | None = None*, *overwrite: bool = False*, *dec: int = 1*, *lat: float | None = None*, *lon: float | None = None*, *height: float | None = None*, *risky: bool = False*, *gzip: bool = True*, *par: int | None = None*, *orb: str | None = None*, *hour: int = 0*, *debug: bool | None = None*)

This code creates SNR files from NMEA files.

The NMEA files should be stored in $REFL_CODE/nmea/ssss/2023 for station ssss and year 2023 or $REFL_CODE/nmea/SSSS/2023 for station SSSS. The NMEA files should be named SSSS1520.23.A or ssss1520.23.A, where the day of year is 152 and year is 2023 in this example.

The SNR files created are stored with upper case if given upper case, lower case if given lower case. Currently I have left the last character in the file name as it was given to me - capital A. If this should be lower case for people that use lowercase station names, please let me know. As far as I can tell, the necessary fields in the NMEA files are GPGGA and GPGSV.

Originally this code used interpolations of the az and el NMEA fields. I have decided this is DANGEROUS. If you really want to use those low-quality measurements, you have to say -risky T

The default usage is to use multi-GNSS orbits from GFZ. To compute az-el, you need to provide a priori station coordinates. You can submit those on the command line or it will read them from the $REFL_CODE/input/ssss.json file (for station ssss or SSSS) if it exists.

As of 2023 September 14, the SNR files are defined in GPS time, which is how the file is defined. Prior to version 1.7.0, if you used the sp3 option, the SNR files were written in UTC. This led to the orbits being propagated to the wrong time and thus az-el values are biased. The impact on RH is not necessarily large - but you should be aware. The best thing to do is remake your SNR files.

As for March 16, 2024, this code has been changed to use gnssrefl standards for inputs and outputs. The code, in principle, now looks for final, rapid, and ultra rapid orbits from GFZ, in that order.

As of version 3.6.4 you no longer have to enter station coordinates on the command line. You just need to follow the instructions in the file formats documentation to set up a list of the locations of your local stations.

I added parallel processing at some point - but I do not think this code allows you to call it as a function within a python script. If anyone knows how to fix this please submit a PR or let me know.

Parameters

- **station** (*str*) – 4 ch name of station
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snr** (*int, optional*) – snr file type (default is 66); 99: 5-30 deg.; 66: < 30 deg.; 88: all data; 50: < 10 deg
- **year_end** (*int, optional*) – final year
- **doy_end** (*int, optional*) – final day of year
- **overwrite** (*bool, optional*) – whether make a new SNR file even if one already exists
- **dec** (*int, optional*) – decimation in seconds
- **lat** (*float, optional*) – latitude, deg,
- **lon** (*float, optional*) – longitude, deg

- **height** (*float, optional*) – ellipsoidal height, m
- **risky** (*bool, optional*) – confirm you want to use low quality orbits (default is False)
- **gzip** (*bool, optional*) – compress SNR files after creation. Default is true
- **par** (*int*) – number of parallel processes. default is none (i.e. 1)
- **orb** (*str*) – specific orbit source (gnss, rapid, ultra, wum2). default is rapid
- **hour** (*int*) – specific hour of ultrarapid orbit. default is zero
- **debug** (*bool*) – whether you want your processing to be behind a "try". Without that, you will get uglier error messages, which can be helpful in figuring out why the code did not work.

Examples

nmea2snr wesl 2023 8 -dec 5
makes SNR file with decimation of 5 seconds with good orbits

nmea2snr wesl 2023 8
makes SNR file with original sampling rate and good orbits

nmea2snr xyz2 2023 8 -lat 40.2342 -lon -120.32424 -height 12
makes SNR file with user provided station coordinates and good orbits

gnssrefl.nmea2snr_cl.**parse_arguments**()

gnssrefl.nmea2snr_cl.**process_jobs_multi**(*index*, *args*, *datelist*, *error_queue*)

runs the nmea2snr queue

Parameters

- **index** (*int*) – which job to run
- **args** (*dict*) – dictionary of parameters for run_nmea2snr
- **datelist** (*dict*) – start and stop dates in MJD
- **error_queue** – not sure how to describe this

gnssrefl.nmea2snr_cl.**process_jobs_nopar**(*args*, *mjd_list*, *debug*)

runs the nmea2snr queue but in series, not in parallel optional debug input that moves your work outside the try so you can better see where the code is failing. Hopefully.

Parameters

- **args** (*dict*) – selections for nmea2snr like snr, orbit
- **datelist** (*list*) – modified julian day begin and end
- **debug** (*bool*) – whether you want the work done behind a try or not default is False

gnssrefl.nyquist_libs module

gnssrefl.nyquist_libs.**ny_plot**(*station*, *allN*, *info*, *hires_figs*)

allN is a numpy array of azimuth(deg)/nyquist(m) info is only needed for the title

Parameters

- **station** (*str*) – 4 char station name
- **allN** (*numpy array ?*) – azimuth and nyquist answers
- **info** (*str*) – information for the title
- **hires_figs** (*bool*) – whether you want eps instead of png

Returns
pngfile – name of plot file

Return type
str

gnssrefl.nyquist_libs.**pickup_files_nyquist**(*station*, *recv*, *obsfile*, *constel*, *e1*, *e2*, *reqsamplerate*, *hires_figs*)

Parameters

- **station** (*str*) – lowercase four character station name
- **recv** (*numpy array*) – Cartesian coordinates of station (m)
- **obsfile** (*str*) – name of orbit file
- **constel** (*int*) – requested constellation (1-4)
- **e1** (*float*) – min elevation angle (deg)
- **e2** (*float*) – max elevation angle (deg)
- **reqsamplerate** (*float*) – sample rate of receiver
- **hires_figs** (*bool*) – whether you want eps instead of png

gnssrefl.nyquist_libs.**read_the_orbits**(*obsfile*, *constel*)

Parameters

- **obsfile** (*str*) – name of the orbit file to be read
- **constel** (*int*) – which constellation (1-4), 1 for gps, 2 for glonass etc

gnssrefl.phase_functions module

gnssrefl.phase_functions.**apply_vwc_leveling**(*vwc_values*, *tmin*, *years=None*, *doys=None*, *mjd=None*, *level_doys=None*, *polyorder=-99*, ***kwargs*)

Apply global baseline leveling to time-averaged VWC estimates.

This is the FINAL leveling step applied to time-binned VWC values. Uses per-year polynomial fitting with dry season nodes for baseline calculation.

Parameters

- **vwc_values** (*array-like*) – Time-averaged VWC values in percentage units (0-60 range) These should be AFTER per-track processing and time-averaging
- **tmin** (*float*) – Minimum soil texture value (0.05 typical)

- **years** (*array-like, optional*) – Year values (required, or auto-derived from mjd)
- **doys** (*array-like, optional*) – Day of year values (required, or auto-derived from mjd)
- **mjd** (*array-like, optional*) – Modified Julian Day values (alternative to years/doys, will be auto-converted)
- **level_doys** (*list of int, optional*) – [start_doy, end_doy] for dry season baseline (required)
- **polyorder** (*int, optional*) – Polynomial order override (default: -99 = auto)
- ****kwargs** (*dict*) – Optional parameters: - station : str (for plot labels) - plot_debug : bool (generate diagnostic plots) - plt2screen : bool (show plots on screen) - extension : str (extension for subdirectory, e.g. 'test1' → 'station/test1') - fr : int (frequency code for file naming) - bin_hours : int (temporal binning for file naming) - bin_offset : int (temporal offset for file naming)

Returns

- **leveled_vwc** (*numpy.ndarray*) – Leveled VWC values in decimal units (0.0-0.6 range)
- **info** (*dict*) – Additional information about the leveling: - 'method': str - Always 'polynomial' - 'nodes': numpy.ndarray or None - Level nodes used - 'baseline_points': int or None - Number of baseline points used

Examples

leveled, info = apply_vwc_leveling(vwc, tmin=0.05,
years=yrs, doys=dys, level_doys=[152, 244])

gnssrefl.phase_functions.**apriori_file_exist**(*station*, *fr*, *extension=''*)

reads in the a priori RH results

Parameters

- **station** (*string*) – station name
- **fr** (*integer*) – frequency
- **extension** (*str, optional*) – analysis extension for finding files

Return type
boolean as to whether the apriori file exists

gnssrefl.phase_functions.**calculate_avg_phase**(*vxyz*, *bin_hours=24*, *bin_offset=0*, *minvalperbin=10*)

Calculate time-averaged phase statistics from track-level data

This is a pure calculation function that bins vxyz data by time and computes statistics without any file I/O.

Parameters

- **vxyz** (*numpy array*) – Track-level phase observations (16 columns) Columns: [year, doy, phase, azimuth, sat, rh, norm_amp_LSP, norm_amp_LS,

  hour, raw_LSP, raw_LS, apriori_RH, quadrant, delRH, vegMask, MJD]
- **bin_hours** (*int, optional*) – Time bin size in hours (default: 24 for daily)
- **bin_offset** (*int, optional*) – Bin timing offset in hours (default: 0)
- **minvalperbin** (*int, optional*) – Minimum observations required per bin (default: 10)

Returns
avg_phase – Averaged phase data with columns: [Year, DOY, Phase, PhaseSig, NormAmp, FracYear, Month, Day, [BinStart]] For daily: 8 columns (no BinStart) For subdaily: 9 columns (with BinStart)

Return type
numpy array (N x 8 or N x 9)

gnssrefl.phase_functions.**get_bin_schedule_info**(*bin_hours*, *bin_offset=0*)

Helper function to generate bin schedule information strings

Parameters

- **bin_hours** (*int*) – Time bin size in hours
- **bin_offset** (*int, optional*) – Bin timing offset in hours. Default is 0

Returns
Bin schedule description line

Return type
str

gnssrefl.phase_functions.**get_temporal_suffix**(*bin_hours=24*)

Generate the temporal-resolution suffix for vwc output filenames.

Format is *_{bin_hours}hr* (e.g. *_24hr*, *_6hr*). The bin_offset is intentionally not in the filename; it lives in the file header instead (see *write_avg_phase* / *write_vwc_output* subdaily header lines).

Parameters
bin_hours (*int, optional*) – Time bin size in hours. Default is 24 (daily)

Returns
Suffix string like "_24hr", "_6hr".

Return type
str

gnssrefl.phase_functions.**get_vwc_frequency**(*station: str*, *extension: str*, *fr_cmd: str | None = None*, *station_config: dict | None = None*)

Determines the frequency to use for VWC workflows. Priority is: command line -> json file -> default (20).

Parameters

- **station** (*str*) – 4-character station name.
- **extension** (*str*) – Analysis extension name.
- **fr_cmd** (*str, optional*) – Frequency provided from the command line (e.g., '1', '20'), by default None.
- **station_config** (*dict, optional*) – Pre-loaded json dict. Pass to skip re-reading the json file.

Returns
A list of frequencies to be used for analysis.

Return type
list

gnssrefl.phase_functions.**help_debug**(*rt*, *xdir*, *station*)

Takes the input of phase files, read by other functions, and writes out a file to help with debugging (comparion of matlab and python codes)

rt
[numpy array of floats] contents of the phase files stored in a numpy array

xdir
[str] where the otuput should be written

station
[str] name of the station

gnssrefl.phase_functions.**kinda_qc**(*satellite*, *track_avg_az*, *y*, *t*, *new_phase*, *avg_date*, *avg_phase*, *warning_value*, *remove_bad_tracks*, *avg_exist*)

Parameters

- **satellite** (*int*) – satellite number
- **track_avg_az** (*float*) – yearly average azimuth of the track, degrees
- **y** (*numpy array of ints*) – year
- **t** (*numpy array of ints*) – day of year
- **new_phase** (*numpy array of floats*) – phase values for a given satellite track
- **avg_date** (*numpy array of floats*) – y + doy/365.25
- **avg_phase** (*numpy array of floats*) – average phase, in degrees
- **warning_value** (*float*) – phase noise value
- **remove_bad_tracks** (*bool*) – whether you write out new tracks with bad ones removed
- **avg_exist** (*bool*) – whether you have previous solution to compare to

Returns
keepit – whether this track should be kept

Return type
bool

gnssrefl.phase_functions.**load_avg_phase**(*station*, *fr*, *bin_hours=24*, *extension=''*)

loads a previously computed averaged phase solution with matching temporal resolution. this is NOT the same as the multi-track phase results. This file is now stored in station subdirectory in $REFL_CODE/Files/

Parameters

- **station** (*str*) – 4 character station ID, lowercase
- **fr** (*int*) – frequency
- **bin_hours** (*int, optional*) – time bin size in hours (1,2,3,4,6,8,12,24). Default is 24 (daily). Only compares against files with exact same temporal resolution.
- **extension** (*str, optional*) – analysis extension for finding files, default is ''

Returns

- **avg_exist** (*bool*) – whether the necessary file exists
- **avg_date** (*list of floats*) – fractional year, i.e. year + doy/365.25
- **avg_phase** (*list of floats*) – average phase for the given temporal resolution

gnssrefl.phase_functions.**load_phase**(*station*, *year1*, *year2*, *fr*, *snowmask*, *extension=''*, *legacy=False*)

Load all phase data and attempt to remove outliers from snow if snowmask provided.

Parameters

- **station** (*str*) – four character station name
- **year1** (*int*) – starting year
- **year2** (*int*) – ending year
- **fr** (*int*) – frequency, i.e. 1 or 20
- **snowmask** (*str*) – name/location of the snow mask file None if this value is not going to be used

Returns

- **dataexist** (*bool*) – whether phase data were found
- **year** (*numpy array of int*) – calendar years
- **doy** (*numpy array of int*) – day of year
- **hr** (*numpy array of floats*) – UTC hour of measurement
- **ph** (*numpy array of floats*) – LS phase estimates
- **azdata** (*numpy array of floats*) – average azimuth, degrees
- **ssat** (*numpy array of int*) – satellite number
- **rh** (*numpy array of floats*) – reflector height, meters
- **amp_lsp** (*numpy array of floats*) – lomb scargle periodogram amplitude
- **amp_ls** (*numpy array of floats*) – least squares amplitude
- **ap_rh** (*numpy array of floats*) – apriori rh
- **results_trans** (*numpy array*) – all phase results concatenated into numpy array plus column for quadrant and unwrapped phase

gnssrefl.phase_functions.**load_sat_phase**(*station*, *year*, *year_end*, *freq*, *extension=''*)

Picks up the phase estimates from local (REFL_CODE) results section and returns most of the information from those files

Parameters

- **station** (*str*) – four character station name
- **year** (*integer*) – beginning year
- **year_end** (*integer*) – ending year
- **freq** (*integer*) – GPS frequency (1,20 allowed)

Returns

- **dataexist** (*bool*) – whether data found?
- **results** (*numpy array of floats*) – basically one variable with everything in the original columns from the daily phase files

gnssrefl.phase_functions.**low_pct**(*amp*, *basepercent*)

emulated amp_normK code from PBO H2O inputs are the amplitudes and a percentage used to define the bottom level. returns normalized amplitudes

this is meant to be used by individual tracks (I think) in this case they are the top values, not the bottom … ugh

gnssrefl.phase_functions.**make_snow_filter**(*station*, *medfilter*, *ReqTracks*, *year1*, *year2*)

Runs daily_avg code to make a snow mask file. This is so you have some idea of when the soil moisture products are contaminated by snow. Make a file with these years and doys saved. The user can edit if they feel the suggestsions are poor (i.e. days in the summer might show up as "snow")

If snow mask file exists, it does not overwrite it.

Parameters

- **station** (*str*) – 4 ch station name
- **medfilter** (*float*) – how much you allow the individual tracks to deviate from the daily median (meters)
- **ReqTracks** (*int*) – number of tracks to compute trustworthy daily average
- **year1** (*int*) – starting year
- **year2** (*int*) – ending year

Returns

- **snowmask_exists** (*bool*) – whether file was created
- **snow_file** (*str*) – name of the snow mask file
- *Creates output file into a file $REFL_CODE/Files/{ssss}/snowmask_{ssss}.txt*
- *where ssss is the station name*

gnssrefl.phase_functions.**mjd_to_file_columns**(*mjd_list*)

Convert MJD values to year, doy, month, day for file writing.

Parameters
mjd_list (*list or array*) – Modified Julian Day values

Returns

- **years** (*np.ndarray*) – Year values
- **doys** (*np.ndarray*) – Day of year values
- **months** (*np.ndarray*) – Month values
- **days** (*np.ndarray*) – Day of month values

gnssrefl.phase_functions.**normAmp**(*amp*, *basepercent*)

emulated amp_normK code from PBO H2O inputs are the amplitudes and a percentage used to define the bottom level. returns normalized amplitudes this is meant to be used by individual tracks (I think) in this case they are the top values, not the bottom … ugh

gnssrefl.phase_functions.**old_quad**(*azim*)

calculates oldstyle quadrants from PBO H2O

Parameters

- **azim** (*float*) – azimuth, dgrees
- **q** (*int*) – old quadrant system used in pboh2o

gnssrefl.phase_functions.**phase_file_fmt**(*columns*)

np.savetxt fmt string for a phase file given its column tuple.

gnssrefl.phase_functions.**phase_file_header**(*columns*)

Two-line '#'-style header ('Name1 Name2 …n(1) (2) …') for a phase file.

gnssrefl.phase_functions.**phase_tracks**(*station*, *year*, *doy*, *snr_type*, *fr_list*, *station_config*, *extension=''*, *legacy=False*)

Estimate phase and related parameters from the SNR file for one day.

Uses extract_arcs_from_station to detect arcs and tag each one with a per-arc apriori reflector height (apriori_RH) and track-mean azimuth (track_azim). In the default path the tagging is driven by vwc_tracks.json (a filtered copy of tracks.json with per-epoch apriori_RH attached); arcs that don't match any VWC-eligible epoch are skipped. The loop is the same in either path: it reads meta['apriori_RH'] / meta['track_azim'] off each arc and performs the LSP phase fit.

Parameters

- **name** (*station*) – 4 char id, lowercase
- **year** (*int*) – calendar year
- **doy** (*int*) – day of year
- **snr_type** (*int*) – SNR file extension (i.e. 99, 66 etc)
- **fr_list** (*list of integers*) – Frequencies to process. Typically the full all_frequencies() list in the default multi-GNSS path, or a GPS-only list such as [20] when legacy=True.
- **station_config** (*dict*) – station analysis parameters (from json, with CLI overrides applied)
- **extension** (*str, optional*) – analysis extension for json file. Default is ''.
- **legacy** (*bool, optional*) – When True, arcs are tagged from the legacy GPS apriori_rh_{fr}.txt files (sat + azimuth-within-3 degrees) instead of vwc_tracks.json. Default: False.

Notes

Output layout depends on legacy. The default path writes 18 columns: 16 phase columns plus TrackID and TrackEpoch carried from arc metadata. With legacy=True, the file stays at the historical 16-column layout (no track tags appended).

gnssrefl.phase_functions.**plot_baseline_leveling**(*station*, *years*, *doys*, *vwc_values*, *new_level*, *nodes*, *plot_path*, *tmin*, *level_doys*, *plt2screen=True*)

Plot baseline leveling results - before/after style like vegetation correction

gnssrefl.phase_functions.**prepare_track_dir**(*station*, *extension*, *fr*)

Create (or clear) the per-freq individual_tracks directory for a station.

Parameters

- **station** (*str*) – Station name
- **extension** (*str*) – Optional subdirectory extension
- **fr** (*int*) – Frequency code; selects the vwc_outputs/<label> subdirectory.

Returns
Absolute path to the individual_tracks directory

Return type
str

gnssrefl.phase_functions.**read_apriori_rh**(*station*, *fr*, *extension=''*)

read the track dependent a priori reflector heights needed for phase & thus soil moisture.

Parameters

- **station** (*str*) – four character ID, lowercase
- **fr** (*int*) – frequency (e.g. 1,20)

Returns

results – column 1 is just a number (1,2,3,4, etc)

column 2 is RH in meters

column 3 is satellite number

column 4 is azimuth of the track (degrees)

column 5 is number of values used in average

column 6 is minimum azimuth degrees for the quadrant

column 7 is maximum azimuth degrees for the quadrant

Return type

numpy array

gnssrefl.phase_functions.**rename_vals**(*year_sat_phase*, *doy*, *hr*, *phase*, *azdata*, *ssat*, *amp_lsp*, *amp_ls*, *rh*, *ap_rh*, *ii*)

this is just trying to clean up vwc.py send indices ii - and return renamed variables.

calculates and also returns mjd

Parameters

- **year_sat_sat** –
- **doy** (*numpy array*) – day of year (int)
- **hr** (*numpy array*) – floats (UTC hour)
- **phase** (*numpy array of floats*) – estimated phase (deg)
- **azdata** (*numpy array of floats*) – azimuth (deg)
- **ssat** (*numpy array of int*) – satellite numbers
- **amp_lsp** (*numpy array of floats*) – lomb scargle periodogram amplitude
- **amp_ls** (*numpy array of floats*) – least squares amplitude
- **rh** (*numpy array of floats*) – reflector heights (m)
- **ap_rh** – apriori reflector height (m)
- **ii** –

Returns

- **y** (*numpy array of int*) – year
- **t** (*numpy array of int*) – day of year
- **h** (*numpy array of floats*) – hour of the day (UTC)
- **x** (*numpy array of floats*) – phase, degrees
- **azd** (*numpy array of floats*) – azimuth for the track

- **s** (*numpy array of int*)
- **amps_lsp** (*numpy array of floats*) – LSP amplitude
- **amps_ls** (*numpy array of floats*) – least squares amplitude
- **rhs** (*numpy array of floats*) – estimated RH (m)
- **ap_rhs** (*numpy array of floats*) – a priori RH (m)
- **mjds** (*numpy array of int*) – Modified julian day

gnssrefl.phase_functions.**save_vwc_plot**(*fig*, *pngfile*)

Parameters

- **fig** (*matplotlib figure*) – the figure definition you define when you open a figure
- **pngfile** (*str*) – name of the png file to be saved

gnssrefl.phase_functions.**set_parameters**(*station*, *level_doys*, *minvalperday*, *tmin*, *tmax*, *min_req_pts_track*, *fr*, *year*, *year_end*, *plt*, *auto_removal*, *warning_value*, *extension*, *bin_hours=None*, *minvalperbin=None*, *bin_offset=None*)

the goal of this code is to pick up the relevant parameters used in vwc.

Parameters

- **station** (*str*) – 4 character station name
- **level_doys** (*list of int*) – option doy inputs for defining leveling period. can be in the json or on the command line
- **minvalperday** (*int*) – number of phase values required each day
- **tmin** (*float*) – min soil texture
- **tmax** (*float*) – max soil texture
- **min_req_pts_track** (*int*) – minimum number of phase values per year per track
- **freq** (*int*) – frequency to use (1,20 allowed)
- **year** (*int*) – first year to analyze
- **year_end** (*int*) – last year to analyze
- **plt** (*bool*) – whether you want plots to come to the screen
- **auto_removal** (*bool*) – whther tracks should be removed when they fail QC
- **warning_value** (*float*) – phase RMS needed to trigger warning
- **extension** (*str*) – extension used for inputs and outputs (i.e. test1 in ssss.test1.json)
- **bin_hours** – needs documentation
- **?** (*bin_offset*) – needs documentation
- **?** – needs documentation

Returns

- **minvalperday** (*int*) – number of phase values required to create a daily average
- **tmin** (*float*) – min soil texture
- **tmax** (*float*) – max soil texture

- **min_req_pts_track** (*int*) – minimum number of phase values per year per track
- **freq** (*int*) – frequency to use (1,20 allowed)
- **year_end** (*int*) – last year to analyze
- **plt** (*bool*) – whether you want plots to come to the screen
- **auto_removal** (*bool*) – whther tracks should be removed when they fail QC
- **warning_value** (*float*) – phase RMS needed to trigger warning
- **extension** (*str*) – extra name for the json file
- **return_level_doys** (*list*) – start and end day of years for leveling
- **vegetation_model** (*int*) – vegetation correction model: 1 (simple), 2 (advanced)

gnssrefl.phase_functions.**subdaily_phase_plot**(*station*, *fr*, *vxyz*, *outdir*, *hires_figs*, *bin_hours=24*, *bin_offset=0*, *minvalperbin=10*, *plt2screen=True*)

makes a plot of daily or subdaily averaged phase for vwc code

Parameters

- **station** (*str*) – 4 char station name
- **fr** (*int*) – frequency of signal
- **vxyz** (*numpy array*) – Track-level phase data
- **outdir** (*str or Path*) – directory to write the plot into
- **hires_figs** (*bool*) – whether you want eps instead of png files
- **bin_hours** (*int*) – time bin size in hours (default: 24 for daily)
- **bin_offset** (*int*) – bin timing offset in hours (default: 0)
- **minvalperbin** (*int*) – minimum observations required per bin (default: 10)
- **plt2screen** (*bool*) – whether to display plot to screen (default: True)

gnssrefl.phase_functions.**test_func**(*x*, *a*, *b*, *rh_apriori*)

This is least squares for estimating a sine wave given a fixed frequency, freqLS

gnssrefl.phase_functions.**test_func_new**(*x*, *a*, *b*, *rh_apriori*, *freq*, *sat=None*)

This is least squares for estimating a sine wave given a fixed frequency, freqLS now freq is input so it is not hardwired for L2

Parameters

- **x** (*numpy array of floats*) – sine(elevation angle) I think
- **a** (*float*) – amplitude - estimated
- **b** (*float*) – phase - estimated
- **rh_apriori** (*float*) – reflector height (m)
- **freq** (*int*) – frequency
- **sat** (*int, optional*) – satellite number, required for GLONASS (per-SV FDMA wavelengths)

gnssrefl.phase_functions.**vwc_plot**(*station*, *t_datetime*, *vwcdata*, *plot_path*, *circles*, *plt2screen=True*)

makes a plot of volumetric water content

Parameters

- **station** (*string*) – 4 ch station name
- **t_datetime** (*datetime*) – observation times for measurements
- **vwcdata** (*numpy array of floats (I think)*) – volumetric water content
- **plot_path** (*Saves a plot to*) – full name of the plot file
- **circles** (*boolean*) – circles in the plot. default is a line (really .-)
- **plot_path** –

gnssrefl.phase_functions.**write_all_phase**(*v*, *fname*)

writes out preliminary phase values and other metrics for advanced vegetation option. This is in the hope that it can be used in clara chew's dissertation algorithm.

File is written to $REFL_CODE/Files/station/station_all_phase.txt I think

Parameters

- **v** (*numpy of floats as defined in vwc_cl*) – TBD year, doy, phase, azimuth, satellite number estimated RH, LSP amplitude, LS amplitude, UTC hours raw LSP amp, raw LS amp
- **fname** (*str*) – name of the output file
- **filestatus** (*int*) – 1, open the file 2, write to file (well, really any value)
- **rhtrack** (*float*) – apriori reflector height for the given track, meters

Returns
allrh

Return type
fileID

gnssrefl.phase_functions.**write_apriori_rh**(*filepath*, *tracks*, *station*, *year*, *tmin*, *tmax*)

Write apriori RH track list file. Used by both vwc_input and vwc auto_removal.

A track is defined by satellite number and the azimuth at minimum elevation angle (az_min_ele) of each arc. Arcs are clustered into tracks when their az_min_ele values fall within 3 degrees of the cluster center. The track's mean azimuth (track_avg_az) is the circular mean of az_min_ele over e.g. a year of arcs in the cluster.

Parameters

- **filepath** (*path-like*) –
- **tracks** (*list*) – each element is [track_num, rh, satellite, track_avg_az, nvals, az_min, az_max]
- **station** (*str*) –
- **year** (*int*) –
- **tmin** (*float*) – minimum soil texture (header metadata only)
- **tmax** (*float*) – maximum soil texture (header metadata only)

gnssrefl.phase_functions.**write_avg_phase**(*station*, *fr*, *avg_phase*, *extension=''*, *bin_hours=24*, *bin_offset=0*)

writes averaged phase results to a text file

Parameters

- **station** (*string*) –
- **fr** (*int*) – frequency
- **avg_phase** (*numpy array*) – output of calculate_avg_phase columns: [Year, DOY, Phase, PhaseSig, NormAmp, FracYear, Month, Day, [BinStart]]
- **extension** (*str, optional*) – analysis extension for directory organization
- **bin_hours** (*int, optional*) – Time bin size in hours (default: 24). Used for filename suffix and header.
- **bin_offset** (*int, optional*) – Bin timing offset in hours (default: 0). Used for filename suffix and header.

gnssrefl.phase_functions.**write_out_raw_phase**(*v*, *fname*, *legacy=False*)

Write daily phase values used in vwc to a consolidated file, inserting quadrant and unwrapped-phase columns after the core columns. In the default (multi-GNSS) path TrackID/TrackEpoch stay as the final two columns; see PHASE_COLS in this module for the exact schema.

Parameters

- **v** (*numpy array*) – Per-day phase results concatenated across years (optionally snow-filtered). Has the per-day layout: PHASE_COLS_LEGACY / PHASE_COLS minus quad and unphase.
- **fname** (*str*) – Filename for output.
- **legacy** (*bool, optional*) – Phase-file layout. `True` for the historical GPS layout (no track tags). `False` (default) for the multi-GNSS layout, which groups the unwrap by (TrackID, TrackEpoch) rather than by (sat, quadrant).

Returns
newv – Rows laid out per PHASE_COLS_LEGACY or PHASE_COLS.

Return type
numpy array

gnssrefl.phase_functions.**write_phase_for_advanced**(*filename*, *vxyz*, *station*, *fr*)

Writes per-track phase observations as a QA dump (one row per observation).

File written to $REFL_CODE/Files/<station>/vwc_outputs/<freq>/<station>_all_phase.txt

Parameters

- **filename** (*str*) – name for output file
- **vxyz** (*numpy array of floats*) – as defined in vwc_cl.py
- **station** (*str*) – 4-char station name
- **fr** (*int*) – frequency code

gnssrefl.phase_functions.**write_rolling_vwc_output**(*station*, *vwc_data*, *fr*, *bin_hours*, *extension=''*, *vegetation_model=2*)

Write hourly rolling VWC output file.

This writes a combined file with all hourly rolling measurements (e.g., bins starting at 0:00, 1:00, 2:00, etc.) sorted chronologically.

Parameters

- **station** (*str*) – 4-character station name
- **vwc_data** (*dict*) – Must contain: - 'mjd': Modified Julian Day values - 'vwc': VWC values (already leveled) - 'datetime': datetime objects - 'bin_starts': list of bin start hours
- **fr** (*int*) – Frequency code (1=L1, 5=L5, 20=L2C)
- **bin_hours** (*int*) – Time bin size in hours (e.g., 6 for 6-hour windows)
- **extension** (*str, optional*) – Extension for file naming (default: '')
- **vegetation_model** (*int, optional*) – 1=simple, 2=advanced (for header documentation, default: 2)

gnssrefl.phase_functions.**write_track_file**(*track_dir*, *station*, *sat_num*, *avg_az*, *rows*, *track_id*, *track_epoch*, *extra_headers=None*)

Write a single track file with standard header and row format.

Track identity is always carried in the filename via (sat_num, avg_az). Under the modern path the (track_id, track_epoch) pair from vwc_tracks.json is also prefixed; under legacy those are -1 sentinels and the prefix is omitted.

Parameters

- **track_dir** (*str*) – Directory to write the track file in
- **station** (*str*) – Station name
- **sat_num** (*int*) – Satellite number for filename display
- **avg_az** (*float*) – Track average azimuth (degrees) for filename display
- **rows** (*numpy.ndarray*) – (N, 17) array of track data columns
- **track_id** (*int*) – Track id, or -1 under legacy
- **track_epoch** (*int*) – Track epoch, or -1 under legacy
- **extra_headers** (*list of str, optional*) – Additional header lines inserted before the column legend

gnssrefl.phase_functions.**write_vwc_output**(*station*, *vwc_data*, *year*, *fr*, *bin_hours*, *bin_offset*, *extension=''*, *vegetation_model=1*)

Write VWC output file in standard gnssrefl format. Works for both vegetation models.

Parameters

- **station** (*str*) – 4-character station name
- **vwc_data** (*dict*) – Must contain: - 'mjd': Modified Julian Day values - 'vwc': VWC values - 'datetime': datetime objects - 'bin_starts': list of bin start hours (subdaily) or empty list (daily)
- **year** (*int*) – Year for file naming
- **fr** (*int*) – Frequency code (1=L1, 5=L5, 20=L2C)
- **bin_hours** (*int*) – Time bin size in hours (default: 24)
- **bin_offset** (*int*) – Bin timing offset in hours (default: 0)
- **extension** (*str, optional*) – Extension for file naming (default: '')

- **vegetation_model** (*int, optional*) – 1=simple, 2=advanced (for header documentation, default: 1)

gnssrefl.pickle_dilemma module

gnssrefl.pickle_dilemma.**main**()

no inputs. checks for the feared pickle file

gnssrefl.prn2gps module

gnssrefl.prn2gps.**download_prn_gps**()

downloads PRN to GPS name conversion file from JPL

gnssrefl.prn2gps.**main**()

Displays the PRN and SVN numbers for the GPS constellation on a given date.

Parameters

- **date** (*str*) – example 2012-01-01
- **overwrite** (*bool, optional*) – whether you want to download new file from JPL

gnssrefl.prn2gps.**read_jpl_file**(*fname*)

Parameters
fname (*str*) – filename with prn gps conversion info

Returns
tv – start date (frac year), end date, GPS#, PRN #

Return type
list

gnssrefl.query_unr module

gnssrefl.query_unr.**main**()

Extracts coordinates for stations that were in the UNR database in late 2021. Prints both geodetic and cartesian values, and height above sea level.

Parameters
station (*str*) – four character station name

gnssrefl.quickLook_cl module

quickLook command line function

gnssrefl.quickLook_cl.**main**()

gnssrefl.quickLook_cl.**parse_arguments**()

gnssrefl.quickLook_cl.**quicklook**(*station: str*, *year: int*, *doy: int*, *snr: int = 66*, *fr: int = 1*, *ampl: float = 7.0*, *e1: float = 5*, *e2: float = 25*, *h1: float = 0.5*, *h2: float = 8.0*, *sat: int | None = None*, *peak2noise: float = 3.0*, *screenstats: bool = False*, *plt: bool = True*, *azim1: float = 0.0*, *azim2: float = 360.0*, *ediff: float = 2.0*, *delTmax: float = 75.0*, *hires_figs: bool = False*)

quickLook assessment of GNSS-IR results using SNR data. It creates two plots: one with periodograms for four different quadrants (northwest, northeast, southeast, southwest) and the other with the RH results shown as a function of azimuth. This plot also summarizes why the RH retrievals were accepted or rejected in terms of the quality control parameters.

Examples

quickLook p041 2023 1
analyzes station p041 on day of year 1 in the year 2023 with defaults (L1, e1=5, e2=25)

quickLook p041 2023 1 -h1 1 -h2 10
analyzes station p041 on day of year 1 in the year 2023. The periodogram would be restricted to RH of 1-10 meters.

quickLook p041 2023 1 -fr 20
Uses L2C frequency instead of GPS L1.

quickLook p041 2023 1 -azim1 10 -azim2 180
Only shows azimuths between 10 and 180

If your site name is in the GNSS-IR database (which is generated from the Nevada Reno geodesy group), or in the local coordinates database, a standard refraction correction is applied. If not, it does not. This is most relevant for very very tall sites, i.e. > 200 meters. Refraction models are always applied in the gnssir module.

If users would like the refraction correction to be applied here for stations that are not in the standard database, they need to submit a pull request making that possible. One option is to read from an existing gnssir_input json file. That is what is done in nmea2snr and invsnr_input.

Parameters

- **station** (*str*) – 4 character ID of the station
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **snr** (*int, optional*) – SNR format. This tells the code which SNR file to use. 66 is the default. Other options: 50, 88, and 99.
- **f** (*int, optional.*) – GNSS frequency. Default is GPS L1 value options:

  1,2,20,5 : GPS L1,L2,L2C,L5 (1 is default)

  101,102 : GLONASS L1 and L2

  201,205,206,207,208 : GALILEO E1 E5a E6,E5b,E5

  301,302,305,306,307,308 : various BEIDOU
- **reqAmp** (*int or array_like, optional*) – Lomb-Scargle Periodogram (LSP) amplitude significance criterion in volts/volts. Default is 7
- **e1** (*float, optional*) – elevation angle lower limit in degrees for the LSP. default is 5.
- **e2** (*float, optional*) – elevation angle upper limit in degrees for the LSP. default is 25.
- **h1** (*float, optional*) – The allowed LSP reflector height lower limit in meters. default is 0.5.
- **h2** (*float, optional*) – The allowed LSP reflector height upper limit in meters. default is 6.

- **sat** (*integer, optional*) – specific satellite number, default is None.
- **peak2noise** (*int, optional*) – peak to noise ratio of the periodogram values (periodogram peak divided by the periodogram noise). For snow and ice, 3.5 or greater, tides can be tricky if the water is rough (and thus you might go below 3 a bit, say 2.7 default is 3.
- **screenstats** (*boolean, optional*) – Whether to print stats to the screen. default is False.
- **plt** (*boolean, optional*) – Whether to print plots to the screen. default is True. Regardless, png files are made
- **azim1** (*float, optional*) – minimum azimuth angle (deg) default is 0.
- **azim2** (*float, optional*) – maximum azimuth angle (deg) default is 360.
- **ediff** (*float, optional*) – elevation angle difference, quality control parameter default is 2 degrees.
- **delTmax** (*float, optional*) – maximum allowed arc length, in minutes default is 75 minutes.
- **hires_figs** (*bool, optional*) – eps instead of png files

gnssrefl.quickLook_function2 module

gnssrefl.quickLook_function2.**colorful**(*a*, *px*, *pz*, *lw*, *fullcolor*, *ax*)

plots the quadrant periodograms

Parameters

- **a** (*int*) –
- **px** (*numpy array of floats*) – x axis of amplitude power spectrum
- **pz** (*numpy array of floats*) – y axis of amplitude power spectrum
- **lw** (*float*) – line width
- **fullcolor** – whether you want full color (if not, it is gray)
- **ax** (*axis handle*) –

gnssrefl.quickLook_function2.**goodbad**(*fname*, *station*, *year*, *doy*, *h1*, *h2*, *PkNoise*, *reqAmp*, *freq*, *e1*, *e2*, *hires_figs*)

makes a plot that shows "good" and "bad" refletor height retrievals as a function of azimuth

Parameters

- **fname** (*str*) – filename
- **station** (*str*) – 4 char station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **h1** (*float*) – minimum reflector height (m)
- **h2** (*float*) – max reflector height (m)
- **PkNoise** (*float*) – peak 2 noise QC
- **reqAmp** (*float*) – required LSP amplitude
- **freq** (*int*) – frequency

- **e1** (*float*) – minimum elevation angle (deg)
- **e2** (*float*) – maximum elevation angle (deg)
- **hires_figs** (*bool*) – whether to use eps instead of png

Return type
plot is written $REFL_CODE/Files/station/quickLook_summary.png

gnssrefl.quickLook_function2.**quickLook_function**(*station*, *year*, *doy*, *snr_type*, *f*, *e1*, *e2*, *minH*, *maxH*, *reqAmp*, *pele*, *satsel*, *PkNoise*, *pltscreen*, *azim1*, *azim2*, *ediff*, *delTmax*, *hires_figs*, ***kwargs*)

This is the main function to compute spectral characteristics of a SNR file. It takes in all user inputs and calculates reflector heights. It makes two png files to summarize the data.

This is a new version that tries to picks all rising and setting arcs, not just those constrained to 90 degree quadrants.

It will attempt to make a refraction correction if you have a local coordinate file or a station analysis json. The format of the local coordinate file can be found in the documentation for query_coordinate_file which is in the gps library.

Parameters

- **station** (*str*) – station name (4 char)
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **snr_type** (*int*) – snr file extension (i.e. 99, 66 etc)
- **f** (*int*) – frequency (1, 2, 5), etc
- **e1** (*float*) – minimum elevation angle in degrees
- **e2** (*float*) – maximum elevation angle in degrees
- **minH** (*float*) – minimum allowed reflector height in meters
- **maxH** (*float*) – maximum allowed reflector height in meters
- **reqAmp** (*float*) – is LSP amplitude significance criterion
- **pele** (*list of floats*) – is the elevation angle limits for the polynomial removal. units: degrees
- **satsel** (*int*) – satellite number
- **PkNoise** (*float*) – peak to noise ratio for QC
- **pltscreen** (*bool*) – whether you want plots sent to the terminal
- **azim1** (*float*) – minimum azimuth in degrees
- **azim2** (*float*) – maximum azimuth in degrees
- **ediff** (*float*) – QC parameter - restricts length of arcs (degrees)
- **delTmax** (*float*) – maximum arc length in minutes
- **hires_figs** (*bool*) – whether to use eps instead of png

gnssrefl.quickLook_function2.**quick_refraction**(*station*)

computes the necessary information for a refraction correction used in quickLook. no time dependence.

Parameters
station (*str*) – 4 character station name

Returns

- **p** (*float*) – pressure, hPa
- **T** (*float*) – temperature, Celsius
- **irefr** (*int*) – refraction model number I believe, which is also sent, so not needed
- **e** (*float*) – water vapor pressure, hPa

gnssrefl.quickLook_function2.**set_labels**(*ax*, *axisSize*, *fs*)

try to set the appropriate labels depending on the quadrant

Parameters

- **ax** (*dictionary*) – plot handles
- **axisSize** (*numpy array of floats*) – lists the amplitudes of the various periodograms in a given quadrant pairs of quad (0-3), amplitude
- **fs** (*int*) – font size

gnssrefl.quickLook_function2.**whichquad**(*iaz*)

Parameters

- **iaz** (*float*) – azimuth of the arc, degrees
- **a** (*int*) – quad number in funny system

gnssrefl.quickPhase module

gnssrefl.quickPhase.**count_result_arcs**(*result_path*)

Count non-comment lines in a result file.

gnssrefl.quickPhase.**main**()

gnssrefl.quickPhase.**parse_arguments**()

gnssrefl.quickPhase.**process_phase_day**(*year*, *doy*, *args*)

Process a single day for sequential mode.

gnssrefl.quickPhase.**process_phase_day_worker**(*worker_args*)

Worker for parallel phase processing. Returns arc count for progress bar.

gnssrefl.quickPhase.**process_phase_sequential**(*year*, *year_end*, *doy*, *doy_end*, *args*)

Sequential processing function

gnssrefl.quickPhase.**quickphase**(*station: str*, *year: int*, *doy: int*, *year_end: int | None = None*, *doy_end: int | None = None*, *snr: int = 66*, *fr=None*, *e1: float | None = None*, *e2: float | None = None*, *plt: bool = False*, *screenstats: bool = False*, *gzip: bool | None = None*, *extension: str = ''*, *par: int | None = None*, *midnite: bool = True*, *ampl: float | None = None*, *savearcs: bool = False*, *savearcs_format: str | None = None*, *dec: int | None = None*, *dbhz: bool | None = None*, *recompute_lsp: bool = False*, *legacy: bool = False*)

quickphase computes phase, which are subsequently used in vwc. The command line call is phase (which maybe we should change).

By default, tagging of arcs with apriori RH and track-mean azimuth is driven by vwc_tracks.json (produced by vwc_input). One unified pipeline across GPS/GLONASS/Galileo/BeiDou. Pass -legacy T to use the old GPS-only flow that reads apriori_rh_{fr}.txt.

Examples

phase p038 2021 4
analyzes data for year 2021 and day of year 4

phase p038 2021 1 -doy_end 365
analyzes data for the whole year

phase p038 2021 1 -doy_end 365 -par 5
analyzes data for the whole year using 5 parallel processes

Parameters

- **station** (*str*) – 4 character ID of the station.
- **year** (*int*) – full Year to evaluate.
- **doy** (*int*) – day of year to evaluate.
- **year_end** (*int, optional*) – year to end analysis. Using this option will create a range from year-year_end. Default is None.
- **doy_end** (*int, optional*) – Day of year to end analysis. Using this option will create a range of doy-doy_end. If also using year_end, then this will be the day to end analysis in the year_end requested. Default is None.
- **snr** (*int, optional*) – SNR format. This tells the code what elevation angles are in the SNR file value options:

  66 (default) : data with elevation angles less than 30 degrees

  99 : data with elevation angles between 5 and 30 degrees

  88 : data with all elevation angles

  50 : data with elevation angles less than 10 degrees
- **fr** (*list of int, optional*) – Default path: subset of the frequencies present in vwc_tracks.json. When None (default), all freqs in vwc_tracks.json are processed. An error is raised if a requested freq is not in vwc_tracks.json. Legacy path: single frequency (1, 20, or 5). Default 20 (L2C).
- **e1** (*float, optional*) – Elevation angle lower limit in degrees for the LSP. default is from json (typically 5)
- **e2** (*float, optional*) – Elevation angle upper limit in degrees for the LSP. default is from json (typically 25)
- **plt** (*bool, optional*) – Whether to plot results. Default is False
- **screenstats** (*bool, optional*) – Whether to print stats to the screen. Default is False
- **gzip** (*bool, optional*) – gzip the SNR file after use. Default is True
- **par** (*int, optional*) – Number of parallel processes to spawn (up to 10). Default is 1 (single process).
- **midnite** (*bool, optional*) – Allow midnight crossings. When True, loads +/- 2 hours from adjacent days. Default is True.

Returns

- **Saves a file for each day in the doy-doy_end range** (*$REFL_CODE/<year>/phase/<station>/<doy>.txt*)

- *columns in files* – year doy hour phase nv azimuth sat ampl emin emax delT aprioriRH freq estRH pk2noise LSPAmp

gnssrefl.quicklib module

gnssrefl.quicklib.**save_plot**(*out*)

Makes a png plot and saves it in REFL_CODE/Files

Parameters

out (*str*) – name of output file (or None)

gnssrefl.quicklib.**set_xlimits_ydoy**(*xlimits*)

translate command line xlimits into datetime for nicer plots

Parameters

xlimits (*list of floats*) – year (fractional, i.e. 2015.5)

Returns

- **t1** (*datetime*) – beginning date for x-axis
- **t2** (*datetime*) – end date for x-axis

gnssrefl.quicklib.**trans_time**(*tvd*, *ymd*, *ymdhm*, *convert_mjd*, *ydoy*, *xcol*, *ycol*, *utc_offset*)

translates time for quickplt

Parameters

- **tvd** (*numpy array*) – contents of whatever file was read by loadtxt in quickplt
- **ymd** (*bool*) – first three columns are year,month,day, hour, minute,second
- **ymdhm** (*bool*) – first five columns are year,month,day, hour, minut,
- **convert_mjd** (*bool*) – convert from MJD (column 1 designation) time is datetime obj
- **ydoy** (*bool*) – first two columns are year and day of year time is datetime obj
- **xcol** (*int*) – column number for x-axis in python speak
- **ycol** (*int*) – column number for y-axis in python speak
- **utc_offset** (*int*) – offst in hours from UTC/GPS time. None means do not use

Returns

- **tval** (*numpy array*) – time, via floats or datetime, depending on what was requested
- **yval** (*numpy array*) – floats - whatever is being plotted on the yaxis

gnssrefl.quickplt module

gnssrefl.quickplt.**main**()

gnssrefl.quickplt.**parse_arguments**()

gnssrefl.quickplt.**run_quickplt**(*filename: str*, *xcol: str*, *ycol: str*, *errorcol: int | None = None*, *mjd: bool = False*, *xlabel: str | None = None*, *ylabel: str | None = None*, *symbol: str | None = None*, *reverse: bool = False*, *title: str | None = None*, *outfile: str | None = None*, *xlimits: str | None = None*, *ylimits: float = []*, *ydoy: bool = False*, *ymd: bool = False*, *ymdhm: bool = False*, *filename2: str | None = None*, *freq: int | None = None*, *utc_offset: int | None = None*, *yoffset: float | None = None*, *keepzeros: bool = False*, *sat: str | None = None*, *yoffset2: float | None = None*, *scale: float = 1.0*, *scale2: float = 1.0*, *elimits: float = [0]*, *azlimits: float = [0]*, *plt: bool = True*)

quick file plotting using matplotlib

A png file is saved as temp.png or to your preferred filename if outfile is given. In either case, it goes to REFL_CODE/Files

Allows you to set x and y-axis limits with a title and various axes labels, symbols etc.

Someone could easily update this to include different filetypes (e.g. jpeg)

I rewrote this recently to take advantage of our boolean argument translator. Let me know if things have broken or submit a PR.

To make simple plots of observables in SNR files, the x-axis can be either time or elevation. The latter is short for elevation angle. To pick this option set -sat to a specific satellite nubmer or a constellation (gps, glonass, etc). You can also set elimits and azlimits for simple elevation angle and azimuth angle limits. Only for SNR files, you can send the name of the SNR file without the directory, i.e. sc021500.22.snr66 instead of REFL_CODE/2022/snr/sc02/sc021500.22.snr66

You may submit a snr filename that has been gzipped. The code will checked to see if the gzip version is there and gunzip it for you. But this option does not exist for other kinds of files.

Examples

quickplt txtfile 1 16
: would plot column 1 on the x-axis and column 16 on the y-axis

quickplt sc021500.22.snr66 time L1 -sat gps
: would plot all the GPS L1 SNR data for the given SNR file, with time on the x-axis column 1 on the x-axis and SNR data on the y-axis You have to set -sat or it will not work. For a specific satellite number, provide that instead of gps. The other allowed x-axis option is elevation which is short for elevation angle.

quickplt txtfile 1 16 -xlabel Time
: would plot column 1 on the x-axis and column 16 on the y-axis and add Time on the x-axis label

quickplt txtfile 1 16 -xlabel "Time (sec)"
: would plot column 1 on the x-axis and column 16 on the y-axis and add Time (sec) on the x-axis label, but need quote marks since you have spaces in the x-axis labe.

quickplt txtfile 1 16 -reverse T
: would plot column 1 on the x-axis and column 16 on the y-axis it would reverse the y-axis parameter as you might want if you are ploting RH but want it to have the same sense as a tide gauge.

quickplt txtfile 1 3 -errorcol 4
: would plot error bars from column 4

quickplt txtfile 1 3 -errorcol 4 -ydoy T
: would plot error bars from column 4 and assume columns 1 and 2 are year and day of year

quickplt txtfile 1 4 -mjd T
: assume column 1 is modified julian day

quickplt txtfile 1 16 -ylimits 0 2
would restrict y-axis to be between 0 and 2

quickplt txtfile 1 16 -outfile myfile.png
would save png file to $REFL_CODE/Files/myfile.png

quickplt snrfile 4 7
Assuming you submitted a standard SNR file, it would plot the L1 data (column 7) vs time (seconds of the day in column 4). Zeroes would be removed, but you can toggle that to keep it.

quickplt snrfile 2 7
Assuming you submitted a standard SNR file, it would plot the L1 data (column 7) vs elevation angle (degrees in column 2).

quickplt snrfile 4 8 -sat 25
Assuming you submitted a standard SNR file, it would plot the data for satellite 25 for L2 data (column 8) vs time (seconds of the day in column 4).

quickplt snrfile 4 8 -sat 22
Assuming you submitted a standard SNR file, it would plot the data for Glonass satellite 22 for L2 data (column 8) vs time (seconds of the day in column 4).

Parameters

- **filename** (*str*) – name of file to be plotted
- **xcol** (*str*) – column number in the file for the x-axis parameter for snrfiles, you can say time or elevation
- **ycol** (*str*) – column number in the file for the y-axis parameter for snrfiles, you can say L1, L2, or L5
- **errorcol** (*int, optional*) – column number for the error bars
- **mjd** (*bool, optional*) – code will convert MJD to datetime (for xcol)
- **xlabel** (*str, optional*) – label for x-axis
- **ylabel** (*str*) – label for y-axis
- **symbol** (*str, optional*) – prescribe the marker used in the plot . It can include the color, i.e. 'b.' or 'b^'
- **reverse** (*bool, optional*) – to reverse y-axis limits which is helpful for when you are showing RH results but want the tides to be shown.
- **title** (*str, optional*) – title for plot
- **outfile** (*str, optional*) – name of png file to store plot
- **xlimits** (*list of str,*) – xaxis limits. if you selected any time options (ymd, mjd, ydoy,mjd), the code assumes a format of yyyyMMdd (year,month,day) for xlimits
- **ylimits** (*list of floats, optional*) – yaxis limits
- **ydoy** (*bool, optional*) – if columns 1 and 2 are year and doy, the x-axis will be plotted in obstimes you should select column 1 to plot
- **ymd** (*bool, optional*) – if columns 1,2,3 are year, month, date. So meant for plots with daily measurements - not subdaily.
- **ymdhm** (*bool, optional*) – if columns 1-5 are Y,M,D,H,M then x-axis will be plotted in obstimes

- **filename2** (*str*) – in principle this allows you to make plots from two files with identical formatting but I am not sure that it one hundred percent always works
- **freq** (*int, optional*) – use column 11 to find (and extract) a single frequency
- **utc_offset** (*int, optional*) – offset time axis by this number of hours (for local time) this only is used when the mjd option is used
- **yoffset** (*float*) – add or subtract to the y-axis values
- **keepzeros** (*bool, optional*) – keep/remove zeros, default is to remove
- **sat** (*str*) – satellite number for SNR file plotting for all gps satellites, say gps instead of a number similar for glonass, galileo, and beidou
- **yoffset2** (*float*) – add or subtract to the y-axis values in filename2
- **scale** (*float*) – multiply all y-axis values in file 1 by this value
- **scale2** (*float*) – multiply all y-axis values in file 2 by this value
- **elimits** (*list of floats*) – if SNR file is plotted, elevation angle limits are applied
- **azlimits** (*list of floats*) – if SNR file is plotted, azimuth angle limits are applied
- **plt** (*bool*) – whether you want the plot to be displayed on the screen. png file is always created.

gnssrefl.read_snr_files module

gnssrefl.read_snr_files.**load_snr_time_filtered**(*obsfile*, *sec_min=None*, *sec_max=None*)

Load an SNR file, parsing only rows within a seconds-of-day window.

Decompresses the full file (unavoidable for gzip), but only passes the matching rows to np.loadtxt, which is where most of the time is spent.

Parameters

- **obsfile** (*str or Path*) – Path to SNR file (plain text or .gz)
- **sec_min** (*float or None*) – Keep rows with seconds > sec_min. None means no lower bound.
- **sec_max** (*float or None*) – Keep rows with seconds < sec_max. None means no upper bound.

Returns
data

Return type
numpy array (N x cols) or None if no matching rows

gnssrefl.read_snr_files.**read_snr**(*obsfile*, *buffer_hours=0*, *screenstats=False*)

Load the contents of a SNR file into a numpy array, optionally including data from adjacent days.

Parameters

- **obsfile** (*str*) – name of the snrfile
- **buffer_hours** (*float, optional*) – hours of data to include from adjacent days. If > 0, reads last buffer_hours from previous day and first buffer_hours from next day. Time tags are adjusted: prev day uses negative seconds, next day uses seconds > 86400. Default is 0 (single day only).

- **screenstats** (*bool, optional*) – print verbose information about buffer data loading. Default is False.

Returns

- **allGood** (*int*) – 1, file was successfully loaded, 0 if not. apparently this variable was defined when I did not know about booleans. . . .
- **f** (*numpy array*) – contents of the SNR file
- **r** (*int*) – number of rows in SNR file
- **c** (*int*) – number of columns in SNR file

gnssrefl.refl_zones module

gnssrefl.refl_zones.**FresnelZone**(*f*, *e*, *h*)

based on GPS Tool Box Roesler and Larson (2018). Original source is Felipe Nievinski as published in the appendix of Larson and Nievinski 2013 this code assumes a horizontal, untilted reflecting surface

Parameters

- **f** (*int*) – frequency (1,2, or 5)
- **e** (*float*) – elevation angle (deg)
- **h** (*float*) – reflector height (m)

Returns

firstF – [a, b, R] in meters where: a : is the semi-major axis, aligned with the satellite azimuth b : is the semi-minor axis R : locates the center of the ellispe on the satellite azimuth direction (theta)

Return type

list of floats

gnssrefl.refl_zones.**calcAzEl_new**(*prn*, *newf*, *recv*, *u*, *East*, *North*)

function to gather azel for all low elevation angle data this is used in the reflection zone mapping tool

Parameters

- **prn** (*int*) – satellite number
- **newf** (*3 vector of floats*) – cartesian coordinates of the satellite (meters)
- **recv** (*3 vector of floats*) – receiver coordinates (meters)
- **u** (*3 vector*) – cartesian unit vector for up
- **East** (*3 vector*) – cartesian unit vector for east direction
- **North** (*3 vector*) – cartesian unit vector for north direction

Returns

tv – list of satellite tracks [prn number, elevation angle, azimuth angle]

Return type

numpy array of floats

gnssrefl.refl_zones.**calcAzEl_newish**(*prn*, *newf*, *recv*, *u*, *East*, *North*)

should be consolidated with the other function.but who has the time!

Parameters

- **prn** (*int*) – satellite number ?

- **newf** –
- **u** (*numpy array*) – up unit vector
- **East** –
- **North** –

Returns
tv

Return type
numpy array

gnssrefl.refl_zones.**makeEllipse_latlon**(*freq*, *el*, *h*, *azim*, *latd*, *lngd*)

for given fresnel zone, produces coordinates of an ellipse

Parameters

- **freq** (*int*) – frequency
- **el** (*float*) – elevation angle in degrees
- **h** (*float*) – reflector height in meters
- **azim** (*float*) – azimuth in degrees
- **latd** (*float*) – latitude in degrees
- **lngd** (*float*) – longitude in degrees

Returns

- **lngdnew** (*float*) – new longitudes in degrees
- **latdnew** (*float*) – new latitudes in degrees

gnssrefl.refl_zones.**makeFresnelEllipse**(*A*, *B*, *center*, *azim*)

make an Fresnel zone given size, center, and orientation

Parameters

- **A** (*float*) – semi-major axis of ellipse in meters
- **B** (*float*) – semi-minor axis of ellipse in meters
- **center** (*float*) – center of the ellipse, provided as distance along the satellite azimuth direction
- **azimuth** (*float*) – azimuth angle of ellipse in degrees. this will be clockwise positive as defined from north

Returns

- **x** (*numpy array of floats*) – x value of cartesian coordinates of ellipse
- **y** (*numpy array of floats*) – y value of cartesian coordinates of ellipse
- **xcenter** (*float*) – x value for center of ellipse in 2-d cartesian
- **ycenter** (*float*) – y value for center of ellipse in 2-d cartesian

gnssrefl.refl_zones.**make_FZ_kml**(*station*, *filename*, *freq*, *el_list*, *h*, *lat*, *lng*, *azlist*)

makes fresnel zones for given azimuth and elevation angle lists.

Parameters

- **station** (*str*) – four character station name

- **filename** (*str*) – output filename (the kml extension should already be there)
- **freq** (*int*) – frequency (1,2, or 5)
- **el_list** (*list of floatss*) – elevation angles
- **h** (*float*) – reflector height in meters
- **lat** (*float*) – latitude in deg
- **lng** (*float*) – longitude in degrees
- **azlist** (*list of floats*) – azimuths

gnssrefl.refl_zones.**nyquist_simple**(*t*, *elev*, *azims*, *emin*, *emax*, *azriseset*, *reqsamplerate*)

given numpy array of elevation angles (elev) and limtis (emin,emax) and azriseset, reqsamplerate

Parameters

- **t** – cannot remember
- **elev** (*numpy array of floats*) – elevation angle of rising/setting arcs
- **azims** (*numpy array of floats*) – azimuths of rising/setting arcs
- **emin** (*float*) – minimum elevation angle, degrees
- **emax** (*float*) – maximum elevation angle, degrees
- **azriseset** – cannot remember
- **reqsamplerate** (*float*) – requested receiver sampling rate

gnssrefl.refl_zones.**rising_setting_new**(*recv*, *el_range*, *obsfile*)

Calculates potential rising and setting arcs

Parameters

- **recv** (*list of floats*) – Cartesian coordinates of station in meters
- **el_range** (*list of floats*) – elevation angles in degrees
- **obsfile** (*str*) – orbit filename

Returns

azlist – each line has azimuth angle, PRN, elevation angle in that order angles are in units of degrees

Return type

list of floats

gnssrefl.refl_zones.**save_reflzone_orbits**()

check that orbit files exist for reflection zone code. downloads to $REFL_CODE$/Files directory if needed

Returns

foundfiles – whether needed files were found

Return type

bool

gnssrefl.refl_zones.**set_azlist_multi_regions**(*sectors*, *azlist*)

edits initial azlist to restrict to given azimuth sectors. assumes that illegal list of sectors have been checked (i.e. no negative azimuths, they should be pairs, and increasing)

Parameters

- **sectors** (*list of floats*) – min and max azimuth (degrees). Must be in pairs, no negative numbers
- **azlist** (*list of floats*) – list of tracks, [azimuth angle, satNumber, elevation angle]

Returns
azlist2 – same format as before, but with azimuths removed outside the restricted zones

Return type
list of floats

gnssrefl.refl_zones.**set_final_azlist**(*a1ang*, *a2ang*, *azlist*)

edits initial azlist to restrict to given azimuths

Parameters

- **a1ang** (*float*) – minimum azimuth (degrees)
- **a2ang** (*float*) – maximum azimuth (degrees)
- **azlist** (*list of floats*) – list of tracks, [azimuth angle, satNumber, elevation angle]

Returns
azlist

Return type
list of floats

gnssrefl.refl_zones.**set_system**(*system*)

finds the file needed to compute orbits for reflection zones

Parameters

- **system** (*str*) – gps,glonass,beidou, or galileo
- **it** (*int*) – simple pointer from former code. 1 is GPS, 2 is Glonass etc

Returns
orbfile – orbit filename with Cartesian coordinates for one day

Return type
str

gnssrefl.refl_zones.**write_coords**(*lng*, *lat*)

Parameters

- **lng** (*list of floats*) – longitudes in degrees
- **lat** (*list of floats*) – latitudes in degrees

Returns
points – for google maps

Return type
list of pairs of long/lat

gnssrefl.refl_zones_cl module

gnssrefl.refl_zones_cl.**main**()

gnssrefl.refl_zones_cl.**parse_arguments**()

gnssrefl.refl_zones_cl.**reflzones**(*station: str*, *azim1: int = 0*, *azim2: int = 360*, *lat: float | None = None*, *lon: float | None = None*, *height: float | None = None*, *RH: float | None = None*, *fr: int = 1*, *el_list: float = []*, *azlist: float = []*, *system: str = 'gps'*, *output: str | None = None*)

This module creates "stand-alone" Fresnel Zones maps for google Earth. At a minimum it requires a four station character name as input. The output is a KML file.

If the station is in either your local coordinate file or in the UNR database that comes with gnssrefl, those latitude, longitude, and ellipsoidal height values are used. For information on the local coordinate file see the file formats section of the documentation. You may override those stored values with the optional lat lon height inputs.

The output file will be stored in REFL_CODE/Files/kml unless you specify an output name. In that case it will go into your working directory

The defaults are that it does all azimuths, elevation angles of 5-10-15, GPS L1, and uses the height of the station above sea level to use for the RH. If you want to specify the reflector height, set -RH. If you are making a file for an interior lake or river, you will need to use this option. Similarly, for a soil moisture or snow reflection zone map, where height above sea level is not important, you will want to set the RH value accordingly.

Examples

refl_zones p041 -RH 2
standard Fresnel zones for all azimuths, GPS and L1 frequency, RH of 2 meters

refl_zones sc02 -fr 2 -azlist 40 240
Fresnel zones for limited azimuths, GPS and L2 frequency. RH is mean sea level

refl_zones p041 -RH 5 -system galileo
Fresnel zones for reflector height of 5 meters and Galileo L1

refl_zones xxxx -RH 5 -lat 40 -lon 120 -height 10
Using a station not in the UNR database, so station position given Note that this is the ellipsoidal height.

Parameters

- **station** (*str*) – station name
- **azim1** (*int, optional*) – min azimuth angle in deg
- **azim2** (*int, optional*) – max azimuth angle in deg
- **lat** (*float, optional*) – latitude in deg
- **lon** (*float, optional*) – longitude in deg
- **height** (*float, optional*) – ellipsoidal height in m
- **RH** (*float, optional*) – user-defined reflector height (m) default is to use sea level as the RH
- **fr** (*int, optional*) – frequency (only 1,2, or 5 allowed)
- **el_list** (*list of floats, optional*) – elevation angles desired (deg) default is 5, 10, 15

- **azlist** (*list of floats, optional*) – azimuth angle regions (deg) Must be in pairs, i.e. 0 90 180 270
- **system** (*str, optional*) – name of constellation (gps,glonass,galileo, beidou allowed) default is gps
- **output** (*str, optional*) – name for kml file

Return type
Creates a KML file for Google Earth

gnssrefl.refraction module

written in python from from original TU Vienna codes for GMF

gnssrefl.refraction.**Equivalent_Angle_Corr_NITE**(*Hr_apr*, *e_T*, *N_ant*, *ztd_ant*, *mpf_tot*, *dmpf_de_tot*)

This function computes the "equvilent" angular correction to apply the NITE formula on the true elevation angle ele_eqv = e_T + dele

Equation (24) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

The variable substitude method can be found in Strandberg, J. (2020). New methods and applications for interferometric GNSS reflectometry. Chalmers Tekniska Hogskola (Sweden).

Parameters

- **Hr_apr** (*float*) – approximate a-priori reflector height, in meters
- **e_T** (*float*) – satellite true elevation angle in degree
- **N_ant** (*float*) – atmospheric refractivity at the GNSS antenna, in ppm
- **ztd_ant** (*float*) – zenith total delay at the antenna, in meters
- **mpf_tot** (*float*) – total mapping function for this elevation angle
- **dmpf_de_tot** (*float*) – derivative of the mapping function over elevation angle

Returns
dele – equvilent angular correction in degrees

Return type
float

gnssrefl.refraction.**Equivalent_Angle_Corr_mpf**(*ele*, *mpf_tot*, *N0*, *Hr_apr*)

This function computes the "equvilent" angular correction to apply the tropospheric delay calculated with the mapping function.

See: Williams, S. D. P., & Nievinski, F. G. (2017). Tropospheric delays in ground-based GNSS multipath reflectometry—Experimental evidence from coastal sites. Journal of Geophysical Research: Solid Earth, 122(3), 2310-2327.

Strandberg, J. (2020). New methods and applications for interferometric GNSS reflectometry. Chalmers Tekniska Hogskola (Sweden).

Parameters

- **ele** (*float*) – true elevation angle in degrees
- **mpf_tot** (*float*) – total mapping function, units?
- **N0** (*float*) – refractivity at GNSS antenna in part-per-million
- **Hr_apr** (*float*) – approximate reflector height in meters

Returns
dele – equvilent angular correction in degrees

Return type
float

gnssrefl.refraction.**Hv_Hr_ratio**(*Hr*, *Re*, *e_A*)

This function computes the ratio between the "vertical height difference between the antenna and the refection point" and the "reflector height", assuming a sperical reflector (ocean)

See equation (23) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

Parameters

- **Hr** (*float*) – approximate reflector height in meters (height difference between the antenna and the reflecting surface)
- **Re** (*float*) – (Gaussian) radius of the Earth in meters
- **e_A** (*float*) – apparent elevation angle at the antenna, in degree

Returns
the_ratio – ratio, allways bigger than 1

Return type
float

gnssrefl.refraction.**N_layer**(*N_antenna*, *Hr*)

Computes average refractivity of the top (antenna) and bottom (reflecting surface) of this layer

See Equation (14) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

Parameters

- **N_antenna** (*float*) – refractivity at the antenna in ppm
- **Hr** (*float*) – reflector height in meters (height difference between the antenna and the reflecting surface)

Returns
Nl – average refractivity in ppm in this layer

Return type
float

gnssrefl.refraction.**Ulich_Bending_Angle**(*ele*, *N0*, *station_config*, *p*, *T*, *ttime*, *sat*)

Ulich, B. L. "Millimeter wave radio telescopes: Gain and pointing characteristics." (1981)

Author: 20220629, fengpeng

Modified by KL to use numpy so I can use arrays. I do not know why all these extra input parameters are here.

Parameters

- **ele** (*numpy array of floats*) – true elevation angle, degrees
- **N0** (*float*) – antenna refractivity in ppm
- **station_config** (*dict*) –
- **p** (*float*) – pressure, units?
- **T** (*float*) – temperature, units?
- **ttime** –
- **sat** –

Returns
De – corrected elevation angles, deg

Return type
numpy array of floats

gnssrefl.refraction.**Ulich_Bending_Angle_original**(*ele*, *N0*)

This function computes the atmospheric bending angle with the Ulich equation.

Equation (18) in Ulich, B. L. (1981). Millimeter wave radio telescopes: Gain and pointing characteristics. International Journal of Infrared and Millimeter Waves, 2, 293-310.

Parameters

- **ele** (*float*) – true elevation angle in degrees
- **N0** (*float*) – refractivity in part-per-million

Returns
dele – bending angle (angular difference between apparent and true elevation angle), in degrees

Return type
float

gnssrefl.refraction.**asknewet**(*e*, *Tm*, *lambda_val*)

Determines the zenith wet delay based on the equation 22 by Askne and Nordius (1987)

Askne and Nordius, Estimation of tropospheric delay for microwaves from surface weather data, Radio Science, Vol 22(3): 379-386, 1987.

Source: Peng Feng

Parameters

- **e** (*float*) – water vapor pressure in hPa
- **Tm** (*float*) – mean temperature in Kelvin
- **lambda_val** (*float*) – water vapor lapse rate (see definition in Askne and Nordius 1987)

Returns
zwd – zenith wet delay in meter

Return type
float

gnssrefl.refraction.**corr_el_angles**(*el_deg*, *press*, *temp*)

Corrects elevation angles for refraction using simple angle bending model

Parameters

- **el_deg** (*numpy array of floats*) – elevation angles in degrees
- **press** (*float*) – pressure in hPa
- **temp** (*float*) – temperature in degrees C

Returns
corr_el_deg – corrected elevation angles (in degrees)

Return type
numpy array of floats

gnssrefl.refraction.**correct_elevations**(*ele*, *station_config*, *year*, *doy*, *verbose=True*)

Apply refraction correction to elevation angles.

Reads the GPT2 grid for the station location, computes atmospheric parameters, and dispatches to the selected refraction model (Bennett, Ulich, NITE, or MPF). Models 5 and 6 (NITE/MPF) remove observations below 1.5 degrees elevation where the correction diverges.

Parameters

- **ele** (*numpy array of floats*) – elevation angles in degrees
- **station_config** (*dict*) – station analysis parameters (from json). Required keys: station, lat, lon, ht, refraction. Optional: refr_model, apriori_rh.
- **year** (*int*) – calendar year
- **doy** (*int*) – day of year

Returns

- **corrected_ele** (*numpy array of floats*) – refraction-corrected elevation angles in degrees
- **valid_mask** (*numpy array of bools*) – True for each element that survived filtering (some models remove low-elevation data). Has same length as the *input* `ele`.

gnssrefl.refraction.**dH_curve**(*Hr*, *Re*, *e_A*)

Computes vertical displacement of the reflection point vs. that of a "planar reflection"

See Equation (7) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

Parameters

- **Hr** (*float*) – reflector height in meters (height difference between the antenna and the reflecting surface)
- **Re** (*float*) – (Gaussian) radius of the Earth in meters
- **e_A** (*float*) – apparent elevation angle at the antenna, in degrees

Returns
dH – vertial displacement of the reflection point in meters

Return type
float

gnssrefl.refraction.**dmpf_dh**(*ele*, *dhgt*)

Station height correction of the hydrostatic mapping function (Niell, 1996) This is translated from Johannes Boehm's vmf1_ht.f Fortran code

Niell, A. E. (1996). Global mapping functions for the atmosphere delay at radio wavelengths. Journal of geophysical research: solid earth, 101(B2), 3227-3246.

Boehm, J., Werl, B., & Schuh, H. (2006). Troposphere mapping functions for GPS and very long baseline interferometry from European Centre for Medium-Range Weather Forecasts operational analysis data. JGR: Solid Earth, 111(B2).

Parameters

- **ele** (*float*) – true elevation angle in degree
- **dhgt** (*float*) – height difference in meters. In GNSS-IR, this is reflector height; in applying mapping function grid products, this is the height difference between the antenna and the grid point height

Returns
ht_corr – correction to the hydrostatic mapping function (vmf1h= vmf1h + ht_corr)

Return type
float

gnssrefl.refraction.**gmf_deriv**(*dmjd*, *dlat*, *dlon*, *dhgt*, *zd*)

This subroutine determines the Global Mapping Functions GMF and derivative. Translated from https://vmf.geo.tuwien.ac.at/codes/gmf_deriv.f by Peng Feng in March, 2023.

Johannes Boehm, 2005 August 30

ref 2006 Aug. 14: derivatives (U. Hugentobler) ref 2006 Aug. 14: recursions for Legendre polynomials (O. Montenbruck) ref 2011 Jul. 21: latitude -> ellipsoidal latitude

Parameters

- **dmjd** (*float*) – modified julian date
- **dlat** (*float*) – ellipsoidal latitude in radians
- **dlon** (*float*) – longitude in radians
- **dhgt** (*float*) – height in meters
- **zd** (*float*) – zenith distance in radians ??? (is this really what you mean?? KL: I suspect it is the zenith angle … in radians

Returns

- **gmfh(2)** (*float*) – hydrostatic mapping function and derivative wrt z
- **gmfw(2)** (*float*) – wet mapping function and derivative wrt z

gnssrefl.refraction.**gpt2_1w**(*station*, *dmjd*, *dlat*, *dlon*, *hell*, *it*)

Parameters

- **station** (*str*) – station name
- **dmjd** (*float*) – modified Julian date (scalar, only one epoch per call is possible)
- **dlat** (*float*) – ellipsoidal latitude in radians [-pi/2:+pi/2]
- **dlon** (*float*) – longitude in radians [-pi:pi] or [0:2pi]
- **hell** (*float*) – ellipsoidal height in m
- **it** (*integer*) – case 1: no time variation but static quantities

  case 0: with time variation (annual and semiannual terms)

Returns

- **p** (*float*) – pressure in hPa
- **T** (*float*) – temperature in degrees Celsius
- **dT** (*float*) – temperature lapse rate in degrees per km
- **Tm** (*float*) – mean temperature of the water vapor in degrees Kelvin
- **e** (*float*) – water vapor pressure in hPa
- **ah** (*float*) – hydrostatic mapping function coefficient at zero height (VMF1)
- **aw** (*float*) – wet mapping function coefficient (VMF1)
- **la** (*float*) – water vapor decrease factor

- **undu** (*float*) – geoid undulation in m

gnssrefl.refraction.**look_for_pickle_file**()

latest attempt to solve the dilemma of the pickle file needed for the refraction correction

Returns

- **foundit** (*bool*) – whether pickle file found
- **fullpname** (*str*) – full path to the pickle file

gnssrefl.refraction.**mpf_tot**(*gmf_h*, *gmf_w*, *zhd*, *zwd*)

Finds the total mapping function by weighting the hydrostatic and wet mapping function with the zenith hydrostatic and wet delay.

Author: Peng Feng

Parameters

- **gmf_h** (*float*) – hydrostatic mapping function
- **gmf_w** (*float*) – wet mapping function
- **zhd** (*float*) – zenith hydrostatic delay in meters
- **zwd** (*float*) – zenith wet delay in meters

Returns
mpf_tot1 – total mapping function

Return type
float

gnssrefl.refraction.**readWrite_gpt2_1w**(*xdir*, *station*, *site_lat*, *site_lon*)

makes a grid for refraction correction

Parameters

- **xdir** (*str*) – directory for output
- **station** (*str*) – station name, 4 ch
- **lat** (*float*) – latitude in degrees
- **lon** (*float*) – longitude in degrees

gnssrefl.refraction.**read_4by5**(*station*, *dlat*, *dlon*, *hell*)

reads existing grid points for a given location

Parameters

- **station** (*string*) – name of station
- **dlat** (*float*) – latitude in degrees
- **dlon** (*float*) – longitude in degrees
- **hell** (*float*) – ellipsoidal height in meters

Returns

- **pgrid** (*4 by 5 numpy array*) – pressure in hPa
- **Tgrid** (*4 by 5 numpy array*) – temperature in C
- **Qgrid** (*4 by 5 numpy array*)
- **dTgrid** (*4 by 5 numpy array*) – temperature lapse rate in degrees per km

- **u** (*4 by 1 numpy array*) – geoid undulation in meters
- **Hs** (*4 by 1 numpy array*)
- **ahgrid** (*4 by 5 numpy array*) – hydrostatic mapping function coefficient at zero height (VMF1)
- **awgrid** (*4 by 5 numpy array*) – wet mapping function coefficient (VMF1)
- **lagrid** (*4 by 5 numpy array*)
- **Tmgrid** (*4 by 5 numpy array*) – mean temperature of the water vapor in degrees Kelvin
- *requires that an environment variable exists for REFL_CODE*

gnssrefl.refraction.**refrc_Rueger**(*drypress*, *vpress*, *temp*)

Obtains refractivity index suitable for GNSS-IR

Rueger, Jean M. "Refractive index formulae for radio waves." Proceedings of the FIG XXII International Congress, Washington, DC, USA. Vol. 113. 2002.

Parameters

- **drypress** (*float*) – dry pressure hPa
- **vpress** (*float*) – vapor pressure in hPa
- **temp** (*float*) – temperature in Kelvin

Returns
ref – [Ntotal, Nhydro, Nwet], which are total, hydrostatic and wet refractivity in ppm

Return type
list of floats

gnssrefl.refraction.**saastam2**(*press*, *lat*, *height*)

This function computes the Zenith Hydrostatic Delay using the Saastamoinen model with updated refractivity equation from Rueger (2002)

Saastamoinen, J. (1972). Atmospheric corrections for the troposphere and stratosphere in radio ranging of satellites. The Use of Artificial Satellites for Geodesy, Geophysics Monograph Service, 15, 274-251.

Feng, P., Li, F., Yan, J., Zhang, F., & Barriot, J. P. (2020). Assessment of the accuracy of the Saastamoinen model and VMF1/VMF3 mapping functions with respect to ray-tracing from radiosonde data in the framework of GNSS meteorology. Remote Sensing, 12(20), 3337.

Parameters

- **press** (*float*) – atmospheric total pressure in hPa
- **lat** (*float*) – latitude of the station, degrees
- **height** (*float*) – ellipsoidal height of the station in meters

Returns
zhd – zenith hystostatic delay in meters

Return type
float

gnssrefl.refraction.**sita_Earth**(*Hr*, *e_A*)

This function computes the angular separation of the antenna and the reflection point in earth surface, view from earth center

See Equation (7) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

Parameters

- **Hr** (*float*) – reflector height in meters (height difference between the antenna and the reflecting surface)
- **e_A** (*float*) – apparent elevation angle at the antenna, in degrees

Returns
sita_E – earth center angle in degrees

Return type
float

gnssrefl.refraction.**sita_Satellite**(*Hr*, *e_A*)

This function computes the angle formed by the antenna-satellite line-of-sight and the reflection point-satellite LoS

Equation (8) in Peng (2023), DOI: 10.1109/TGRS.2023.3332422

Parameters

- **Hr** (*float*) – reflector height in meters (height difference between the antenna and the reflecting surface)
- **e_A** (*float*) – apparent elevation angle at the antenna, in degree

Returns
sita_S – satellite angle in degrees (small for MEO satellites)

Return type
float

gnssrefl.retrieve_rh module

gnssrefl.retrieve_rh.**retrieve_rh**(*station*, *year*, *doy*, *extension*, *station_config*, *arcs*, *screenstats*, *irefr*, *logid*, *logfilename*, *dbhz*)

new worker code that estimates LSP from GNSS SNR data. it will now live here and be called by gnssir_v2.py

Parameters

- **station** (*str*) – name of station
- **year** (*int*) – calendar year
- **doy** (*int*) – day of year
- **extension** (*str*) – strategy extension
- **station_config** (*dict*) – inputs to LSP analysis
- **arcs** (*list of (metadata, data) tuples*) – pre-extracted satellite arcs from extract_arcs_from_station
- **screenstats** (*bool*) – whether you want stats to the screen
- **irefr** (*int*) – which refrction model is used
- **logid** (*file ID*) – opened in earlier function
- **logfilename** (*str*) – name of the log file …
- **dbhz** (*bool*) – keep dbhz units (or not)

gnssrefl.rh_plot module

gnssrefl.rh_plot.**main**()

gnssrefl.rh_plot.**parse_arguments**()

gnssrefl.rh_plot.**rh_plot**(*station: str*, *year: int*, *csvfile: bool = False*, *plt: bool = True*, *extension: str = ''*, *doy1: int = 1*, *doy2: int = 366*, *ampl: float = 0*, *h1: float = 0.0*, *h2: float = 300.0*, *azim1: int = 0*, *azim2: int = 360*, *peak2noise: float = 0*)

Parameters

- **station** (*string*) – 4 character id of the station.
- **year** (*integer*) – Year
- **csvfile** (*boolean, optional*) – Set to True if you prefer csv to plain txt. default is False.
- **plt** (*boolean, optional*) – To print plots to screen or not. default is TRUE.
- **extension** (*string, optional*) – Solution subdirectory. default is empty string.
- **doy1** (*integer, optional*) – Initial day of year default is 1.
- **doy2** (*integer, optional*) – End day of year. default is 366.
- **ampl** (*float*) – New amplitude constraint default is 0.
- **azim1** (*int, optional*) – New min azimuth default is 0.
- **azim2** (*int, optional*) – New max azimuth default is 360.
- **h1** (*float optional*) – lowest allowed reflector height default is 0
- **h2** (*float optional*) – highest allowed reflector height default is 300
- **peak2noise** (*float, optional*) – New peak to noise constraint default is 0.

gnssrefl.rinex2snr module

gnssrefl.rinex2snr.**conv2snr**(*year*, *doy*, *station*, *option*, *orbtype*, *receiverrate*, *dec_rate*, *archive*, *log*, ***kwargs*)

This code originally picked up and translated files. You can now optionally send a parameter called rinex2_filename if the file exists. This needs to be done especially for RINEX 3 which have been converted to RINEX 2.11

2024 March 29: change location of logs directory to below REFL_CODE

2024 October 16: logs are created earlier now and locations are sent to this file

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **option** (*int*) – snr choice (66, 99 etc)
- **orbtype** (*str*) – orbit source (nav, gps, gnss, etc)
- **receiverrate** (*int*) – sampling interval of the GPS receiver, e.g. 1, 30, 15
- **dec_rate** (*int*) – decimation value to reduce file size
- **archive** (*str*) – external location (archive) of the rinex files

- **log** (*fileid*) – for screen messages

gnssrefl.rinex2snr.**extract_snr**(*prn*, *con*, *obslist*, *obsdata*, *prntoidx*, *not_ij*, *emp*)

gnssrefl.rinex2snr.**get_local_rinexfile**(*rfile*, *localpath2*)

look for a plain or gzipped version of the RINEX 2.11 file in the year subdirectories copies it to the local directory. this method stops the code from deleting your rinex files. As of 2023 September 19, it should also look for Hatanaka files.

localpath2 = os.environ['REFL_CODE'] + '/' + cyyyy + '/rinex/' + station + '/'

This is unlikely to work for uppercase RINEX files. Try the mk option

Parameters

- **rfile** (*str*) – version2 rinexfile name
- **localpath2** (*str*) – another location of the file (meant to be as defined above)

Returns
allgood – whether file found

Return type
bool

gnssrefl.rinex2snr.**go_from_crxgz_to_rnx**(*c3gz*, *deletecrx=True*)

checks to see if rinex3 file exists, gunzip if necessary, run hatanaka, if necessary

Parameters

- **c3gz** (*str*) – filename for a gzipped RINEX 3 Hatanaka file
- **bool** (*deletecrx =*) – whether to delete the crx file

Returns

- **translated** (*bool*) – if file successfully found and available
- **rnx** (*str*) – name of gunzipped and decompressed RINEX 3

gnssrefl.rinex2snr.**makan_warning**(*missing*, *f*)

Writes a warning for the makan option which checks for non-standard RINEX file names

Parameters

- **missing** (*bool*) – whether file is missing
- **f** (*str*) – filename as defined by Makan

gnssrefl.rinex2snr.**print_archives**()

feeble attempt to print list of archives to screen ...

gnssrefl.rinex2snr.**quickname**(*station*, *year*, *cyy*, *cdoy*, *csnr*)

creates full filename (including directory) for a local SNR file

Parameters

- **station** (*str*) – station name, 4 character
- **year** (*int*) – full year
- **cyy** (*str*) – two character year
- **cdoy** (*str*) – three character day of year
- **csnr** (*str*) – snr ending, i.e. '66' or '99'

Returns
fname – full filename including the directory

Return type
str

gnssrefl.rinex2snr.**readSNRval**(*s1exist*, *s2exist*, *s5exist*, *observationdata*, *prntoidx*, *sat*, *i*)

what it looks like only reads GPS data for now interface between Joakim's code and mine …

Parameters

- **s1exist** (*boolean*) –
- **s2exist** (*boolean*) –
- **s5exist** (*boolean*) –

Returns

- *s1*
- *s2*
- *s5*

gnssrefl.rinex2snr.**rnx2snr**(*obsfile*, *navfile*, *snrfile*, *snroption*, *year*, *month*, *day*, *dec_rate*, *log*)

Converts a rinex v2.11 obs file using Joakim's rinex reading code

Parameters

- **obsfile** (*str*) – RINEX 2.11 filename
- **navfile** (*str*) – navigation file
- **snrfile** (*str*) – SNR filename
- **snroption** (*integer*) – kind of SNR file requested
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **dec_rate** (*int*) – decimation rate in seconds

gnssrefl.rinex2snr.**rnx2snr_v3**(*obsfile*, *navfile*, *snrfile*, *snroption*, *year*, *month*, *day*, *dec_rate*, *log*)

Converts a RINEX 3 obs file directly, bypassing gfzrnx conversion to RINEX 2.

Parameters

- **obsfile** (*str*) – RINEX 3 filename
- **navfile** (*str*) – navigation/orbit file
- **snrfile** (*str*) – SNR output filename
- **snroption** (*int*) – kind of SNR file requested
- **year** (*int*) – full year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **dec_rate** (*int*) – decimation rate in seconds
- **log** (*file*) – log file handle

gnssrefl.rinex2snr.**run_rinex2snr**(*station*, *year*, *doy*, *isnr*, *orbtype*, *rate*, *dec_rate*, *archive*, *nol*, *overwrite*, *srate*, *mk*, *stream*, *strip*, *bkg*, *screenstats*, *gzip*, *timeout*, *quiet*)

main code to convert RINEX files into SNR files. It works on a single year and doy.

Parameters

- **station** (*str*) – 4 or 9 character station name. 6 ch allowed for japanese archive 9 means it is a RINEX 3 file
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **isnr** (*int*) – SNR file type choice, i.e. 66, 88
- **orbtype** (*str*) – orbit type, e.g. nav, rapid, gnss
- **rate** (*str*) – general sample rate. high: use 1-Hz area in the archive low: use default area in the archive
- **dec_rate** (*integer*) – decimation value
- **archive** (*str*) – choice of GNSS archive, of which there are many
- **nol** (*bool*) – True: assumes RINEX files are in local directory False (default): will look at multiple - or specific archive
- **overwrite** (*bool*) – False (default): if SNR file exists, SNR file not made True: make a new SNR file
- **srate** (*int*) – sample rate for RINEX 3 files
- **mk** (*bool*) – makan option
- **stream** (*str*) – naming parameter for RINEX 3 files (R or S)
- **strip** (*bool*) – reduces observables to only SNR (too many observables, particularly in RINEX 2 files will break the RINEX translator)
- **bkg** (*str*) – location of bkg files, EUREF or IGS
- **screenstats** (*bool*) – whether print statements come to screen
- **gzip** (*bool*) – whether SNR files are gzipped after creation
- **timeout** (*int*) – optional parameter I am testing out for requests timeout parameter in seconds
- **quiet** (*bool*) – whether gfzrnx messages are printed to the screen (T) or suppressed (F)

gnssrefl.rinex2snr.**satorb**(*week*, *sec_of_week*, *ephem*)

Calculate GPS satellite orbits

Parameters

- **week** (*integer*) – GPS week
- **sec_of_week** (*float*) – GPS seconds of the week
- **ephem** (*ephemeris block*) –

Returns

the x,y,z, coordinates of the satellite in meters and relativity correction (also in meters), so you add, not subtract

Return type

numpy array

gnssrefl.rinex2snr.**satorb_prop**(*week*, *secweek*, *prn*, *rrec0*, *closest_ephem*)

Calculates and returns geometric range (in metres) given time (week and sec of week), prn, receiver coordinates (cartesian, meters) this assumes someone was nice enough to send you the closest ephemeris returns the satellite coordinates as well, so you can use htem in the A matrix

Parameters

- **week** (*integer*) – GPS week
- **secweek** (*integer*) – GPS second of the week
- **prn** (*integer*) – satellite number
- **rrec0** (*3vector*) – receiver coordinates, meters

Returns

SatOrbn – floats, Cartesian location of satellite in meters [x,y,z]

Return type

3vector

gnssrefl.rinex2snr.**satorb_prop_sp3**(*iX*, *iY*, *iZ*, *recv*, *Tp*, *ij*)

for satellite number prn and receiver coordinates rrec0 find the x,y,z coordinates at time secweek

Parameters

- **iX** (*float*) –
- **iY** (*float*) –
- **iZ** (*float*) –
- **recv** (*3 vector, float*) –
- **Tp** –
- **ij** –
- **it** (*sp3 has the orbit information in*) –

gnssrefl.rinex2snr.**set_rinex2snr_logs**(*station*, *year*, *doy*)

Open a file for translation log messages. Return the file ID and the log path.

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **log** (*file ID*) – translation log
- **general_log** (*str*) – path of the log file

gnssrefl.rinex2snr.**the_makan_option**(*station*, *cyyyy*, *cyy*, *cdoy*)

this code checks many versions of RINEX files (Z, gz, regular, hatanaka) both in the working directory and in an external rinex area $REFL_CODE/rinex/station/year

turns whatever it finds into a regular RINEX file in the working directory that file WILL be deleted, but it will not delete those stored externally.

Parameters

- **station** (*str*) – station name (4 ch)

- **cyyyy** (*str*) – 4 ch year
- **cyy** (*str*) – two ch year
- **cdoy** (*str*) – three ch day of year

gnssrefl.rinex2snr.**write_snr_from_nav**(*navfile*, *obstimes*, *observationdata*, *obslist*, *prntoidx*, *gpssatlist*, *snrfile*, *s1exist*, *s2exist*, *s5exist*, *up*, *East*, *North*, *emin*, *emax*, *recv*, *dec_rate*, *log*)

Strandberg nav reading file?

Parameters

- **navfile** (*string*) –
- **obstimes** –
- **observationdata** –
- **obslist** –
- **prn2oidx** –
- **gpssatlist** –
- **snrfile** (*str*) – name of the output file
- **s1exist** –
- **s2exist** –
- **s5exist** –
- **!** (*This is for GPS only files*) –
- **format** (*navfile is nav broadcast ephemeris in RINEX*) –
- **info** (*inputs are rinex*) –
- **obstimes** –
- **observationdata** –
- **prntoidx** –
- **gpssatlist** –
- **existence** (*various bits about SNR*) –
- **name** (*snrfile is output*) –
- **file** (*log is for screen outputs - now going to a*) –

gnssrefl.rinex2snr.**write_snr_from_sp3**(*gpstime*, *sp3*, *systemsatlists*, *obsdata*, *obstypes*, *prntoidx*, *year*, *month*, *day*, *emin*, *emax*, *outputfile*, *up*, *East*, *North*, *recv*, *dec_rate*, *log*)

inputs are gpstime(numpy array with week and sow) sp3 is what has been read from the sp3 file columsn are satNu, week, sow, x, y, z (in meters) log is for comments

gnssrefl.rinex2snr_cl module

command line tool for the rinex2snr module it translates RINEX 2.11 and 3 files (computing azimuth and elevation angle) and stores these along with time and satellite number and SNR observations into SNR files

gnssrefl.rinex2snr_cl.**main**()

gnssrefl.rinex2snr_cl.**parse_arguments**()

gnssrefl.rinex2snr_cl.**rinex2snr**(*station: str*, *year: int*, *doy: int*, *snr: str | None = None*, *orb: str | None = None*, *rate: str = 'low'*, *dec: int = 0*, *nolook: bool = False*, *archive: str = 'all'*, *doy_end: int | None = None*, *year_end: int | None = None*, *overwrite: bool = False*, *samplerate: int = 30*, *stream: str = 'R'*, *mk: bool = False*, *weekly: bool = False*, *strip: bool = False*, *screenstats: bool = False*, *gzip: bool = True*, *monthly: bool = False*, *par: int | None = None*, *timeout: int = 0*, *extension: str = ''*, *debug: bool = False*, *quiet: bool = True*)

Note: rinex2snr means rinex TO snr. It is not a tool that is only meant for version 2 rinex files.

rinex2snr translates RINEX version 2.11 and 3+ files to a new file in the SNR format. This function will also fetch orbit files for you. RINEX obs files are provided either by the user or fetched from a long list of archives. Although RINEX 3 is supported, the default is RINEX 2.11 files. To tell the code you are using a RINEX 3 file, you should use a RINEX 3 station name, i.e. the 9 character version.

June 29, 2025 - tried to update to new folders for GFZ orbit products ...

New feature as of September 2024: various parameters can be stored in the station.json (created by gnssir_input). This is really just for convenience. Parameters are dec, snr, stream, samplerate, archive, and orb. Why did I add these? because I kept forgetting to set them on the command line! Right now you can add them to the json by hand, but most people will prefer to change them by using gnssir_input. Official documentation for these new inputs is defined in the gnssir_input documentation.

Parallel processing is now available. Set -par to a number <= 10. Some archives have been set to be non-compliant with this feature. Please look in the first few lines of the rinex2snr code to see the names of these archives.

In general, you should not make RINEX 2.11 files with a huge number of observables. Especially do not put Doppler data in your file. If you have more than 25 observables (multi-GNSS) or 20 (GPS only), the code should send an error message to a log. The location of that log is printed to the screen. If you don't want to remake your RINEX files, you can try the -strip T option, which I believe uses gfzrnx to strip out everything except for SNR data.

Real-time users should use ultra, wum2, or wum.

Default orbits are GPS only until day of year 137, 2021 when rapid GFZ orbits became available. If you still want to use the nav message, i.e. GPS only, you can request it by setting orb to nav or gps.

bkg no longer a boolean input - it must be specified with archive name, i.e. bkg-igs or bkg-euref

To analyze your own data you have to use the nolook option :

If you have the RINEX 2.11 file, the file can be in the local directory which is where you are running the code or it can be in $REFL_CODE/YYYY/rinex/ssss, where ssss is the lowercase directory name for your station. nolook now allows regular RINEX 2.11 obs files (ends in o) as well as as Hatanaka compressed (ends in d). The o or d file can be gzipped. The code also allows d files to be unix compressed which is how archives used to save these files.

If you are running the Docker, it can be a bit confusing to figure out where to put the files. Please see the discussion in the Docker installation section, as this is my best effort to help you with this.

Beyond that, you can try the -mk T option which searches other places, i.e. $REFL_CODE/rinex/ etc. I do not recommend that you use this option, but it is there. In general, you should use lowercase file names for RINEX 2.11 files.

If you have a RINEX3 file, you have to use the same naming convention as used by GNSS archive facilities. This means everything is capitalized except for the ending. The station name has 9 characters and various other parameters which can be quite confusing. Please see this page for the details.

https://gnssrefl.readthedocs.io/en/latest/pages/file_structure.html

I believe the code allows crx.gz, rnx, or rnx.gz endings in the local directory. It also checks the $REFL_CODE/YYYY/rinex directory for the crx.gz and rnx versions.

FAQ: what is rate and srate and why do you have both? rate tells the code which folder to use because archives always have files in different directories depending on sample rate. srate is for RINEX 3 files only because RINEX 3 has the sample rate on the filename itself (not just the directory). A RINEX 2.11 filename will not tell you which sample rate it is.

What is the stream parameter? It is a naming convention that is only used by RINEX 3 people. The allowed file types are S or R. I believe S stands for streamed.

RINEX3 30 second archives supported
bev, bkg-euref, bkg-igs, cddis, epn, ga, gfz, nrcan, sonel, gnet, nz

RINEX3 15 sec archives
bfg, unavco - You likely need to specify a 15 second sample rate. bfg requires a password. This archive has been turned off, 2025 Nov 28.

RINEX3 1 sec
bkg-igs, bkg-euref, cddis, ignes (spain), maybe nrcan?? , gnet

Examples

rinex2snr mchn 2018 15 -archive sopac
station mchn, year/doy 2022/15,sopac archive using GPS orbits

rinex2snr mchn 2022 15 -archive sopac
station mchn, year/doy 2022/15,sopac archive using multi-GNSS GFZ orbits

rinex2snr mchn 2022 15 -archive sopac -orb gps
station mchn, year/doy 2022/15,sopac archive using GPS orbits

rinex2snr mchn 2022 15 -orb rapid -archive sopac
now explicitly using rapid multi-GNSS orbits

rinex2snr mchn 2022 15 -orb rapid -archive sopac
now explicitly using final multi-GNSS orbits (includes Beidou)

rinex2snr mchn 2022 15 -orb rapid -archive sopac -overwrite T
have an SNR file, but you want to make a new one

rinex2snr p041 2022 15 -orb rapid -rate high -archive unavco
now using high-rate data from unavco and multi-GNSS orbits

rinex2snr p041 2022 15 -nolook T
using your own data stored as p0410150.22o in the working directory your RINEX o file may also be gzipped. I believe Hatanaka compressed is also allowed.

rinex2snr 940050 2021 31 -archive jp
GSI archive in Japan - password required. Station names are six characters

rinex2snr mchl00aus 2022 15 -orb rapid -archive ga
30 sec RINEX3 data for mchl00aus and Geoscience Australia

rinex2snr mchl00aus 2022 15 -orb rapid -nolook T
works if the RINEX 3 crx.gz or rnx files are in $REFL_CODE/2022/rinex/mchl

rinex2snr mchl00aus 2022 15 -orb rapid -nolook T -strip T
Removes non-SNR data before translating.

rinex2snr mchl00aus 2022 15 -orb rapid -samplerate 30 -nolook T
This should analyze a RINEX 3 file if it exists in your local working directory. it will not search anywhere else for the file. It should be a 30 sec, 1 day file for this example

rinex2snr mchl00aus 2022 15 -orb rapid -samplerate 1 -nolook T -stream S -rate high
This should analyze a RINEX 3 file if it exists in your local working directory. it will not search anywhere else for the file. It should be a 1 sec, 1 day file for this example with S being set for streaming in the filename.

rinex2snr warn00deu 2023 87 -dec 5 -rate high -samplerate 1 -orb rapid -archive bkg-igs -stream S
1 sec data for warn00deu, 1 sec decimated to 5 sec, multi-GNSS, bkg IGS archive, streamed

rinex2snr tgho 2019 1 -doy_end 365 -archive nz
example for multiday SNR file creation

Parameters

- **station** (*str*) – 4 or 9 character ID of the station, respectively for RINEX 2.11 and RINEX 3, preferably lowercase I believe 6 characters are allowed for GSI (Japan), but I have not tested it in a while
- **year** (*int*) – Year
- **doy** (*int*) – Day of year
- **snr** (*int, optional*) – SNR format. This tells the code what elevation angles to save data for. Will be the snr file ending. value options:

  66 (default) : saves all data with elevation angles less than 30 degrees

  99 : saves all data with elevation angles between 5 and 30 degrees

  88 : saves all data

  50 : saves all data with elevation angles less than 10 degrees
- **orb** (*str, optional*) – Which orbit files to download. Value options:

  gps (default < 2021) : will use GPS broadcast orbit

  rapid (default > 2021) : GFZ rapid, multi-GNSS. After 2025/168 the default is gnss.

  gps+glos : will use JAXA orbits which have GPS and Glonass (usually available in 48 hours)

  gnss : use GFZ final orbits, which is multi-GNSS (available in 2-4 days?), but from CDDIS archive

  gnss-gfz : GFZ orbits downloaded from GFZ instead of CDDIS, but do they include beidou?. Same as gnss3?

  nav : GPS broadcast, perfectly adequate for reflectometry. Same as gps.

  igs : IGS precise, GPS only

  igr : IGS rapid, GPS only

  jax : JAXA, GPS + Glonass, within a few days, missing block III GPS satellites

  gbm : GFZ Potsdam, multi-GNSS, not rapid, via CDDIS

  grg : French group, GPS, Galileo and Glonass, not rapid

  esa : ESA, multi-GNSS

gfr : GFZ rapid, GPS, Galileo and Glonass, since May 17 2021

wum : Wuhan ultra-rapid, from CDDIS

wum2 : Wuhan ultra-rapid, from Wuhan FTP

ultra: first tries GFZ ultra-rapid then Wuhan, multi-GNSS

- **rate** (*str, optional*) – The data rate. Rather than numerical value, this tells the code which folder to use value options:

  low (default) : standard rate data. Usually 30 sec, but sometimes 15 sec.

  high : high-rate data

- **dec** (*int, optional*) – Decimation rate. 0 is default which means do nothing.
- **nolook** (*bool, optional*) – tells the code to retrieve RINEX files from your local machine. default is False
- **archive** (*str, optional*) – Select which archive to get the files from. Default is all value options:

  bev : (Austria Federal Office of Metrology and Surveying)

  bfg : (German Agency for water research, only Rinex 3, requires password)

  bkg-igs : IGS data at the BKG (German Agency for Cartography and Geodesy)

  bkg-euref : EUREF data at the BKG (German Agency for Cartography and Geodesy)

  cddis : (NASA's Archive of Space Geodesy Data)

  epn : Belgium

  ga : (Geoscience Australia)

  gnet : Greenland Network, RINEX3 only

  gfz : (GFZ, Germany)

  ignes : IGN in Spain, only RINEX 3

  jp : (GSI, Japan requires password)

  jeff : (My good friend Professor Freymueller!)

  kadaster: (Dutch Geodetic data)

  ngs : (National Geodetic Survey, USA)

  ngs-hourly : (merged hourly files from National Geodetic Survey, USA)

  nrcan : (Natural Resources Canada)

  nz : (GNS, New Zealand)

  sonel : (GLOSS archive for GNSS data)

  sopac : (Scripps Orbit and Permanent Array Center)

  special : (set aside files at UNAVCO for reflectometry users)

  unavco : (University Navstar Consortium, now Earthscope)

  all : (searches sopac and unavco)

- **doy_end** (*int, optional*) – end day of year to be downloaded.
- **year_end** (*int, optional*) – end year.

- **overwrite** (*bool, optional*) – Make a new SNR file even if one already exists (overwrite existing file). Default is False.
- **samplerate** (*int, optional*) – sample rate for RINEX 3 files only. Default is 30.
- **stream** (*str*) – RINEX 3 files only, R (default) or S
- **mk** (*bool, optional*) – Default is False. Use True for uppercase station names and for the non-standard file structure preferred by some users. Look at the function the_makan_option in rinex2snr.py for more information. The general requirement is that your RINEX 2.11 file should be normal RINEX or gzipped normal RINEX. This flag allows access to Hatanaka/compressed files stored locally and in $REFL_CODE/YYYY/rinex/ssss where YYYY is the year and ssss is station name
- **weekly** (*bool, optional, deprecated*) – This originally took 1 out of every 7 days in the doy-doy_end range (one file per week) - used to save cpu time. Default is False.
- **strip** (*bool, optional*) – Reduces observables since the translator does not allow more than 25 in a RINEX 2.11 file. Default is False.
- **screenstats** (*bool, optional*) – if true, prints more information to the screen
- **gzip** (*bool, optional*) – default is true, SNR files are gzipped after creation.
- **monthly** (*bool, optional*) – default is false. snr files created every 30 days instead of every day This does not work anymore
- **par** (*int, optional*) – default is None. parallel processing, valid up to values of 10 for some archives.
- **timeout** (*int, optional*) – This is a non-standard option for timeouts when using high-rate downloads and requests. I added this parameter to let you set the timeout value, but it has not been implemented everywhere. right now just the BKG
- **extension** (*str, optional*) – parameter that tells the code you want to use parameters saved in the gnssir json for that extension parameter. otherwise it uses station.json. It is a convenience for saving things like stream, samplerate, archive, orb, and snr settings that previously had to be input on the command line
- **debug** (*bool, optional*) – run without task queue - important for debugging.
- **quiet** (*bool, optional*) – run gfzrnx for RINEX 3 files but suppress the screen output (default is True)

gnssrefl.rinex2snr_cl.**rinex2snr_day_worker**(*worker_args*)

Worker for parallel rinex2snr. Returns file size in bytes (0 = failure).

gnssrefl.rinex2snr_cl.**snr_file_size**(*station*, *year*, *doy*, *snr_type*)

Return size in bytes of the SNR file for this day, or 0 if not found.

gnssrefl.rinex2snr_cl.**z_process_jobs**(*mjd1*, *mjd2*, *args*)

Sequential processing for rinex2snr.

gnssrefl.rinex3_rinex2 module

Translates rinex3 to rinex2. relies on gfzrnx

gnssrefl.rinex3_rinex2.**main**()

Converts a RINEX 3 file into a RINEX 2.11 file. Uses gfzrnx.

It will delete your RINEX 3 file!

This code has been updated so that your input filename can include a path

Parameters

- **rinex3** (*str*) – filename for RINEX 3 file
- **rinex2** (*str, optional*) – filename for RINEX 2.11 file
- **dec** (*integer, optional*) – decimation value (seconds)
- **overwrite** (*bool, optional*) – whether to overwrite existing RINEX 2 file. Default is False (F)
- **gpsonly** (*bool, optional*) – whether to remove everything except GPS. Default is False (F)
- **quiet** (*bool, optional*) – whether to print gfzrnx output to the screen. Set to F to see it

gnssrefl.rinex3_snr module

gnssrefl.rinex3_snr.**main**()

Creates SNR file from RINEX 3 file that is stored locally Requires the gfzrnx executable to be available. Does not overwrite existing SNR files. It would be nice if someone would add that - plus utilize the libraries for boolean inputs.

Parameters

- **rinex3** (*str*) – name of RINEX3 file
- **orb** (*str*) – optional orbit choice. default is gbm
- **snr** (*int*) – snr file choice. default is 66

gnssrefl.rinex_coords module

gnssrefl.rinex_coords.**main**()

checks the first 100 lines of a rinex file in a quest to determine the approximate Cartesian coordinates, which are also displayed on the screen in LLH. It does not care if it is rinex, hatanaka rinex, or rinex3. But it does require it to not be gzipped or unix compressed. If you would like that option, please submit a PR.

Example

rinex_coords p1013550.22o

gnssrefl.rinpy module

exception gnssrefl.rinpy.**RinexError**

Bases: Exception

gnssrefl.rinpy.**collapse_rinex3_obs**(*obsdata*, *obstypes*)

Collapse RINEX 3 three-char observable names to RINEX 2 two-char band names.

For each constellation and band, merges all matching candidate observables in priority order: highest-priority non-NaN value wins at each (epoch, sat) cell. This mirrors gfzrnx behavior where e.g. S1C and S1X both map to S1, with S1C preferred when available.

Parameters

- **obsdata** (*dict*) – Separated observation data as returned by separateobservables().
- **obstypes** (*dict*) – Observable types per constellation as returned by processrinexfile().

Returns

- **new_obsdata** (*dict*) – Remapped observation data with two-char keys.
- **new_obstypes** (*dict*) – Remapped observable type lists with two-char names.

gnssrefl.rinpy.**get_favourite_obs_dict**()

Parse myfavoriteobs() into a priority dict for collapsing RINEX 3 obs.

Returns dict like {'G': {'S1': ['S1C', 'S1X'], 'S2': ['S2X', 'S2L']}, ...} Bare two-char names (e.g. 'S1' in Galileo) are kept as-is and will match any three-char observable starting with that prefix.

gnssrefl.rinpy.**getrinexversion**(*filename*)

Scan the file for RINEX version number.

Parameters
filename (*str*) – Filename of the rinex file

Returns
version – Version number.

Return type
str

gnssrefl.rinpy.**loadrinexfromnpz**(*npzfile*)

Load data previously stored in npz-format

Parameters
npzfile (*str*) – Path to the stored data.

Returns
observationdata, satlists, prntoidx, obstypes, header, obstimes – Data in the same format as returned by processrinexfile

Return type
dict

gnssrefl.rinpy.**mergerinexfiles**(*filelist*, *savefile=None*)

Process several rinexfiles and merges them into one file.

Can be used to for example merge several rinexfiles from the same day to a single file. All files must be from the same receiver and have the same version. No guarantees are given if files are from different receivers.

!!! Currently only functional for RINEX3. !!!

Parameters

- **filename** (*str*) – Filename of the rinex file
- **savefile** (*str, optional*) – Name of file to save data to. If supplied the data is saved to a compressed npz file.

Returns

- **observationdata** (*dict*) – Dict with a nobs x nsats x nobstypes nd-array for each satellite constellation containing the measurements. The keys of the dict correspond to the systemletter as used in RINEX files (G for GPS, R for GLONASS, etc).

  nobs is the number of observations in the RINEX data, nsats the number of visible satellites for the particular system during the whole measurement period, and nobstypes is the number of different properties recorded.
- **satlists** (*dict*) – Dict containing the full list of visible satellites during the whole measurement period for each satellite constellation.
- **prntoidx** (*dict*) – Dict which for each constellation contains a dict which translates the PRN number into the index of the satellite in the observationdata array.
- **obstypes** (*dict*) – Dict containing the observables recorded for each satellite constellation.
- **header** (*dict*) – Dict containing the header information from the first RINEX file.
- **obstimes** (*list[datetime.datetime]*) – List of time of measurement for each measurement epoch.

gnssrefl.rinpy.**processrinexfile**(*filename*, *savefile=None*)

Process a RINEX file into python format

Parameters

- **filename** (*str*) – Filename of the rinex file
- **savefile** (*str, optional*) – Name of file to save data to. If supplied the data is saved to a compressed npz file.

Returns

- **observationdata** (*dict*) – Dict with a nobs x nsats x nobstypes nd-array for each satellite constellation containing the measurements. The keys of the dict correspond to the systemletter as used in RINEX files (G for GPS, R for GLONASS, etc).

  nobs is the number of observations in the RINEX data, nsats the number of visible satellites for the particular system during the whole measurement period, and nobstypes is the number of different properties recorded.
- **satlists** (*dict*) – Dict containing the full list of visible satellites during the whole measurement period for each satellite constellation.
- **prntoidx** (*dict*) – Dict which for each constellation contains a dict which translates the PRN number into the index of the satellite in the observationdata array.
- **obstypes** (*dict*) – Dict containing the observables recorded for each satellite constellation.

- **header** (*dict*) – Dict containing the header information from the RINEX file.
- **obstimes** (*list[datetime.datetime]*) – List of time of measurement for each measurement epoch.

gnssrefl.rinpy.**readheader**(*lines*, *rinexversion*)

gnssrefl.rinpy.**saverinextonpz**(*savefile*, *observationdata*, *satlists*, *prntoidx*, *obstypes*, *header*, *obstimes*)

Save data to numpy's npz format.

Parameters

- **savefile** (*str*) – Path to where to save the data.
- **observationdata** (*dict*) – Data as returned from processrinexfile
- **satlists** (*dict*) – Data as returned from processrinexfile
- **prntoidx** (*dict*) – Data as returned from processrinexfile
- **obstypes** (*dict*) – Data as returned from processrinexfile
- **header** (*dict*) – Data as returned from processrinexfile
- **obstimes** (*dict*) – Data as returned from processrinexfile

See also:

processrinexfile

gnssrefl.rinpy.**separateobservables**(*observationdata*, *obstypes*)

Parameters

- **observationdata** (*dict*) – Data dict as returned by processrinexfile, or loadrinexfromnpz.
- **obstypes** (*dict*) – Dict with observation types for each system as returned by processrinexfile, or loadrinexfromnpz.

Returns

separatedobservationdata – Dict for each system where the data for each observable is separated into its own dict. I.e. to access the P1 data for GPS from a RINEX2 file it is only necessary to write *separatedobservationdata['G']['C1']*.

Return type

dict

gnssrefl.rt_rinex3_snr module

gnssrefl.rt_rinex3_snr.**main**()

Creates SNR file from RINEX 3 file that is stored locally in makan type folders It may allow crx endings - I am not sure. It only allows GFZ ultra orbit files That could be changed.

This code could be improved - it would be great if the people that are using it would make those changes.

Parameters

- **rinex3** (*str*) – name of RINEX 3 file
- **dec** (*str, optional*) – optional decimation value

gnssrefl.sd_libs module

gnssrefl.sd_libs.**RH_ortho_plot2**(*station*, *H0*, *year*, *txtdir*, *fs*, *time_rh*, *rh*, *gap_min_val*, *th*, *spline*, *delta_out*, *csvfile*, *gap_flag*, *hires_figs*, *knots*)

H0 can only have a single value in this code

Makes a plot of the final spline fit to the data. Output time interval controlled by the user.

It writes out the file with the spline fit. Location is printed to the screen

Parameters

- **station** (*str*) – name of station, 4 ch
- **H0** (*float*) – datum correction (orthometric height) to convert RH to MSL data, in meters (changed to a list for v. 3.18.5)
- **year** (*int*) – year of the time series (ultimately should not be needed)
- **txtdir** (*str*) – location of plot
- **fs** (*int*) – fontsize
- **time_rh** (*numpy of floats*) – time of rh values, in fractional doy I believe
- **rh** (*numpy of floats*) – refl hgt in meters
- **gap_min_val** (*float*) – minimum length gap allowed, in day of year units
- **th** (*numpy floats*) – time values in MJD
- **spline** (*output of interpolate.BSpline*) – used for fitting
- **delta_out** (*int*) – how often spline is printed, in seconds
- **csvfile** (*bool*) – print out csv instead of plain txt
- **gap_flag** (*bool*) – whether to write 999 in file where there are gaps
- **knots** (*int*) – number of knots per day used in final spline

gnssrefl.sd_libs.**find_ortho_height**(*station*, *extension*)

Find orthometric (sea level) height used in final subdaily spline output and plots. This value should be defined for the GPS L1 phase center of the GNSS antenna as this is what is assumed in the subdaily code.

Updated to return Hdate variable

Parameters

- **station** (*str*) – 4 ch station name
- **extension** (*str*) – gnssir analysis, extension mode

Returns

- **Hortho** (*list of floats*) – originally one value, now allowed to be a list. orthometric height from gnssir json analysis file as defined as Hortho, in meters. If your preferred value for Hortho is not present, it is calculated from the ellipsoidal height and EGM96.
- **Hdate** (*list of str*) – date associated with a given Hortho Added to accommodate antenna height changes

gnssrefl.sd_libs.**flipit3**(*tvd*, *col*)

Third version of the flipit code. It takes a time series of RH values, extracts 24 hours of observations from the beginning and end of the series, uses them as fake data to make the spline fit stable. Also fill the temporal gaps with fake data Previous versions assumed you were going to have full days of data at the beginning and end of the series.

This version uses MJD rather than day of year for x-axis

Parameters

- **tvd** (*numpy array of floats*) – output of LSP runs.
- **col** (*integer*) – column number (in normal speak) of the RH results in python-speak, this will have one subtracted from it

Returns

- **tnew** (*numpy array of floats*) – time in MJD
- **ynew** (*numpy array*) – RH in meters

gnssrefl.sd_libs.**fundyplt1**(*station*, *az*, *RH*)

gnssrefl.sd_libs.**mirror_plot**(*tnew*, *ynew*, *spl_x*, *spl_y*, *txtdir*, *station*, *beginT*, *endT*)

Makes a plot of the spline fit to the mirrored RH data Plot is saved to txtdir as station_rhdot1.png

Parameters

- **tnew** (*numpy of floats*) – time in MJD
- **ynew** (*numpy of floats*) – RH in meters
- **spl_x** (*numpy of floats*) – time in days of year
- **spl_y** (*numpy of floats*) – smooth RH, meters
- **txtdir** (*str*) – directory for plot
- **station** (*str*) – name of station for title
- **beginT** (*float*) – first measurement time (MJD) for RH measurement
- **endT** (*float*) – last measurement time (MJD) for RH measurement

gnssrefl.sd_libs.**mjd_to_obstimes**(*mjd*)

takes mjd array and converts to datetime for plotting.

Parameters
mjd (*numpy array of floats*) – mod julian date

Returns
dt

Return type
numpy array of datetime objects

gnssrefl.sd_libs.**numsats_plot**(*station*, *tval*, *nval*, *Gval*, *Rval*, *Eval*, *Cval*, *txtdir*, *fs*, *hires_figs*, *year*, *closefigure*)

makes the plot that summarizes the number of satellites in each constellation per epoch

Parameters

- **station** (*str*) – name of the station
- **tval** (*numpy array*) – datetime objects?

- **nval** (*numpy array*) – number of total satellites at a given epoch
- **Gval** (*numpy array*) – number of galileo satellites at an epoch
- **Rval** (*numpy array*) – number of glonass satellites at an epoch
- **Eval** (*numpy array*) – number of galileo satellites at an epoch
- **Cval** (*numpy array*) – number of beidou satellites at an epoch
- **txtdir** (*str*) – where results are stored
- **fs** (*int*) – fontsize for the plots
- **hires_figs** (*bool*) – try to plot high resolution
- **year** (*int*) – calendar year
- **closefigure** (*bool*) –

gnssrefl.sd_libs.**pickup_subdaily_json_defaults**(*xdir*, *station*, *extension*)

picks up an existing gnssir analysis json. augments with subdaily parameters if needed. Returns the dictionary.

Parameters

- **xdir** (*str*) – REFL_CODE code location
- **station** (*str*) – name of station
- **extension** (*str*) – possible extension location

Returns

station_config – contents of gnssir json

Return type

dictionary

gnssrefl.sd_libs.**print_badpoints**(*t*, *outliersize*, *txtdir*, *real_residuals*)

prints outliers to a file.

Parameters

- **t** (*numpy array*) – lomb scargle result array of "bad points". Format given below
- **outliersize** (*float*) – outlier criterion, in meters
- **txtdir** (*str*) – directory where file is written
- **real_residuals** (*numpy array of floats*) – assume this is RH residuals in meters

Return type

writes to a file called outliers.txt in the Files/station area

gnssrefl.sd_libs.**quickTr**(*year*, *doy*, *frachours*)

takes timing from lomb scargle code (year, doy) and UTC hour (fractional) and returns a date string

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **frachours** (*float*) – real-valued UTC hour

Returns

datestring – date ala YYYY-MM-DD HH:MM:SS

Return type
str

gnssrefl.sd_libs.**rh_plots**(*otimes*, *tv*, *station*, *txtdir*, *year*, *d1*, *d2*, *percent99*)

overview plots for rh_plot

Parameters

- **otimes** (*numpy array of datetime objects*) – observation times
- **tv** (*numpy array*) – gnssrefl results written into this variable using loadtxt
- **station** (*str*) – station name, only used for the title
- **txtdir** (*str*) – directory where the plots will be written to
- **year** (*int*) – what year is being analyzed
- **d1** (*int*) – minimum day of year
- **d2** (*int*) – maximum day of year
- **percent99** (*bool*) – whether you want only the 1-99 percentile plotted

gnssrefl.sd_libs.**rhdot_plots**(*th*, *correction*, *rhdot_at_th*, *tvel*, *yvel*, *fs*, *station*, *txtdir*, *hires_figs*, *year*)

makes the rhdot correction plots

Parameters

- **th** (*numpy array*) – time of obs, MJD
- **correction** (*numpy array*) – rhcorrections in meters
- **rhdot_at_th** (*numpy array of floats*) – spline fit for rhdot in meters
- **tvel** (*numpy array of floats*) – time for surface velocity in MJD
- **yvel** (*numpy array of floats*) – surface velocity in m/hr
- **fs** (*integer*) – fontsize
- **station** (*str*) – station name
- **txtdir** (*str*) – file directory for output
- **hires_figs** (*bool*) – whether you want eps instead of png
- **year** (*int*) – calendar year

gnssrefl.sd_libs.**stack_two_more**(*otimes*, *tv*, *ii*, *jj*, *stats*, *station*, *txtdir*, *sigma*, *kplt*, *hires_figs*, *year*, *close_figures*)

makes a plot of the reflector heights before and after minimal editing

Parameters

- **otimes** (*numpy array of datetime objects*) – observation times ??? or is this a list
- **tv** (*numpy array*) – variable with the gnssrefl LSP results
- **ii** (*numpy array*) – indices of good data
- **jj** (*numpy array*) – indices of bad data
- **station** (*str*) – station name
- **txtdir** (*str*) – directory where plots will be written
- **sigma** (*float*) – what kind of standard deviation is used for outliers (3sigma, 2.5 sigma etc)

- **kplt** (*bool*) – make extra plot for kristine
- **year** (*int*) – calendar year

gnssrefl.sd_libs.**subdaily_resids_last_stage**(*station*, *year*, *th*, *biasCor_rh*, *spline_at_GPS*, *fs*, *strsig*, *hires_figs*, *txtdir*, *ii*, *jj*, *th_even*, *spline_whole_time*)

Makes the final residual plot for subdaily (after RHdot and IF correction made). Returns the bad points ...

Allows either the original or multiyear option..

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – calendar year
- **th** (*numpy array of ??*) – time variable of some kind, fractional day of year ?
- **biasCor_rh** (*numpy array of floats*) – refl hgts that have been corrected for RHdot and IF
- **spline_at_GPS** (*numpy array of floats*) – RH derived From the spline fit and calculated at GPS time tags
- **fs** (*int*) – font size
- **strsig** (*str*) – sigma string to go on the legend
- **hires_figs** (*bool*) – whether to save the plots with better resolution
- **txtdir** (*str*) – directory where the plot will be saved
- **ii** (*numpy array*) – indices of the outliers?
- **jj** (*numpy array*) – indices of the values to keep?
- **th_even** (*numpy array*) – evenly spaced time values, day of year
- **spline_whole_time** (*numpy array of flots*) – splinefit for ???

Returns
badpoints2 – RH residuals

Return type
numpy array of floats

gnssrefl.sd_libs.**testing_nvals**(*Gval*, *Rval*, *Eval*, *Cval*)

writing the number of observations per constellation as a test. not currently used

Parameters Gval: numpy array

GPS RH values

Rval
[numpy array] GLONASS RH values

Eval
[numpy array] Galileo RH values

Cval
[numpy] Beidou RH values

writes to a file - kristine.txt returns nothing

gnssrefl.sd_libs.**the_last_plot**(*tv*, *station*, *plotname*)

simple - reversed - reflector height plot - created after all corrections are made (RHdot and Interfrequency)

Location of the png plot is printed to the screen

Parameters

- **station** (*str*) – station name, four characters
- **tv** (*numpy array*) – output of the subdaily code
- **plotname** (*str*) – where the plot should be stored

gnssrefl.sd_libs.**two_stacked_plots**(*otimes*, *tv*, *station*, *txtdir*, *year*, *d1*, *d2*, *hires_figs*, *close_figures*)

This actually makes three stacked plots - not two, LOL It gives an overview for quality control

Wait wait, you want four plots?

Parameters

- **otimes** (*numpy array of datetime objects*) – observations times
- **tv** (*numpy array*) – gnssrefl results written into this variable using loadtxt
- **station** (*str*) – station name, only used for the title
- **txtdir** (*str*) – where the plots will be written to
- **year** (*int*) – what year is being analyzed
- **d1** (*int*) – minimum day of year
- **d2** (*int*) – maximum day of year
- **hires_figs** (*bool*) – true for eps instead of png
- **close_figures** (*bool*) – whether plot will come to the screen this is turned off specifically when too many plot windows are opened

gnssrefl.sd_libs.**vary_Hortho**(*station*, *H0*, *year*, *txtdir*, *fs*, *time_rh*, *rh*, *gap_min_val*, *th*, *spline*, *delta_out*, *csvfile*, *gap_flag*, *hires_figs*, *knots*)

Makes a plot of the final spline fit to the data. Output time interval controlled by the user.

It writes out the file with the spline fit. Location is printed to the screen

Parameters

- **station** (*str*) – name of station, 4 ch
- **H0** (*float*) – datum correction (orthometric height) to convert RH to MSL data, in meters (changed to a list for v. 3.18.5)
- **year** (*int*) – year of the time series (ultimately should not be needed)
- **txtdir** (*str*) – location of plot
- **fs** (*int*) – fontsize
- **time_rh** (*numpy of floats*) – time of rh values, in fractional doy I believe
- **rh** (*numpy of floats*) – refl hgt in meters
- **gap_min_val** (*float*) – minimum length gap allowed, in day of year units
- **th** (*numpy floats*) – time values in MJD
- **spline** (*output of interpolate.BSpline*) – used for fitting
- **delta_out** (*int*) – how often spline is printed, in seconds

- **csvfile** (*bool*) – print out csv instead of plain txt
- **gap_flag** (*bool*) – whether to write 999 in file where there are gaps
- **knots** (*int*) – number of knots per day used in final spline

gnssrefl.sd_libs.**write_spline_output**(*iyear*, *th*, *spline*, *delta_out*, *station*, *txtdir*, *Hortho*)

Writing the output of the spline fit to the final RH time series. No output other than this text file for year 2023 and station name ssss:

$REFL_CODE/Files/ssss/ssss_2023_spline_out.txt

I do not think this is used anymore. It has been consolidated with the plot code.

Parameters

- **iyear** (*int*) – full year
- **th** (*numpy array*) – time values of some kind … maybe fractional day of years?
- **spline** (*fit, output of interpolate.BSpline*) – needs doc
- **delta_out** (*int*) – how often you want the splinefit water level written, in seconds
- **station** (*str*) – station name
- **txtdir** (*str*) – output directory
- **Hortho** (*float*) – orthometric height used to convert RH to something more sea level like meters

gnssrefl.sd_libs.**writejsonfile**(*ntv*, *station*, *outfile*)

subdaily RH values written out in json format

This does not appear to be used

Parameters

- **ntv** (*numpy of floats*) – LSP results
- **station** (*str*) – 4 ch station name
- **outfile** (*str*) – filename for output

gnssrefl.sd_libs.**writeout_spline_outliers**(*tvd_bad*, *txtdir*, *residual*, *filename*)

Write splinefit outliers to a text file.

Parameters

- **tvd_bad** (*numpy array*) – output of the lomb scargle calculations
- **txtdir** (*str*) – directory for the output, i.e. $REFL_CODE/FiLes/station
- **residual** (*numpy array*) – RH outliers in units of meters (!)
- **filename** (*str*) – name of file being written

gnssrefl.simple_vegetation_correction module

Simple vegetation correction model for VWC estimation. The default and only option until October 2025.

gnssrefl.simple_vegetation_correction.**save_individual_track_data**(*station*, *vxyz*, *extension*, *fr*)

Save individual track files for model 1. Columns 10-17 (model-2-specific) are written as NaN since model 1 corrects aggregate bins, not individual tracks.

Parameters

- **station** (*str*) – Station name
- **vxyz** (*numpy array*) – Track-level observations (N x 18)
- **extension** (*str*) – Extension used in the analysis json
- **fr** (*int*) – Frequency code

gnssrefl.simple_vegetation_correction.**simple_vegetation_filter**(*station*, *vxyz*, *extension=''*, *bin_hours=24*, *bin_offset=0*, *plt2screen=True*, *fr=20*, *minvalperbin=10*, *skip_plots=False*, *save_tracks=False*)

Simple vegetation model (model 1)

This function applies the simple vegetation correction filter to a vxyz array input. This was the default and only available vegetation correction until October 2025.

Parameters

- **station** (*str*) – 4-char GNSS station name
- **vxyz** (*numpy array*) – Track-level phase observations (16 columns)
- **extension** (*str*) – Extension used in the analysis json (default: '')
- **bin_hours** (*int*) – Time bin size for subdaily support (default: 24)
- **bin_offset** (*int*) – Bin timing offset for subdaily support (default: 0)
- **plt2screen** (*bool*) – Whether plots come to the screen (default: True)
- **fr** (*int*) – Frequency code (1=L1, 5=L5, 20=L2C, default: 20)
- **minvalperbin** (*int, optional*) – Minimum observations required per time bin (default: 10)
- **skip_plots** (*bool, optional*) – Skip saving diagnostic plots (default: False). Used by vwc_hourly to avoid generating redundant per-offset plots.
- **save_tracks** (*bool, optional*) – Save individual track data files (default: False). Columns 10-17 are NaN since model 1 does not compute per-track corrections.

Returns

Dictionary containing: - 'mjd': list of Modified Julian Day values - 'vwc': numpy array of VWC values (percentage units, not leveled) - 'datetime': list of datetime objects for plotting - 'bin_starts': numpy array of bin start hours (subdaily) or empty list (daily)

Return type

dict

gnssrefl.smoosh module

gnssrefl.smoosh.**main**()

Decimates and strips out SNR data from a RINEX 2.11 file. Only RINEX, no Hatanaka RINEX or gzipped files allowed. If you would like to add these features, please submit a PR.

Examples

smoosh p0410010.22o 5
decimates a highrate rinex 2.11 file to 5 seconds

smoosh p0410010.22o 5 -snr T
also eliminates all observation types except for SNR

Parameters

- **rinex** (*str*) – rinex 2.11 filename
- **dec** (*int*) – decimation value in seconds
- **snr** (*bool, optional*) – whether you want only SNR observables helpful when you have too many observables in your file.

gnssrefl.smoosh_snr module

gnssrefl.smoosh_snr.**main**()

decimates a SNR file

Examples

smoosh_snr p041 2015 175 5
decimates an existing SNR file (year 2015 and day of year 175) to 5 seconds

Parameters

- **station** (*str*) – four ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **dec** (*int*) – decimation value in seconds
- **snr** (*int, optional*) – snr file name, i.e. 99 or 10, default is 66
- **doy_end** (*int, optional*) – allows you to analyze data from doy to doy_end

gnssrefl.snow_functions module

gnssrefl.snow_functions.**snow_azimuthal**(*station*, *gps*, *year*, *longer*, *doy1*, *doy2*, *bs*, *plt*, *end_dt*, *outputpng*, *outputfile*, *minS*, *maxS*, *barereq_days*, *end_doy*)

azimuthal snow depth algorithm tries to determine the bare soil correction in 20 degree azimuth swaths

Parameters

- **station** (*str*) – 4 ch station name
- **gps** (*numpy array*) – output of daily average RH file (year,doy,RH etc)
- **year** (*int*) – water year
- **longer** (*bool*) – whether you want the plot to include late summer
- **doy1** (*int*) – day of year, beginning of bare soil
- **doy2** (*int*) – day of year, ending of bare soil
- **bs** (*int*) – bare soil year
- **plt** (*bool*) – whether you want the plots displayed to the screen
- **end_dt** (*datetime*) – I think this is the datetime for the optional end of the plot
- **outputpng** (*str*) – name of the output (plot) png file
- **outputfile** (*str*) – name of the output snowdepth txt file
- **minS** (*float*) – minimum snowdepth for plot (m)
- **maxS** (*float*) – maximum snowdepth for plot (m)
- **barereq_days** (*int*) – number of days required to trust a bare soil average
- **end_doy** (*int*) – last day of year you want to compute snow depth for

gnssrefl.snow_functions.**snow_simple**(*station*, *gps*, *year*, *longer*, *doy1*, *doy2*, *bs*, *plt*, *end_dt*, *outputpng*, *outputfile*, *minS*, *maxS*, *barereq_days*, *end_doy*)

simple snow depth algorithm

Parameters

- **station** (*str*) – 4 ch station name
- **gps** (*numpy array*) – output of daily average RH file (year,doy,RH etc)
- **year** (*int*) – water year
- **longer** (*bool*) – whether you want the plot to include late summer
- **doy1** (*int*) – day of year, beginning of bare soil
- **doy2** (*int*) – day of year, ending of bare soil
- **bs** (*int*) – bare soil year
- **plt** (*bool*) – whether you want the plots displayed to the screen
- **end_dt** (*datetime*) – I think this is the datetime for the optional end of the plot
- **outputpng** (*str*) – name of the output (plot) png file
- **outputfile** (*str*) – name of the output snowdepth txt file
- **minS** (*float*) – minimum snowdepth for plot (m)
- **maxS** (*float*) – maximum snowdepth for plot (m)

- **barereq_days** (*int*) – min number of days to believe a bare soil average
- **end_doy** (*int*) – last day of year you want a snow depth value for

gnssrefl.snow_functions.**snowplot**(*station*, *gobst*, *snowAccum*, *yerr*, *left*, *right*, *minS*, *maxS*, *outputpng*, *pltit*, *end_dt*)

creates and displays snow depth plot. Saves to outputpng

Parameters

- **station** (*str*) – name of GNSS station
- **gobts** (*datetime object*) – time of measurements in datetime format
- **snowAccum** (*numpy array*) – snow depth (meters)
- **yerr** (*numpy array*) – snow depth error (meters)
- **left** (*datetime obj*) – min x-axis limit
- **right** (*datetime obj*) – max x-axis limit
- **minS** (*float*) – minimum snow depth (m)
- **maxS** (*float*) – maximum snow depth (m)
- **outputpng** (*str*) – name of output png file
- **pltit** (*bool*) – whether plot should be displayed to the screen
- **end_dt** (*datetime*) – user provided override date for the end of the plot if None, then ignore

gnssrefl.snow_functions.**time_limits**(*wateryear*, *longer*, *end_doy*)

pick up some time values for windowing RH/snow depth data

Parameters

- **wateryear** (*int*) – water year
- **longer** (*bool*) – whether you want a longer plot
- **end_doy** (*int*) – last day of the year in the water year you want snow depth calculated for

Returns

- **starting** (*float*) – start time, fractional (year + doy/365.25)
- **ending** (*float*) – end time, fractional (year + doy/365.25)
- **left** (*datetime*) – beginning for the plot in datetime
- **right** (*datetime*) – end for the plot in datetime

gnssrefl.snow_functions.**unused**(*plot_begindate*, *plot_enddate*)

gnssrefl.snow_functions.**writeout_azim**(*station*, *outputfile*, *usegps*, *snowAccum*)

writes plain txt file with snow depth results this is for the azimuth leveling version

February 6 2024 no longer setting snowdepth to zero when it is below 5 cm So there will be values above and below zero when there is no snow on the ground but THIS SHOULD NOT be interpreted as negative snow!

Parameters

- **station** (*str*) – 4 char station name
- **outputfile** (*str*) – location of output file (plain txt)
- **usegps** (*numpy array*) – LSP results (year, doy, RH etc

- **snowAccum** (*numpy array*) – snow accumulation results in meters

Returns

- **gobst** (*numpy array*) – datetime useful for plotting
- **snowAccum** (*numpy array*) – snow depth that has passed QC (meters)
- **snowAccumError** (*numpy array*) – standard deviation of daily snow depth retrievals (meters)

gnssrefl.snow_functions.**writeout_snowdepth_v0**(*station*, *outputfile*, *usegps*, *snowAccum*, *yerr*)

writes plain txt file with snow depth results

Parameters

- **station** (*str*) – 4 char station name
- **outputfile** (*str*) – location of output file (plain txt)
- **usegps** (*numpy array*) – LSP results (year, doy, RH etc
- **snowAccum** (*numpy array*) – snow accumulation results in meters
- **yerr** (*numpy array*) – standard deviation of snow depth in meters

Returns
gobst – datetime useful for plotting

Return type
numpy array

gnssrefl.snowdepth_cl module

gnssrefl.snowdepth_cl.**main**()

gnssrefl.snowdepth_cl.**parse_arguments**()

gnssrefl.snowdepth_cl.**snowdepth**(*station: str*, *year: int*, *minS: float | None = None*, *maxS: float | None = None*, *longer: bool = False*, *plt: bool = True*, *bare_date1: str | None = None*, *bare_date2: str | None = None*, *plt_enddate: str | None = None*, *simple: bool = False*, *medfilter: float | None = None*, *ReqTracks: int | None = None*, *barereq_days: int = 15*, *fr: int | None = None*, *hires_figs: bool = False*)

Calculates snow depth for a given station and water year. Before you run this code you must have run gnssir for each day of interest.

You can then run daily_avg to concatenate the results or you can input appropriate values to optional inputs medfilter and ReqTracks.

Currently set for northern hemisphere constraints. This could easily be fixed for the southern hemisphere by reading the json input file. Default values use the median of September results to set "bare soil value" These can be overriden with bare_date1 and bare_date2 (as one would do in Alaska)

Output is currently written to a plain text file and a plot is written to a png file. Both are located in the $REFL_CODE/Files/station directory

If simple is set to true, the algorithms computes bare soil (and thus snow depth), using all values together. The default defines bare soil values every 10 degrees in azimuth.

2024 Feb 6 : stopped the code from setting snowdepth values < 5 cm to zero. This means that "negative" snowdepth will be in the files, but it should not be interpreted to be a new form of snow.

Examples

snowdepth p101 2022
would use results from a previous run of daily_avg

snowdepth p101 2022 -medfilter 0.25 -ReqTracks 50
would run daily_avg for you using 50 tracks/0.25 meter median filter

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – water year (i.e. jan-june of that year and oct-dec of the previous year)
- **minS** (*float, optional*) – minimum snow depth for y-axis limit (m), optional
- **maxS** (*float, optional*) – maximum snow depth for y-axis limit (m), optional
- **longer** (*bool, optional*) – whether you want to plot longer time series (useful for Alaskan sites)
- **plt** (*bool, optional*) – whether you want the plot to come to the screen
- **bare_date1** (*str, optional*) – an override for start bare soil definition
- **bare_date2** (*str, optional*) – an override for end bare soil definition
- **plt_enddate** (*str, optional*) – an override for where you want the plot to end earlier than default
- **simple** (*bool, optional*) – whether you want to use simple algoirthm. Default is False which means you use azimuth corrected bare soil values
- **medfilter** (*float, optional*) – to avoid running daily_avg, you can set median filter in meters; this is used to remove large outliers
- **ReqTracks** (*int, optional*) – to avoid running daily_avg, you can set required number of tracks to create a daily average RH
- **barereq_days** (*int, optional*) – how many bare soil days are required to trust the result, default is 15
- **fr** (*int, optional*) – if you want to restrict to a single frequency at the daily-avg stage (1, 20, etc)
- **hires_figs** (*bool, optional*) – whether you want eps instead of png plots

gnssrefl.snrfile_functions module

class gnssrefl.snrfile_functions.**constants**
Bases: object

c = 299792458

mu = 3986005000000000.0

omegaEarth = 7.2921151467e-05

gnssrefl.snrfile_functions.**elev_limits**(*snroption*)
For given SNR option, returns elevation angle limits

Parameters
snroption (*integer*) – snr file delimeter

Returns

- **emin** (*float*) – minimum elevation angle (degrees)
- **emax** (*float*) – maximum elevation angle (degrees)

gnssrefl.snrfile_functions.**propagate_and_azel_sp3**(*iX*, *iY*, *iZ*, *t*, *recv*, *up*, *East*, *North*, *oE*, *clight*)

Vectorized SP3 orbit propagation with light-time iteration, then azimuth/elevation.

Parameters

- **iX** (*CubicSpline*) – interpolators for satellite ECEF coordinates
- **iY** (*CubicSpline*) – interpolators for satellite ECEF coordinates
- **iZ** (*CubicSpline*) – interpolators for satellite ECEF coordinates
- **t** (*ndarray*) – observation times (GPS seconds of week), shape (N,)
- **recv** (*ndarray*) – receiver ECEF position, shape (3,)
- **up** (*ndarray*) – unit vectors at receiver, shape (3,)
- **East** (*ndarray*) – unit vectors at receiver, shape (3,)
- **North** (*ndarray*) – unit vectors at receiver, shape (3,)
- **oE** (*float*) – Earth rotation rate (rad/s)
- **clight** (*float*) – speed of light (m/s)

Returns

- **elv** (*ndarray, shape (N,)*) – elevation angles in degrees
- **azm** (*ndarray, shape (N,)*) – azimuth angles in degrees [0, 360)

gnssrefl.spline_functions module

gnssrefl.spline_functions.**arc_plots**(*lspfigs*, *snrfigs*, *reflh*, *pgram*, *sat*, *datet*, *elvlims*, *elvt*, *snrdt*, *azdesc*, *xsignal*)

moved these individual plots out of the way

lspfigs : bool

snrfigs : bool

reflh
[numpy array] reflector heights (m)

pgram
[numpy array] periodogram ?

sat : numpy array

datet : datetime

elvlims
[list of floats] min and max elev angle (deg)

xsignal
[str] i think this is L1, L2 etc

gnssrefl.spline_functions.**datetime2gps**(*dt*)

Parameters
dt (*datetime*) –

Returns
gpstime

Return type
float

gnssrefl.spline_functions.**define_inputfile**(*station*, *year*, *doy*, *snr_ending*)

Parameters

- **station** (*str*) – 4 ch name of station
- **year** (*integer*) –
- **doy** (*int*) – day of year
- **snr_ending** (*int*) – file ending, e.g. 66, 99

Returns

- **snrfile** (*str*) – name of snrfile
- **snrdir** (*str*) – name of output directory
- **cyyyy** (*str*) – four character year
- **cdoy** (*str*) – three character day of year

gnssrefl.spline_functions.**freq_out**(*x*, *ofac*, *hifac*)

inputs: x ofac: oversamping factor hifac outputs: two sets of frequencies arrays

gnssrefl.spline_functions.**get_ofac_hifac**(*elevAngles*, *cf*, *maxH*, *desiredPrec*)

computes two factors - ofac and hifac - that are inputs to the Lomb-Scargle Periodogram code. We follow the terminology and discussion from Press et al. (1992) in their LSP algorithm description.

Parameters

- **elevAngles** (*numpy array*) – satellite elevation angles in degrees
- **cf** (*float*) – L-band wavelength/2 in meters
- **maxH** (*float*) – maximum LSP grid frequency in meters
- **desiredPrec** (*float*) – the LSP frequency grid spacing in meters

Returns

- **ofac** (*float*) – oversampling factor
- **hifac** (*float*) – high-frequency factor

gnssrefl.spline_functions.**gps2datenum**(*gt*)

needs documentation

Parameters
gt (*float*) – gps time

Returns
dn

Return type
datetime?

gnssrefl.spline_functions.**gps2datetime**(*gt*)

needs documentation

gnssrefl.spline_functions.**invsnr_header**(*xdir*, *outfile_type*, *station*, *outfile_name*)

Makes header for output of invsnr analysis

Parameters

- **xdir** (*str*) – directory for the output file
- **outfile_type** (*str*) – csv or txt
- **station** (*str*) – 4 character name
- **outfile_name** (*str*) – name of output - if empty string, it uses default

Returns

- **fileID** (*file*) – used for writing to file
- **usetxt** (*bool*) – boolean for the code calling this function to use if you write out special files, they go in the working directory

gnssrefl.spline_functions.**kristine_dictionary**(*alld*, *sat*, *xsignal*)

22feb09 added beidou

gnssrefl.spline_functions.**loadsnrfile**(*snrfile*, *thedir*)

loads the snr file , but does not pick out the signal. using two functions will make it easier to use more than one frequency

do time modification here now. column 4 is time since GPS began, in seconds

Parameters

- **snrfile** (*str*) – name of the SNR file
- **thedir** (*str*) – location of the SNR file

Returns
snrdata – floats. Time (python col 3) is converted to fake gps time

Return type
numpy array

gnssrefl.spline_functions.**make_wavelength_column**(*nr*, *snrdata*, *signal*)

NEEDS DOCUMENTATION

Parameters

- **nr** (*integer*) – number of rows in snrdata
- **snrdata** (*numpy array*) – snrfile array
- **signal** (*string*) – frequency 'L1','L2, etc

Returns
onecolumn – snr data for the requested signal

Return type
one-d numpy array

gnssrefl.spline_functions.**plot_tracks**(*rh_arr*, *rh_dn*)

send the array of LSP results (rh_arr) with time variable for plotting (rh_dn) kl feb09 adding beidou

Parameters

- **rh_arr** (*numpy array*) – data used by inverse code. Need to add desc
- **rh_dn** (*numpy array*) – data used by inverse code. Need to add desc

gnssrefl.spline_functions.**readklsnrtxt**(*snrfile*, *thedir*, *signal*)

parses the contents of a snrfile. The file itself is read in a separate function now; if SNR data are zero for a given signal, the row is eliminated

As of Oct 28, 2023, gzip after reading SNR file

Parameters

- **snrfile** (*str*) – variable with the file contents
- **thedir** (*str*) – directory where it is located
- **signal** (*str*) – 'L1', 'L2' etc.

Returns

snrdata – 0 : satellite, usual (100 added for glonass, 200 added for galileo) 1 : elev angle, deg 2 : azimuth angle, deg 3 : time in seconds since GPS began 4 : SNR data in db-Hz 5 : new column with wavelength in it, in meters.

Return type

numpy array of floats . Columns defined as:

gnssrefl.spline_functions.**residuals_cubspl_js**(*inparam*, *knots*, *satconsts*, *signal*, *snrdt_arr*, *final_list*, *Nfreq*)

function needed for snr-fitting inverse analysis js must stand for joakim strandberg ???

this has to be modified for multi-frequency fspecdict and Nfreq 22feb09 added beidou

Parameters

- **inparam** –
- **knots** –
- **satconsts** –
- **signal** –
- **snrdt_arr** –
- **final_list** –
- **Nfreq** –

gnssrefl.spline_functions.**residuals_cubspl_spectral**(*kval*, *knots*, *rh_arr*)

function needed for inverse analysis

Parameters

- **kval** –
- **knots** (*numpy array*) –
- **rh_arr** (*numpy array*) – reflector heights in meters

gnssrefl.spline_functions.**satfreq2waveL**(*satc*, *xsignal*, *fsatnos*)

given satellite constellation ('G', 'E' ...) xsignal ('L1','L2' ...) satnos (satellite numbers) 2022feb09 added Beidou.

gnssrefl.spline_functions.**save_lsp_results**(*datet*, *maxind*, *reflh_sub*, *sat*, *elvt*, *azit*, *pgram_sub*, *snrdt*, *pktn*, *isignal*)

just cleaning up - move the temp_arr definition to a function each column is defined below.

Parameters

- **datet** (*float*) – seconds in GPSish time
- **reflh_sub** (*numpy of floats?*) – windowed rh estimates
- **sat** (*int*) – satellite number
- **elvt** (*numpy array of floats*) – elevation angles(deg)
- **azit** (*numpy array of floats*) – azimuth angles (deg)
- **snrdt** (*numpy array of floats*) – detrended SNR data (DC component removed)
- **pktn** (*float*) – peak 2 noise via Dave Purnell's definition
- **isignal** (*int*) – frequency, 1,2, or 5

Returns
tmp_arr

Return type
numpy array (12 columns)

gnssrefl.spline_functions.**set_refraction_model**(*station*, *dmjd*, *station_config*, *imodel*)

imodel is 1 for simple refraction model eventually will add other refraction models

Looks like this was copied from other code and should be consolidated …

Parameters

- **station** (*str*) – 4 ch station name
- **dmjd** (*float*) – modified julian date
- **station_config** (*dictionary*) – station information including latitude and longitude
- **imodel** (*integer*) – set to 1 (time varying off) or 0 (time varying on)

Returns

- **p** (*float*) – pressure (units?)
- **T** (*float*) – temperature in deg C
- **irefr** (*int*) – number value written to output files to keep track of refraction model
- **e** (*float*) – water vapor pressure, hPa
- **Tm** (*float*) – temperature in kelvin
- **lapse_rate** (*float*) – see source code for details

gnssrefl.spline_functions.**signal2list**(*signal*)

turns signal input (e.g. L1+L2) to a list 22feb09 tried to add more frequencies …

Returns
signal_list

Return type
str

gnssrefl.spline_functions.**simpleLSP**(*rhlims*, *lcar*, *precision*, *elvt*, *sinelvt*, *snrdt*, *sat*, *xsignal*, *screenstats*, *fout*, *pktnlim*)

Parameters

- **input** –
- **rhmax)** (*rhlims from dave's code (rhmin and*) –
- **lcar** (*is gnss wavelength in m*) –
- **periodogram** (*precision of the*) –
- **meters** (*in*) –
- **degrees** (*elvt - elevation angles in*) –
- **sinelvt** –
- **angle** (*sine elevation*) –
- **data** (*snrdt - detrended snr*) –

gnssrefl.spline_functions.**smarterWay**(*a*)

just want to know how many true values there are in the a dictionary and then write them to a list, as in ['G1','G2'] sure to be a better way - but this works for now

gnssrefl.spline_functions.**snr2arcs**(*station*, *snrdata*, *azilims*, *elvlims*, *rhlims*, *precision*, *year*, *doy*, *signal='L1'*, *normalize=False*, *snrfigs=False*, *lspfigs=False*, *polydeg=2*, *gaptlim=300*, *pktnlim=4*, *savefile=False*, *screenstats=False*, *l2c_only=False*, *satconsts=['G', 'R', 'E']*, ***kwargs*)

reads an array of snr data (output from readklsnrtxt) and organises into: reflector height estimates, stats and detrended snr data for inverse analysis

Parameters

- **station** (*str*) – 4 ch station name
- **snrdata** (*numpy array*) – contents of SNR datafile
- **azilims** (*list of floats*) – azimuth angle limits (e.g., [90, 270])
- **elvlims** (*list of floats*) – elevation angle limits (e.g., [5, 30])
- **rhlims** (*list of floats*) – upper and lower reflector height limits (in metres) for quality control
- **signal** (*str*) – default 'L1' (C/A), can also use L2... if want to use L5 or whatever else you need to make some edits
- **normalize** (*bool*) – if you want to normalize the arcs so that they have the same amplitude
- **snrfigs** (*bool*) – if you want to produce some figures of SNR arcs
- **lspfigs** (*bool*) – if you want to produce some figures of Lomb-Scargle Periodograms
- **polydeg** (*float*) – degree of polynomial for DC
- **gaptlim** (*float*) – if there is a gap in time bigger than [gaptlim] seconds in a particular arc then it will be ignored
- **pktnlim** (*float*) – peak to noise ratio qc condition = the peak of the LSP / mean of LSP within the range [rhlims]
- **savefile** (*bool*) – if you want to save the output to a pickle file then use this parameter as the name (string)

- **kwargs** (*see below*) –
- **tempres** (*int*) – if want to use different temporal resolution to input data (in seconds)
- **satconsts** (*default use all given, otherwise specify from ['G', 'R', 'E'] (gps / glonass / galileo)*) –

Returns

- **rh_arr** (*numpy array*) – reflector height estimates and stats
- **snrdt_arr** (*numpy array*) – detrended SNR data for inverse analysis

gnssrefl.spline_functions.**snr2spline**(*station*, *year*, *doy*, *azilims*, *elvlims*, *rhlims*, *precision*, *kdt*, *snrfit=True*, *signal='L1'*, *savefile=False*, *doplot=True*, *rough_in=0.1*, ***kwargs*)

function analyzes a SNR file and outputs a fitted spline

note that the file must be 24 hours long or it will not work

Parameters

- **station** (*str*) – 4 ch station name
- **year** (*int*) – full year
- **doy** (*int*) – day of year
- **azilims** (*list of floats*) – azimuth angle limits (e.g., [90, 270])
- **elvlims** (*list of floats*) – elevation angle limits (e.g., [5, 30])
- **rhlims** (*list of floats*) – upper and lower reflector height limits (in metres) for quality control e.g., [5, 10] is 5 and 10 m
- **precision** (*float*) – precision of the periodogram (m)
- **kdt** (*float*) – spline knot spacing in seconds
- **day** (*knots are spaced evenly except for at the start and end of the*) –
- **spline** (*The idea is that you would ignore the first and last knots and then you could have a continuous*) –
- **hours)** (*if kdt = 2 * 60 * 60 (2)* –
- **0h** (*then knots at*) –
- **1h** –
- **3h** –
- **21h** (*...*) –
- **23h** –
- **24h** –
- **spline** –
- **days** (*with knots every 2 hours over multiple*) –
- **snrfit** (*True or False if you want to do inverse modelling of the SNR data*) –
- **signal** (*'L1', 'L2', currently under development*) –
- **savefile** (*set True if you want to save the output to a file*) –

- **doplot** (*set True if you want to produce a plot with the output from the analysis*) –
- **rough_in** (*'roughness' parameter in the inverse modelling of SNR data (see Strandberg et al., 2016)*) –
- **kwargs** (*see below*) –
- **tempres** (*if want to use different temporal resolution to input data (in seconds)*) –
- **satconsts** (*default use all given, otherwise specify from ['G', 'R', 'E'] (gps / glonass / galileo)*) –

Returns

- **invout** (*dictionary*) – outputs from inverse analysis
- *This documentation was provided by the original author, David Purnell*

gnssrefl.subdaily module

gnssrefl.subdaily.**apply_new_constraints**(*tv*, *azim1*, *azim2*, *ampl*, *peak2noise*, *year*, *d1*, *d2*, *h1*, *h2*, *freqs*)

cleaning up the main code. this sorts data and applies various "commandline" constraints tv is the full set of results from gnssrefl

Parameters

- **tv** (*numpy array*) – lsp results
- **azim1** (*float*) – min azimuth (deg)
- **azim2** (*float*) – max azimuth (deg)
- **ampl** (*list of float*) – required amplitude for periodogram
- **peak2noise** (*float*) – require peak2noise criterion
- **year** (*int*) – full year
- **d1** (*int*) – min day of year
- **d2** (*int*) – max day of year
- **h1** (*float*) – min reflector height (m)
- **h2** (*float*) – max reflector height (m)
- **freqs** (*list of int*) – list of frequencies that correspond to minimum amplitudes if empty list, then just use the one amplitude for all

Returns

- **tv** (*numpy array*) – LSP, edited from input
- **t** (*numpy of floats*) – crude time for obs, in fractional days
- **rh** (*numpy of floats*) – reflector heights (m)
- **firstmjd** (*int*) – first MJD in the file (requested)
- **lastmjd** (*int*) – last MJD in the file (requested)

gnssrefl.subdaily.**flipit**(*tvd*, *col*)

take RH values from the first and last day and attaches them as fake data to make the spline fit stable. Also fill the temporal gaps with fake data

Parameters

- **tvd** (*numpy array of floats*) – output of LSP runs.
- **col** (*integer*) – column number (in normal speak) of the RH results in python-speak, this has one subtracted

Returns

- **tnew** (*numpy array of floats*) – time in days of year
- **ynew** (*numpy array*) – RH in meters

gnssrefl.subdaily.**flipit2**(*tvd*, *col*)

take RH values from the first and last day and attaches them as fake data to make the spline fit stable. Also fill the temporal gaps with fake data

This version uses MJD rather than day of year for x-axis

Parameters

- **tvd** (*numpy array of floats*) – output of LSP runs.
- **col** (*integer*) – column number (in normal speak) of the RH results in python-speak, this has one subtracted

Returns

- **tnew** (*numpy array of floats*) – time in days of year
- **ynew** (*numpy array*) – RH in meters

gnssrefl.subdaily.**fract_to_obstimes**(*spl_x*)

this does not seem to be used

Parameters

- **spl_x** (*numpy array*) – fractional time
- **obstimes** (*numpy array*) – datetime format

gnssrefl.subdaily.**my_percentile**(*rh*, *p1*, *p2*)

numpy percentile was crashing docker build this is a quick work around

Parameters

- **rh** (*numpy array*) – reflector heights, but could be anything really
- **p1** (*float*) – low percentage (from 0-1)
- **p2** (*float*) – high percentage (from 0-1)

Returns

- **low** (*float*) – low value (using input percentile)
- **highv** (*float*) – high value (using input percentile)

gnssrefl.subdaily.**output_names**(*txtdir*, *txtfile*, *csvfile*, *jsonfile*)

figures out what the names of the outputs are going to be

I have modified this so it always returns plain txt. csv will simply be written out in addition.

this function no longer has much point.

Parameters

- **txtdir** (*str*) – the directory where the results should be written out
- **txtfile** (*str*) – name of the output file
- **csvfile** (*bool*) – cl input whether the output file should be csv format
- **jsonfile** (*bool*) – cl input for whether the output file should be in the json format

Returns

- **writetxt** (*bool*) – whether output should be plain txt
- **writecsv** (*bool*) – whether output should be csv format
- **writejson** (*bool*) – whether output should be json format
- **outfile** (*str*) – output filename

gnssrefl.subdaily.**readin_and_plot**(*station*, *year*, *d1*, *d2*, *plt2screen*, *extension*, *sigma*, *writecsv*, *azim1*, *azim2*, *ampl*, *peak2noise*, *txtfile*, *h1*, *h2*, *kplt*, *txtdir*, *default_usage*, *hires_figs*, *fs*, ***kwargs*)

Reads in and concatenates RH results from previous runs of gnssir and makes various plots to help users assess the quality of the solution

This is basically "section 1" of the code

Exits if no data found

Parameters

- **station** (*str*) – 4 character station name
- **year** (*int*) – full year
- **d1** (*int*) – first day of year evaluated
- **d2** (*int*) – last day of year evaluated
- **plt2screen** (*bool*) – if True plots are displayed to the screen
- **extension** (*str*) – allow user to specify an extension for results (i.e. gnssir was run using extension string)
- **sigma** (*float*) – how many standard deviations away from mean you allow for the crude outlier detector.
- **writecsv** (*bool*) – whether output is written in csv format.
- **azim1** (*float*) – minimum azimuth value (degrees)
- **azim2** (*float*) – maximum azimuth value (degrees)
- **ampl** (*float*) – minimum LSP amplitude allowed this has been changed to a list as of v 3.6.6 it corresponds to the frequency list
- **peak2noise** (*float*) – minim peak2noise value to set solution good
- **txtfile** (*str*) – name of plain text output file

- **h1** (*float*) – minimum reflector height (m)
- **h2** (*float*) – maximum reflector height (m)
- **kplt** (*bool*) – special plot made
- **txtdir** (*str*) – directory where the results will be written
- **default_usage** (*bool*) – flag as to whether you are using this code for subdaily or for rh_plot. this changes the plots a bit.
- **hires_figs** (*bool*) – whether to switch from png to eps
- **fs** (*int*) – fontsize for figure axes

Returns

- **tv** (*numpy array*) – LSP results (augmented)
- **otimes** (*datetime object*) – times of observations
- **fname** (*str*) – initial result file - colated
- **fname_new** (*str*) – result file with outliers removed

gnssrefl.subdaily.**rhdot_correction2**(*station*, *fname*, *fname_new*, *pltit*, *outlierV*, *outlierV2*, ***kwargs*)

Part two of subdaily. It computes rhdot correction and interfrequency bias correction for RH time series. This code assumes you have at least removed crude outliers in the previous section of the subdaily code.

Parameters

- **station** (*str*) – 4 char station name
- **fname** (*list of str*) – input filename(s)
- **fname_new** (*str*) – output filename for results
- **pltit** (*bool*) – whether you want plots to the screen
- **outlierV** (*float*) – outlier criterion, in meters used in first go thru if None, then use 3 sigma (which is the default)
- **outlierV2** (*float*) – outlier criterion, in meters used in second go thru if None, then use 3 sigma (which is the default)
- **delta_out** (*float, optional*) – seconds for smooth output
- **txtdir** (*str*) – if wanting to set your own output directory
- **apply_if_corr** (*bool, optional*) – whether you want to apply the IF correction default is true
- **apply_rhdot** (*bool, optional*) – whether you want to apply the rhdot correction default is true
- **gap_min_val** (*float, optional*) – gap allowed in last spline, in hours
- **knots2** (*int, optional*) – a secondary knot value if you want the final output to use a different one than the one used for outliers and RH dot
- **gap_flag** (*bool, option*) – whether gaps are written as 999 in final output
- **lastpng** (*str*) – name of the last file created.

gnssrefl.subdaily.**spline_in_out**(*x*, *y*, *knots_per_day*)

Given a time series (x in days) and the number of knots you want per day, computes a spline fit for you. It returns both the fitted values and the evenly space x values.

Parameters

- **x** (*numpy of floats*) – time of observations in fractional days
- **y** (*numpy of floats*) – reflector heights in meters
- **knots_per_day** (*int*) – number of knots per day

Returns

- **xx** (*numpy of floats*) – time of the regularly spaced observations
- **spline(xx)** (*numpy of floats*) – spline value at those times

gnssrefl.subdaily.**write_out_header**(*fout*, *station*, *extraline*, ***kwargs*)

writes out header for results file …

Parameters

- **fout** (*fileID*) –
- **station** (*str*) – 4 character station name
- **extraline** (*bool*) – not sure why this is here

gnssrefl.subdaily.**write_subdaily**(*outfile*, *station*, *ntv*, *csv*, *extraline*, ***kwargs*)

writes out the subdaily results. currently only works for plain txt

>> this code should be moved to the library

Parameters

- **input** (*str*) – output filename
- **station** (*str*) – 4 character station name, lowercase
- **nvt** (*numpy multi-dimensional*) – the variable with the LSP results read via np.loadtxt
- **csv** (*bool*) – whether both csv and txt file should be written
- **extraline** (*bool*) – whether the header has an extra line

gnssrefl.subdaily_cl module

gnssrefl.subdaily_cl.**main**()

gnssrefl.subdaily_cl.**parse_arguments**()

gnssrefl.subdaily_cl.**subdaily**(*station: str*, *year: int*, *txtfile_part1: str = ''*, *txtfile_part2: str | None = None*, *csv: bool = False*, *plt: bool = True*, *spline_outlier1: float | None = None*, *spline_outlier2: float | None = None*, *knots: int | None = None*, *sigma: float | None = None*, *extension: str | None = None*, *rhdot: bool = True*, *doy1: int = 1*, *doy2: int = 366*, *testing: bool = True*, *ampl: float = []*, *h1: float = 0.4*, *h2: float = 300.0*, *azim1: int = 0*, *azim2: int = 360*, *peak2noise: float = 0*, *kplt: bool = False*, *subdir: str | None = None*, *delta_out: int | None = None*, *if_corr: bool = True*, *knots_test: int = 0*, *hires_figs: bool = False*, *apply_rhdot: bool = True*, *fs: int = 10*, *alt_sigma: bool = False*, *gap_min_val: float = 6.0*, *year_end: int | None = None*, *knots2: int | None = None*, *gap_flag: bool = False*, *date1: str | None = None*, *date2: str | None = None*, *fundy: str | None = None*, *fundy_trigger: float = 2.25*)

Subdaily combines gnssir solutions and applies relevant corrections needed to measure water levels (tides). As of January 2024, it will allow multiple years. You can also specify which day of year to start with, i.e. -doy1 300

and -doy2 330 will do that range in a single year, or you could specific doy1 and doy2 as linked to the start and stop year (year and year_end)

As of verison 3.10.6 I am starting to move to full ymd start and stop times, i.e. date1=20240101 and date2=20240131 will start on January 1 and end on January 31. Eventually doy1 and doy2 will be ignored.

In general this code is meant to be used at sites with tidal signals. If you have a site without tidal signals, you should consider using daily_avg instead. If you would still like to use this code for rivers and lakes, you should change the defaults for the spline fits. For tidal sites, 8 knots per day is the default. For nearly stationary surfaces, as you would expect for a lake or river, you should use many fewer knots per day.

This code calculates and applies various corrections. New Reflector Height values are added to the output files as new columns. If you run the code but continue to assume the "good answers" are in still in column 3, you are essentially not using the code at all.

As of version 3.1.4 you can use some of the subdaily optional inputs from the gnssir_input created json. See gnssir_input for details.

As of version 3.18.4 you can use multiple Hortho corrections (use Hdate definition in your json). This would accommodate antenna changes.

As of version 2.0.0:

The final output of subdaily is a smooth spline fit to reflector heights (RH) which has been adjusted to mean sea level (meters). For this to be accurate, the user is asked to provide the orthometric height of the L1 GPS antenna phase center. This value should be stored as Hortho in the gnssir analysis strategy file (ssss.json where ssss is the 4 character station name). The output water levels are then defined as Hortho minus RH. If the user does not provide Hortho, one is computed from the station ellipsoidal height stored in the gnssir analysis strategy file and EGM96.

The subdaily code has two main sections.

I. Summarize the retrievals (how many retrievals per constellation), identify and remove gross outliers, provide plots to allow a user to evaluate Quality Control parameters. The solutions can further be edited from the command line (i.e. restrict the RH using -h1 and -h2, in meters, or azimuths using -azim1 and -azim2)

II. This section has the following goals:

- removes more outliers based on a spline fit to the RH retrievals
- calculates and applies RHdot correction
- removes an interfrequency (IF) bias. All solutions are then relative to GPS L1.

txtfile_part1 is optional input if you want to skip concatenating daily gnssir output files and use your own file. Make sure results are in the same format.

txtfile_part2 is optional input that skips part 1 and uses this file as input to the second part of the code.

Examples

subdaily at01 2023 -plt F
for station at01, all solutions in 2023 but no plots to the screen

subdaily at01 2023 -doy1 15 -doy2 45
for all solutions in 2023 between days of year 15 through 45

subdaily at01 2023 -date1 20230115 -date2 20230615
for all solutions in 2023 between January 15 and June 15

subdaily at01 2023 -h2 14 -if_corr F
for all solutions in 2023 but with max RH set to 14 meters and interfrequency correction not applied

subdaily at01 2022 -year_end 2023
analyze all data for years 2022 and 2023

subdaily at03 2022 -azim1 180 -azim2 270
restrict solutions to azimuths between 180 and 270

Parameters

- **station** (*str*) – 4 character id of the station.
- **year** (*int*) – full year
- **txtfile_part1** (*str, optional*) – input File name for part 1.
- **txtfile_part2** (*str, optional*) – Input filename for part 2.
- **csv** (*bool, optional*) – Set to True if you would like csv in addition to plain txt. default is False.
- **plt** (*bool, optional*) – To print plots to screen or not. default is True.
- **spline_outlier1** (*float, optional*) – Outlier criterion used in first splinefit, before RHdot (m)
- **spline_outlier2** (*float, optional*) – Outlier criterion used in second splinefit, after IF & RHdot (meters)
- **knots** (*integer, optional*) – Knots per day, spline fit only. default is 8.
- **sigma** (*float, optional*) – Simple sigma outlier criterion (e.g. 1 for 1sigma, 3 for 3sigma) default is 2.5
- **extension** (*str, optional*) – Solution subdirectory.
- **rhdot** (*bool, optional*) – Set to True to turn on spline fitting for RHdot correction. default is True.
- **doy1** (*int, optional*) – Initial day of year, default is 1.
- **doy2** (*int, optional*) – End day of year. Default is 366.
- **testing** (*bool, optional*) – Set to False for older code. default is now True.
- **ampl** (*float, optional*) – New amplitude constraint. Default is to do nothing. Now allows a list of amplitude values for the frequencies listed in your json. You have to use the same order of frequencies - but do not have to list them all, i.e. if you had fr 1 20 101 102 and you only wanted to use GPS, you could provide just two ampl values and it would return L1 and L2 GPS. If you had a subdaily value stored before, that is currently restricted to one value for all frequencies. That can be changed if someone submits a PR.
- **azim1** (*int, optional*) – minimum azimuth. Default is 0.
- **azim2** (*int, optional*) – Max azimuth. Default is 360.
- **h1** (*float, optional*) – lowest allowed reflector height in meters. Default is 0.4
- **h2** (*float, optional*) – highest allowed reflector height in meters. Default is 300
- **peak2noise** (*float, optional*) – New peak to noise constraint. Default is 0.
- **kplt** (*bool, optional*) – plot for kristine
- **subdir** (*str, optional*) – name for output subdirectory in REFL_CODE/Files

- **delta_out** (*int, optional*) – how frequently - in seconds - you want smooth spline model output written default is 1800 seconds
- **if_corr** (*bool, option*) – whether you want the inter-frequency removed default is true
- **hires_figs** (*bool, optional*) – whether high resolution figures are made
- **apply_rhdot** (*bool, optional*) – whether you want the RH dot correction applied for a lake or river you would not want it to be.
- **fs** (*int, optional*) – fontsize for Figures. default is 10 for now.
- **alt_sigma** (*bool, optional*) – whether you want to use Nievinski definition for outlier criterion. in part 1 of the code (the crude outlier detector)
- **gap_min_val** (*float, optional*) – removes splinefit values from output txt and plot for gaps bigger than this value, in hours
- **year_end** (*int, optional*) – last year of analysis period.
- **knots2** (*int, optional*) – testing out allowing different knots for last spline
- **gap_flag** (*bool*) – whether you want gaps filled with 999 values in the final spline file
- **date1** (*str*) – avoid doy1/doy2. start time in yyyyymmdd
- **date2** (*str*) – avoid doy1/doy2. end time in yyyyymmdd
- **fundy** (*str*) – name of file with low tide times (MJD)

gnssrefl.tracks module

Build, manage, and tag multi-GNSS satellite tracks.

The on-disk `tracks.json` catalogs the periodic ground tracks visible at a station, keyed by `track_id` and grouped by `(sat, freq)`. Each track holds a list of `epochs` describing the matching parameters (repeat interval, anchor MJD, mean azimuth, drift rate).

The module has two halves: a build side (`build_tracks` and helpers) that collects per-arc geometry and writes the JSON, and a runtime side (`load_tracks_json`, `build_lookup_index`, `lookup_arc`, `attach_track_id`) that tags arcs with their `(track_id, track_epoch)` at extract time. `track_id` is stable across every artifact derived from a tracks-shaped JSON (tracks.json, vwc_tracks.json, per-day phase files, vwc output files).

See `docs/pages/tracks.md` for the full design and workflow.

gnssrefl.tracks.**active_epoch_days**(*tracks_json*)

Set of (year, doy) pairs spanning the union of active epoch windows.

gnssrefl.tracks.**assign_tracks**(*df_freq*, *T_candidates_solar*)

Walk arcs forward in MJD and assign track ids.

Returns (df_sorted, track_ids, match_T) where match_T[i] is the candidate T (in solar days) used to extend arc i, or NaN if arc i seeded a new track.

gnssrefl.tracks.**attach_legacy_apriori**(*arcs*, *station*, *extension=''*)

Tag arcs with legacy GPS-only per-freq apriori_rh_{fr}.txt entries.

Groups arcs by frequency, loads `apriori_rh_{fr}.txt` once per freq, and matches each arc to a track by (satellite, circular azimuth distance <= 3 deg), the same rule used historically by the legacy VWC pipeline.

Sets `meta['apriori_RH']`, `meta['track_azim']`, `meta['track_id']`, and `meta['track_epoch']` on every arc. `apriori_RH` / `track_azim` are `None` on miss; `track_id` / `track_epoch` are `-1` on miss. `track_epoch` is always `0` on match (the legacy path has only one epoch per track).

gnssrefl.tracks.**attach_track_id**(*arcs*, *track_file_path*, *year*, *doy*, *track_cache=None*)

Tag each arc's metadata with track info from tracks-shaped JSON.

Works against both `tracks.json` (the station-wide catalog) and `vwc_tracks.json` (the VWC-eligible filtered subset, which adds a per epoch `apriori_RH` field).

Each `metadata` dict gets these new keys:

`track_id`, `track_epoch` (both -1 on no match), `track_azim` (`az_avg_minel` of the matched epoch, or `None`), `apriori_RH` (matched epoch's `apriori_RH`, or `None`; only present in `vwc_tracks.json`).

Parameters

- **arcs** (*list of (metadata, data) tuples*) – Output of `extract_arcs`, modified in place.
- **track_file_path** (*path-like*) – Path to a tracks-shaped JSON file (`tracks.json` from build_tracks, or `vwc_tracks.json` from `vwc_input`).
- **year** (*int*) – Year and day-of-year of the arcs (used together with each arc's `arc_timestamp` to compute MJD for the lookup).
- **doy** (*int*) – Year and day-of-year of the arcs (used together with each arc's `arc_timestamp` to compute MJD for the lookup).
- **track_cache** (*dict, optional*) – Path-keyed cache of prebuilt lookup indexes. Defaults to the module-level `TRACK_INDEX_CACHE`. Cache entries are never invalidated; restart the process if a tracks file is rewritten on disk.

Returns

The same `arcs` list (modified in place), for chaining.

Return type

list

gnssrefl.tracks.**build_lookup_index**(*tracks_json*)

Build a (sat, freq) -> [(track_id, track_epoch, def_dict), ...] index.

Each `def_dict` carries the fields needed by lookup_arc:

rise, repeat_interval_d, anchor_mjd, az_avg_minel, az_drift_rate, first_mjd, last_mjd, epoch_type

gnssrefl.tracks.**build_tracks**(*station*, *year*, *year_end=None*, *extension=''*, *snr_type=66*, *source='auto'*)

Build tracks.json for a station over [year .. year_end].

Collects per-arc geometry (sat, freq, mjd, azim, rise), folds arcs into periodic tracks, drops fragment tracks below the per-freq filter threshold (10 percent of per-freq median arcs per track), fits a single periodic epoch per surviving track, and writes the JSON.

Two arc sources are supported, selected by `source`:

- **`'snr'` walk SNR files via `load_arcs` (slow, covers every** frequency in the SNR file).
- **`'results'` read `results/` + `failQC/` via** `load_arcs(..., fast=True)` (fast, covers only frequencies gnssir was run with).
- **`'auto'` (default) prefer `'results'` if any `results/` dir is** populated in range; else fall back to `'snr'`.

Parameters

- **station** (*str*) – 4-char station name (lowercase)

- **year** (*int*) – Start year
- **year_end** (*int, optional*) – End year inclusive. Defaults to year.
- **extension** (*str*) – Strategy extension subdirectory. Default ‘’.
- **snr_type** (*int*) – SNR file type for the SNR-walk path. Default 66.
- **source** (*str*) – Arc source: ‘auto’ | ‘results’ | ‘snr’. Default ‘auto’.

Returns

- **tracks_json** (*dict*) – In-memory tracks_json matching the on-disk JSON.
- **arcs_df** (*pandas.DataFrame or None*) – Per-arc DataFrame in the extract_arcs_gnssir_results schema (mjd, azim, constellation, RH, match_T, track_id, track_epoch), or None when source=’snr’ (no RH available).

gnssrefl.tracks.**doy_hour_to_mjd**(*year*, *doy*, *hours*)

Convert (year, doy, fractional hours UTC) to MJD.

gnssrefl.tracks.**fit_segment**(*arcs*)

Fit T and azimuth model for a single track’s arcs.

Returns (T_fit, anchor_mjd, az_avg_minel, az_drift_rate). az_drift_rate is only nonzero for BeiDou (secular azimuth drift).

gnssrefl.tracks.**iso_to_mjd**(*iso_str*)

ISO 8601 ‘YYYY-MM-DDTHH:MM:SSZ’ UTC string -> MJD float.

gnssrefl.tracks.**load_arcs**(*station*, *year*, *year_end*, *extension*, *snr_type=66*, *fast=False*)

Collect per-arc geometry for station across [year..year_end] into a DataFrame.

Returns columns (year, doy, sat, freq, mjd, azim, rise); the fast path additionally carries RH.

- fast=False (default): walk SNR files day by day via extract_arcs_from_station, covering every frequency the SNR file contains.
- fast=True: read the gnssir results/ + failQC/ artifacts via load_results_with_failqc. Orders of magnitude faster, but only covers frequencies gnssir was configured to run, and requires a prior gnssir run with save_failqc=True.

BeiDou GEO/IGSO PRNs in BEIDOU_NON_MEO_SATS are skipped here so the rest of the pipeline never sees them.

gnssrefl.tracks.**load_tracks_json**(*path*)

Load a tracks.json file from disk and return the tracks_json dict.

gnssrefl.tracks.**lookup_arc**(*sat*, *freq*, *obs_time_mjd*, *obs_az_minel*, *track_lookup_index*, *az_tol=5.0*, *time_tol_min=30*)

Look up the (track_id, track_epoch, epoch_entry) for a single arc.

Active matches require the query to fall inside the track’s [first_mjd, last_mjd] interval AND fit the periodic model within time_tol_min and az_tol.

Parameters

- **sat** (*int*) – Satellite number and frequency code identifying the candidate list.
- **freq** (*int*) – Satellite number and frequency code identifying the candidate list.
- **obs_time_mjd** (*float*) – Arc observation time in MJD.

- **obs_az_minel** (*float*) – Arc azimuth at minimum elevation (degrees). Compared against each candidate's drift-corrected expected azimuth.
- **track_lookup_index** (*dict*) – Pre-built (sat, freq) -> [(track_id, track_epoch, entry_dict), ...] candidate index produced by `build_lookup_index(tracks_json)`. Each `entry_dict` carries the epoch's matching parameters (`first_mjd`, `last_mjd`, `anchor_mjd`, `repeat_interval_d`, `az_avg_minel`, `az_drift_rate`, `epoch_type`, `ignored_ranges`).

Returns
(track_id, track_epoch, entry) – `(-1, -1, None)` if no track def covers this arc. `entry` is the matched epoch dict from `build_lookup_index` (keys include `az_avg_minel` and `apriori_RH`) when the match succeeds.

Return type
tuple

`gnssrefl.tracks.`**`mjd_to_iso_ceil`**(*mjd*)

MJD -> ISO 8601 Z UTC string, rounded UP to the nearest second.

`gnssrefl.tracks.`**`mjd_to_iso_floor`**(*mjd*)

MJD -> ISO 8601 Z UTC string, rounded DOWN to the nearest second.

`gnssrefl.tracks.`**`results_dir_has_files`**(*station*, *year*, *year_end*, *extension*)

True if any year in the range has a populated results/ dir for station.

`gnssrefl.tracks.`**`unwrap_az`**(*az*)

Bring all azimuths into a single ±180° window centered on az[0].

`gnssrefl.tracks.`**`warn_legacy_apriori_and_exit`**(*station*, *missing_file*, *extension=''*)

If any GPS `apriori_rh_{fr}.txt` exists, print a `-legacy T` hint and exit.

Called from modern-path entry points when `missing_file` (e.g. `vwc_tracks.json`) is absent.

`gnssrefl.tracks.`**`write_tracks_json`**(*tracks_json*, *f*)

Write tracks_json to file handle `f` with one `ignored_ranges` pair per line.

gnssrefl.tracks_cl module

tracks_cl.py: CLI wrapper for the multi-GNSS tracks.json builder.

Thin argparse wrapper around gnssrefl.tracks.build_tracks. Walks SNR files for the requested station and year window, folds arcs into multi-GNSS ground-truth tracks, and writes the result to `REFL_CODE/Files/{station}/{extension}/tracks.json`.

Examples

generate_tracks mchl 2023 -year_end 2025
build mchl tracks for 2023-2025

generate_tracks mchl 2024 -extension m1
build mchl/m1 tracks (always overwrites any existing tracks.json)

generate_tracks mchl 2024 -source snr
force SNR walk even when results/+failQC/ are available

`gnssrefl.tracks_cl.`**`main`**()

gnssrefl.tracks_cl.**parse_arguments**()

gnssrefl.tracks_qc module

Quality-control edits for tracks.json / vwc_tracks.json files.

Available operations:

- `split_epoch` / `merge_epochs`: subdivide one epoch into two at a chosen MJD, or combine two adjacent epochs back into one. Use when hardware (receiver or satellite) or significant environmental changes occur that motivate a new apriori RH on the same geometric track.
- `ignore_range` / `unignore_range`: mark (or un-mark) a time window within an epoch so its arcs are excluded from fits and stats.
- `deactivate_epoch`: turn an epoch off without deleting it, so downstream tools skip it.
- `delete_track`: drop a track entirely from this file onward.
- `save_tracks`: write the JSON back after refitting every active epoch and (for vwc_tracks files) recomputing `apriori_RH` / `RH_std` from the arcs. Read with `tracks.load_tracks_json`.

Edits are applied to an in-memory dict and only become a self- consistent file after `save_tracks` runs the refit + stats pass. Structurally invalid edits raise `ValueError`.

gnssrefl.tracks_qc.**apply_auto_removal**(*tracks_json*, *fr*, *good_keys*)

Drop bad tracks/epochs at frequency fr via the QC primitives.

For every track whose freq equals fr, walks its active epochs. Epochs whose (track_id, track_epoch) is not in good_keys are removed: the whole track goes via delete_track when none of its active epochs survive, otherwise each bad epoch goes via deactivate_epoch. Tracks on other frequencies are untouched.

Parameters

- **tracks_json** (*dict*) – vwc_tracks.json document (mutated in place).
- **fr** (*int*) – Frequency to filter. Only tracks with track['freq'] == fr are considered.
- **good_keys** (*set of (int, int)*) – (track_id, track_epoch) pairs that passed the run's QC.

Returns

{'tracks_removed', 'tracks_total', 'epochs_deactivated', 'epochs_total'}. Totals count only tracks/epochs at fr.

Return type

dict

gnssrefl.tracks_qc.**check_track_exists**(*tracks_json*, *track_id*)

gnssrefl.tracks_qc.**compute_tracks_stats**(*tracks_json*, *arcs_df*, *apriori_rh_ndays*)

Compute `n_qc_arcs` and apriori_RH/RH_std on active epochs.

Inactive epochs are skipped; their `n_qc_arcs` is zeroed at deactivation time.

gnssrefl.tracks_qc.**deactivate_epoch**(*tracks_json*, *track_id*, *epoch_id*)

Mark an active epoch inactive. Refit/stats will skip it on save.

Zeros `n_arcs` and `n_qc_arcs` so inactive epochs carry the invariant "0 arcs live". `epoch_id` is scoped to this saved tracks_json; see tracks.py module docstring.

gnssrefl.tracks_qc.**delete_track**(*tracks_json*, *track_id*)

Remove a track entirely from tracks_json['tracks'].

track_id is the geometric identity and is stable across saves (see tracks.py module docstring); deleting it means that id is gone from this snapshot onward.

gnssrefl.tracks_qc.**filter_by_ignored_ranges**(*sub*, *ranges*)

Return sub with rows whose mjd falls in any ignored range removed.

gnssrefl.tracks_qc.**find_active_epoch_containing**(*track*, *mjd*)

gnssrefl.tracks_qc.**ignore_range**(*tracks_json*, *track_id*, *epoch_id*, *mjd_start*, *mjd_end*)

Append [mjd_start, mjd_end] to the target active epoch's ignored ranges.

Range must lie inside the epoch window and satisfy mjd_end > mjd_start. epoch_id is scoped to this saved tracks_json; see tracks.py module docstring.

gnssrefl.tracks_qc.**merge_epochs**(*tracks_json*, *track_id*, *epoch_id_a*, *epoch_id_b*)

Merge two adjacent active epochs with matching constellation / match_T.

Window becomes the union; ignored_ranges are concatenated. anchor_time / repeat_interval_d inherit from the earlier epoch and are refit on save. Renumbers all epochs after the merged pair (epoch_id is scoped to this saved tracks_json; see tracks.py module docstring).

gnssrefl.tracks_qc.**recompute_derived_fields**(*tracks_json*, *arcs_df*)

Refit active epochs and refresh every derived per-epoch field.

For file_type == 'vwc_tracks' JSON, also recomputes each epoch's apriori_RH / RH_std / n_qc_arcs. The order matters: recompute_n_arcs and compute_tracks_stats share the same ignored-range mask logic and must agree on every epoch.

gnssrefl.tracks_qc.**recompute_durations**(*tracks_json*)

Refresh duration_d for every epoch from current start/end times.

gnssrefl.tracks_qc.**recompute_metadata_aggregates**(*tracks_json*, *arcs_df*)

Refresh top-level metadata totals and the data time range.

Writes n_tracks, n_epochs, n_arcs (and n_qc_arcs for vwc_tracks) and start_time / end_time / duration_d on metadata. Rebuilds metadata key order so the totals sit together. Removes the legacy nested time_range field.

gnssrefl.tracks_qc.**recompute_n_arcs**(*tracks_json*, *arcs_df*)

Set n_arcs on every active epoch from arcs_df.

Counts arcs whose (track_id, track_epoch) matches the epoch and whose mjd is outside any ignored_ranges. Writes 0 when no arcs match. Inactive epochs are skipped; their n_arcs is zeroed at deactivation time.

gnssrefl.tracks_qc.**refit_active_epochs**(*tracks_json*, *arcs_df*)

For each active epoch, refit anchor_time / repeat_interval_d via fit_segment.

gnssrefl.tracks_qc.**renumber_epoch_ids**(*track*)

gnssrefl.tracks_qc.**reorder_epoch_keys**(*epoch*)

Rewrite epoch in place with keys in canonical order.

gnssrefl.tracks_qc.**require_active_epoch**(*track*, *track_id*, *epoch_id*)

gnssrefl.tracks_qc.**save_tracks**(*tracks_json*, *path*, *tool*, *note=''*, *arcs_df=None*)

Append a history entry, refresh every derived field, and write the JSON.

For `file_type == 'vwc_tracks'` JSON, also recomputes each epoch's `apriori_RH` and `RH_std` from arcs in that epoch's trailing `apriori_rh_ndays` window. Writes atomically (temp file + rename).

`arcs_df` can be passed by callers that already have the walked arcs (avoids re-reading the results files). If `None`, arcs are loaded via *load_gnssir_results_from_tracks*.

gnssrefl.tracks_qc.**split_epoch**(*tracks_json*, *track_id*, *split_mjd*)

Split one active epoch into two adjacent actives at `split_mjd`.

`split_mjd` must lie strictly inside the target epoch's window. Both halves inherit the original epoch's fit parameters (anchor_time, repeat_interval_d, az_avg_minel, az_drift_rate); fresh values come from `save_tracks`' refit pass.

gnssrefl.tracks_qc.**unignore_range**(*tracks_json*, *track_id*, *epoch_id*, *mjd_start*, *mjd_end*)

Subtract `[mjd_start, mjd_end]` from the epoch's ignored ranges.

Raises if the subtraction leaves every existing range unchanged (i.e. no overlap anywhere). `epoch_id` is scoped to this saved tracks_json; see tracks.py module docstring.

gnssrefl.tracks_qc.**validate_epoch_ids**(*tracks_json*)

Enforce that each track's epoch ids are exactly 0..N-1 in order.

Raises ValueError on the first violation.

gnssrefl.utils module

class gnssrefl.utils.**FileManagement**(*station*, *file_type*, *year: int | None = None*, *doy: int | None = None*, *file_not_found_ok: bool = False*, *frequency: int | None = None*, *extension: str = ''*, *snr_type: int | None = None*)

Bases: `object`

FileManagement is designed to easily read the files that this package relies on. Required parameters include station and file_type from FileTypes class. Optional parameters are year, doy, and file_not_found_ok.

find_apriori_rh_file()

Find apriori RH file with backwards compatibility fallback.

Search order: 1. New (registry) format: input/{station}/[{ext}/]{station}_phaseRH_<C>_<label>.txt 2. Pre-registry station-dir format: input/{station}/[{ext}/]{station}_phaseRH_L{1,2,5}.txt 3. Legacy root format: input/{station}_phaseRH[_L1|_L5].txt (L2 has no suffix)

Returns
(file_path, format_type) where format_type is one of 'new_format', 'preregistry', 'legacy'.

Return type
tuple of (Path, str)

find_daily_avg_phase_file()

Find daily average phase file with backwards compatibility fallback.

Search order: - No extension:

1. Files/{station}/{station}_phase.txt (new format)
2. Files/{station}_phase.txt (legacy fallback)

- With extension: 1. Files/{station}/{extension}/{station}_phase.txt (new format) 2. Files/{station}_phase.txt (legacy fallback - no extension separation in legacy)

Returns: (Path, str) - (file_path, format_type)

`find_json_file()`

Find JSON file with backwards compatibility fallback.

Search order (no cross-priority): - No extension:

1. input/{station}/{station}.json (new format)
2. input/{station}.json (legacy fallback)

- With extension: 1. input/{station}/{extension}/{station}.json (new format) 2. input/{station}.{extension}.json (legacy fallback)

Returns: (Path, str) - (file_path, format_type)

`find_snr_file(gzip=None)`

Find an SNR file, optionally converting to match the desired storage format.

A .gz is considered trustworthy only when it is non-empty and the uncompressed sibling is absent. A zero-byte .gz, or coexistence of .gz and the uncompressed original, indicates an interrupted gzip run (successful gzip removes the original on completion). In that case the corpse is unlinked and the uncompressed copy is treated as authoritative.

Parameters
gzip (*bool or None*) – If None (default): find whatever exists, no conversion. If True: prefer .gz. Compress uncompressed files. If False: prefer uncompressed. Decompress .gz files.

Returns
tuple

Return type
(Path, bool) - (file_path, found)

`find_volumetric_water_content_file()`

Find volumetric water content file with backwards compatibility fallback.

Search order: - No extension:

1. Files/{station}/{station}_vwc.txt (new format)
2. Files/{station}_vwc.txt (legacy fallback)

- With extension: 1. Files/{station}/{extension}/{station}_vwc.txt (new format) 2. Files/{station}_vwc.txt (legacy fallback - no extension separation in legacy)

Returns: (Path, str) - (file_path, format_type)

`get_directory_path(ensure_directory=True)`

Get the path of a specific directory from the FileTypes class.

Parameters
ensure_directory (*bool, optional*) – If True, creates the directory if it doesn't exist. Default is True.

Returns
Directory path requested as a Path object

Return type
Path

`get_file_path`(*ensure_directory=True*)

Get the path of a specific file from the FileTypes class.

Parameters
ensure_directory (*bool, optional*) – If True, creates the parent directory if it doesn't exist. Default is True.

Returns
File path requested as a Path object

Return type
Path

`get_individual_tracks_path`()

Per-frequency individual_tracks directory, nested under vwc_outputs so concurrent freqs don't clobber each other.

Directory structure: - No extension: Files/{station}/vwc_outputs/{label}/individual_tracks/ - With extension: Files/{station}/{extension}/vwc_outputs/{label}/individual_tracks/

`label` is `f{freq}{get_file_suffix}` (e.g. `f20_G_L2C`).

`get_tracks_file_path`()

Path to the multi-GNSS tracks.json file.

Directory structure: - No extension: Files/{station}/tracks.json - With extension: Files/{station}/{extension}/tracks.json

`get_vwc_tracks_file_path`()

Path to the vwc_tracks.json file written by `vwc_input`.

Same schema as `tracks.json` with one added per-epoch field (`apriori_RH`). Consumed by `phase` and `vwc`.

Directory structure: - No extension: Files/{station}/vwc_tracks.json - With extension: Files/{station}/{extension}/vwc_tracks.json

`read_file`(*transpose=False*, ***kwargs*)

Reads the requested file amd returns results of file as an array. Can use transpose parameter to transpose the results.

`class` gnssrefl.utils.**FileTypes**(*value*)

Bases: `str`, `Enum`

Files to either read from or save to.

apriori_rh_file = 'apriori_rh_file'

arcs_directory = 'arcs_directory'

daily_avg_phase_results = 'daily_avg_phase_results'

directory = 'directory'

gnssir_failqc_result = 'gnssir_failqc_result'

gnssir_result = 'gnssir_result'

individual_tracks = 'individual_tracks'

make_json = 'make_json'

phase_file = 'phase_file'

raw_phase_file = 'raw_phase_file'

snr_file = 'snr_file'

tracks_file = 'tracks_file'

volumetric_water_content = 'volumetric_water_content'

vwc_outputs = 'vwc_outputs'

vwc_tracks_file = 'vwc_tracks_file'

gnssrefl.utils.**check_arc_quality**(*meta*, *peak_rh*, *max_amp*, *noise*, *station_config*)

Apply QC filters to a single arc. Returns (passed, fail_reason).

gnssrefl.utils.**check_environment**()

gnssrefl.utils.**circular_distance_deg**(*a*, *b*)

Shortest angular distance between two azimuths in degrees.

Works with scalars and numpy arrays (via broadcasting).

gnssrefl.utils.**circular_mean_deg**(*angles*)

Circular mean of angles in degrees, handling the 0/360 wrap correctly.

gnssrefl.utils.**format_qc_summary**(*freq*, *n_total*, *qc_counts*, *n_saved*)

Build condensed QC summary showing only filters that rejected arcs.

gnssrefl.utils.**get_sys**()

gnssrefl.utils.**pre_check_arc**(*meta*, *station_config*)

Quick QC check using arc metadata only. Returns (passed, fail_reason).

Checks ediff and delT — no LSP results needed, so call before strip_compute.

gnssrefl.utils.**read_files_in_dir**(*directory*, *transpose=False*)

Read all files in a given directory. Directory given must be an absolute path. Returns an n-d array of results. Can use optional parameter transpose to transpose the results.

gnssrefl.utils.**set_environment**(*refl_code*, *orbits*, *exe*)

gnssrefl.utils.**str2bool**(*args*, *expected_bools*)

gnssrefl.utils.**validate_input_datatypes**(*obj*, ***kwargs*)

gnssrefl.veg_multiyr module

kristine larson combine multiple years of teqc multipath metrics, write a file, and make a plot

gnssrefl.veg_multiyr.**in_winter**(*d*)

(td testing autodoc api generation)

pretty silly winter screen tool

Parameters
d (*int*) – day of year

Returns
True if doy is in winter, False if not considered "winter".

Return type
bool

gnssrefl.veg_multiyr.**main**()

command line interface for download_rinex

gnssrefl.veg_multiyr.**newvegplot**(*vegout*, *station*)

send the file name and try to make a plot segreating for changes in teqc metric and receiver type

gnssrefl.veg_multiyr.**vegoutfile**(*station*)

make sure directories exist for prelim veg output file returns name of the otuput file

gnssrefl.vwc_cl module

gnssrefl.vwc_cl.**get_vwc_tracks_freqs**(*station*, *extension=''*)

Return sorted list of unique freq codes in vwc_tracks.json, or None if missing.

gnssrefl.vwc_cl.**iter_vwc_epochs**(*vwc_tracks*)

Yield (track_id, track_epoch, epoch, track) for every active epoch in a vwc_tracks.json document.

gnssrefl.vwc_cl.**load_vwc_tracks**(*vwc_tracks_path*)

Read a vwc_tracks.json document from disk.

gnssrefl.vwc_cl.**main**()

gnssrefl.vwc_cl.**parse_arguments**()

gnssrefl.vwc_cl.**process_vwc_from_tracks**(*station*, *year*, *year_end*, *fr*, *extension*, *tracks*, *vwc_tracks*, *plt*, *screenstats*, *min_req_pts_track*, *polyorder*, *minvalperday*, *bin_hours*, *minvalperbin*, *bin_offset*, *snow_filter*, *tmin*, *tmax*, *warning_value*, *auto_removal*, *hires_figs*, *advanced*, *veg_model*, *save_tracks*, *level_doys*, *skip_leveling*, *colors*, *bx*, *by*, *fs*, *snow_file*)

Run the shared VWC pipeline on a prepared tracks array.

Both the default multi-GNSS flow (vwc) and the GPS-only legacy flow (vwc_legacy) build a tracks array their own way and delegate the identical downstream work (set_parameters, snow filter, per-track phase fit, vegetation correction, leveling, output writing) to this function.

Parameters

- **tracks** (*np.ndarray*) – Track array with shape (N, 7). Cols 0-4 carry, in both paths: idx, mean RH (m), sat, track avg azimuth (deg), nvalstrack. Cols 5-6 carry (az_min, az_max)

for the legacy path and (track_id, track_epoch) for the default path; the choice is selected by vwc_tracks.

- **vwc_tracks** (*dict or None*) – Loaded vwc_tracks.json document for the default path (triggers tag-based arc matching via (track_id, track_epoch)), or None for the legacy path (triggers sat + azimuth-proximity matching).

gnssrefl.vwc_cl.**vwc**(*station: str*, *year: int*, *year_end: int | None = None*, *fr: int | None = None*, *plt: bool = True*, *screenstats: bool = False*, *min_req_pts_track: int | None = None*, *polyorder: int = -99*, *minvalperday: int | None = None*, *bin_hours: int | None = None*, *minvalperbin: int | None = None*, *bin_offset: int | None = None*, *snow_filter: bool = False*, *tmin: float | None = None*, *tmax: float | None = None*, *warning_value: float | None = None*, *auto_removal: bool = False*, *hires_figs: bool = False*, *advanced: bool = False*, *vegetation_model: int | None = None*, *save_tracks: bool = False*, *extension: str | None = None*, *level_doys: list = []*, *skip_leveling: bool = False*, *legacy: bool = False*)

Compute volumetric water content (VWC) from GNSS-IR phase estimates across all constellations/frequencies selected during vwc_input.

Reads vwc_tracks.json to get each arc's apriori RH and track-mean azimuth, then concatenates the daily phase files, plots phase by azimuth quadrant, bins by time, applies vegetation correction, and converts to VWC. Pass -legacy T to use the old GPS-only flow that matches arcs against apriori_rh_{fr}.txt by satellite + azimuth.

The code supports two vegetation correction models: Model 1 (simple, default) uses amplitude-based correction suitable for sites with bare soil or sparse vegetation. Model 2 (advanced) uses the Chew et al. (2016) GPS Solutions (DOI: 10.1007/s10291-015-0462-4) algorithm, and should be used for sites with dense or tall vegetation.

Code now allows inputs (minvalperday, tmin, and tmax) to be stored in the gnssir analysis json file. To avoid confusion, in the json they are called vwc_minvalperday, vwc_min_soil_texture, and vwc_max_soil_texture. These values can also be overwritten on the command line ...

Code now allows doy values for level nodes to be set in the json (variable name vwc_level_doys) and it is a list, i.e. [170,230] would be the input for summer months in NOAM. This can be overriden by the command line -level_inputs 170 230. If no information is provided by the user, it has defaults based on the station being in the northern or southern hemisphere. However, this is a hack, and I have no idea if it works well for Australia, e.g. It worked well for PBO H2O in the semi-arid western U.S. Those values are defined in set_parameters in phase_functions.py if you want to take a look If you want a time period that crosses the year boundary (i.e. you want all of december and january), you can input level_doys of 335 and 31, and the code will process accordingly.

Examples

vwc p038 2017
one year for station p038

vwc p038 2015 -year_end 2017
three years of analysis for station p038

vwc p038 2015 -year_end 2017 -warning_value 6
warns you about tracks with phase RMS greater than 6 degrees rms

vwc p038 2015 -year_end 2017 -warning_value 6 -auto_removal T
makes new list of tracks based on your new warning value

Parameters

- **station** (*str*) – 4 character ID of the station
- **year** (*int*) – full Year
- **year_end** (*int, optional*) – last year for analysis

- **fr** (*int, optional*) – Single GNSS frequency to process. main() handles the CLI -fr list (one/multiple/omitted) and calls this function once per freq; callers using vwc() directly must pass a single value. Only GPS is officially supported; other constellations are available for research/testing only.
- **plt** (*bool, optional*) – Whether to produce plots to the screen. Default is True
- **min_req_pts_track** (*int, optional*) – how many points needed to keep a satellite track. Default is 30. Can be set in the gnssir_input json (vwc_min_req_pts_track).
- **polyorder** (*int*) – polynomial order used for leveling. Usually the code picks it but this allows to users to override. Default is -99 which means let the code decide
- **minvalperday** (*integer*) – how many phase measurements are needed for each daily measurement default is 10
- **snow_filter** (*bool*) – whether you want to attempt to remove points contaminated by snow default is False
- **tmin** (*float*) – minimum soil texture value, default 0.05. This can now be set in the gnssir_input json (with **vwc_** added)
- **tmax** (*float*) – maximum soil texture value, default 0.5. This can now be set in the gnssir_input json (with **vwc_** added)
- **warning_value** (*float*) – screen warning about bad tracks (phase rms, in degrees). default is 5.5
- **auto_removal** (*bool, optional*) – whether to automatically remove tracks that hit your bad track threshold default value is false
- **hires_figs** (*bool, optional*) – whether to make eps instead of png files default value is false
- **advanced** (*bool, optional*) – shorthand for vegetation_model=2
- **vegetation_model** (*int, optional*) – vegetation correction model: 1=simple (default), 2=advanced (Chew et al. 2016) can be set in gnssir analysis JSON as vwc_vegetation_model
- **save_tracks** (*bool, optional*) – save individual track VWC data to files (advanced model only)
- **extension** (*str*) – extension used when you made the json analysis file
- **level_doys** (*list*) – pair of day of years that are used to define time period for "leveling" default is north american summer
- **skip_leveling** (*bool, optional*) – internal use only - skip leveling and return percentage units for unified leveling in vwc_hourly. Default is False.
- **legacy** (*bool, optional*) – Use the legacy GPS-only apriori_RH flow. Default is False. Deprecated; may be removed after 2027-01-01.

Returns

- *Daily phase results in a file at* – $REFL_CODE/Files/<station>/[<extension>/]vwc_outputs/<label>/<station>_phase
  with columns: Year DOY Ph Phsig NormA MM DD (<label> is f<fr>_<C>_<signal>, e.g.
  f20_G_L2C; bin_offset lives in the file header)
- *VWC results in a file at* – $REFL_CODE/Files/<station>/[<extension>/]vwc_outputs/<label>/<station>_vwc_<bin_h
  with columns: FracYr Year DOY VWC Month Day

gnssrefl.vwc_cl.**vwc_legacy**(*station*, *year*, *year_end*, *fr*, *plt*, *screenstats*, *min_req_pts_track*, *polyorder*, *minvalperday*, *bin_hours*, *minvalperbin*, *bin_offset*, *snow_filter*, *tmin*, *tmax*, *warning_value*, *auto_removal*, *hires_figs*, *advanced*, *veg_model*, *save_tracks*, *extension*, *level_doys*, *skip_leveling*, *colors*, *bx*, *by*, *fs*, *snow_file*)

Legacy GPS-only VWC flow.

Reads apriori_rh_{fr}.txt (the legacy per-freq track list), then delegates the downstream pipeline to process_vwc_from_tracks, which matches each phase-file row to a track by (satellite, azimuth within 3 deg) when no vwc_tracks.json is supplied.

gnssrefl.vwc_hourly module

VWC Hourly Rolling Module

This module generates VWC estimates from n-hour windows that start at every hour throughout the day, creating a rolling/sliding time window dataset.

For example, with 6-hour windows (bin_hours=6), this creates VWC estimates from:

- Window 00:00-06:00
- Window 01:00-07:00
- Window 02:00-08:00
- … continuing through each hour of the day

gnssrefl.vwc_hourly.**combine_and_level_vwc_data**(*all_vwc_data*, *tmin*, *level_doys*, *polyorder*, *station*, *extension*, *fr*, *bin_hours*)

Combine unleveled VWC data from multiple offsets and apply unified leveling.

This solves the bias problem in Model 1 by ensuring all offsets share the same leveling baseline calculation. See GitHub issue #358.

Parameters

- **all_vwc_data** (*list of dict*) – List of vwc_data dicts from each offset (PERCENTAGE units, 0-60)
- **tmin** (*float*) – Minimum soil texture value (e.g., 0.05)
- **level_doys** (*list*) – [start_doy, end_doy] for dry season baseline window
- **polyorder** (*int*) – Polynomial order for leveling (-99 = auto)
- **station** (*str*) – Station name
- **extension** (*str*) – File extension
- **fr** (*int*) – Frequency code
- **bin_hours** (*int*) – Time bin size in hours

Returns

vwc_data – Combined and leveled data with keys: 'mjd', 'vwc' (DECIMAL units), 'datetime', 'bin_starts'

Return type

dict

gnssrefl.vwc_hourly.**generate_rolling_vwc_from_tracks**(*station*, *fr*, *bin_hours*, *minvalperbin*, *extension=''*, *year=None*, *year_end=None*)

Generate complete hourly rolling VWC dataset from saved track files.

Loads all track data once and creates bins at every hour (0:00, 1:00, 2:00, etc.) for the entire dataset.

Parameters

- **station** (*str*) – 4-character station name
- **fr** (*int*) – Frequency code (20 for L2C, etc.)
- **bin_hours** (*int*) – Time bin size in hours (e.g., 6 for 6-hour windows)
- **minvalperbin** (*int*) – Minimum tracks required per time bin
- **extension** (*str*) – Extension for file paths (default: '')
- **year** (*int, optional*) – Start year for filtering track files
- **year_end** (*int, optional*) – End year for filtering track files

Returns

vwc_data – Dictionary with 'mjd', 'vwc', 'datetime', 'bin_starts' (NOT leveled yet) Returns None if no track data found

Return type

dict or None

gnssrefl.vwc_hourly.**main**()

Alternative entry point (for compatibility)

gnssrefl.vwc_hourly.**main_hourly**()

CLI entry point for vwc_hourly command

gnssrefl.vwc_hourly.**parse_arguments_hourly**()

Parse command line arguments for vwc_hourly command

gnssrefl.vwc_hourly.**plot_hourly_vs_daily_vwc**(*station*, *fr*, *bin_hours*, *extension=''*)

Plot hourly rolling VWC (gray dots) vs daily VWC (bold red) for comparison.

Requires both daily and hourly VWC files to exist.

gnssrefl.vwc_hourly.**vwc_hourly**(*station: str*, *year: int*, *fr: int*, *year_end: int | None = None*, *plt: bool = True*, *bin_hours: int = 6*, *minvalperbin: int = 5*, *min_req_pts_track: int | None = None*, *polyorder: int = -99*, *snow_filter: bool = False*, *tmin: float | None = None*, *tmax: float | None = None*, *warning_value: float | None = None*, *auto_removal: bool = False*, *hires_figs: bool = False*, *advanced: bool = False*, *vegetation_model: int | None = None*, *extension: str = ''*, *level_doys: list = []*)

Generate VWC estimates from n-hour windows that start at every hour throughout the day, creating a rolling/sliding time window dataset.

For example, with bin_hours=6, this creates VWC estimates from: - Window 00:00-06:00 - Window 01:00-07:00 - Window 02:00-08:00 - … continuing through the day

Warning: This function is EXPERIMENTAL. Only daily (24-hour) VWC measurements are officially supported. Subdaily/hourly results should be used with caution and are provided for research purposes only.

Vegetation Model 1 (simple):
Uses two-pass processing: first collects unleveled VWC from all offsets, then applies unified leveling to the combined dataset. This ensures all offsets share the same leveling baseline.

Vegetation Model 2 (advanced):
Runs vwc() once with -save_tracks to generate track files, then aggregates those saved tracks into different time bins with unified leveling.

Examples

vwc_hourly p038 2022
6-hour rolling bins for station p038 (default, model 1)

vwc_hourly p038 2022 -bin_hours 12 -minvalperbin 5
12-hour rolling bins with minimum 5 tracks per bin

vwc_hourly okl2 2012 -vegetation_model 2
6-hour rolling bins using advanced vegetation model (model 2)

Parameters

- **station** (*str*) – 4 character ID of the station
- **year** (*int*) – full Year
- **year_end** (*int, optional*) – last year for analysis
- **fr** (*int*) – GNSS frequency code (e.g. 20 for GPS L2C, 101 for GLONASS L1). Must be a single freq present in vwc_tracks.json. The CLI auto-loops over multiple freqs by calling this function once per freq.
- **bin_hours** (*int, optional*) – time bin size in hours (1,2,3,4,6,8,12). Default is 6
- **minvalperbin** (*int, optional*) – min number of satellite tracks needed per time bin. Default is 5
- **level_doys** (*list, optional*) – pair of day of years for baseline leveling period
- **function)** (*(other parameters same as vwc*) –

Return type
Creates hourly rolling VWC file under $REFL_CODE/Files/<station>/vwc_outputs/<freq_label>/<station>_vwc_rolling

gnssrefl.vwc_input module

gnssrefl.vwc_input.**build_vwc_tracks**(*tracks*, *arcs_df*, *min_req_pts_track*, *fr_list*, *apriori_rh_ndays*)

Build a vwc_tracks dict from a (comprehensive) tracks dict.

Works on a copy of tracks: drops tracks whose freq is not in fr_list via delete_track; deactivates (track_id, epoch_id) buckets with fewer than min_req_pts_track QC-passing arcs in arcs_df via deactivate_epoch; drops tracks left with no active epochs. Sets metadata['file_type'] = 'vwc_tracks' and metadata['apriori_rh_ndays']. Per-epoch derived fields (apriori_RH, RH_std, n_arcs, n_qc_arcs) are populated by save_tracks.

The input tracks dict is not mutated.

gnssrefl.vwc_input.**define_track_clusters**(*azimuths*, *gap_threshold=10*)

Define satellite track clusters from observed azimuths.

Groups azimuths into distinct tracks by sorting them on the circle, finding the largest gap (natural break), then splitting at any gap exceeding the threshold. Handles 0/360 wrap.

Parameters

- **azimuths** (*array-like*) – Observed azimuth values in degrees for a single satellite
- **gap_threshold** (*float*) – Minimum azimuth gap in degrees to split into separate tracks. Default 10.

Returns

Each array contains the azimuths belonging to one track.

Return type

list of numpy arrays

gnssrefl.vwc_input.**main**()

gnssrefl.vwc_input.**parse_arguments**()

gnssrefl.vwc_input.**vwc_input**(*station: str*, *year: int*, *fr=None*, *min_req_pts_track: int | None = None*, *minvalperday: int | None = None*, *bin_hours: int | None = None*, *minvalperbin: int | None = None*, *bin_offset: int | None = None*, *extension: str = ''*, *tmin: float | None = None*, *tmax: float | None = None*, *warning_value: float | None = None*, *year_end: int | None = None*, *apriori_rh_ndays: int | None = None*, *legacy: bool = False*, *vwc_tracks_builder=None*)

Sets inputs for the estimation of VWC (volumetric water content).

Auto-builds tracks.json for the station if it doesn't already exist, walks every day in [year, year_end] calling extract_arcs with that tracks.json so each arc is tagged with its (track_id, track_epoch) and gnssir QC result, aggregates per (track_id, track_epoch), and writes the VWC-eligible subset as vwc_tracks.json. That file has the same schema as tracks.json with two added per-epoch fields: apriori_RH (the mean RH across QC-passing arcs in the last apriori_rh_ndays days of the range, default 365) and RH_std (the standard deviation over the same sample, or null when fewer than 3 samples are available).

Downstream phase and vwc commands consume vwc_tracks.json directly to get each arc's apriori RH and track-mean azimuth.

Pass -legacy T to fall back to the old GPS-only flow that produces apriori_rh_{fr}.txt from per-day gnssir result files. -year_end is not supported with -legacy T.

Parameters

- **station** (*str*) – 4 character ID of the station
- **year** (*int*) – full year
- **fr** (*int or list of int, optional*) – Default path: frequency filter. Only tracks whose freq is in this list are written to vwc_tracks.json. When None (default), all frequencies present in tracks.json are kept. Legacy path: single frequency (1 L1, 20 L2C, 5 L5). Only L2C is officially supported; passing a list of length > 1 with legacy=True is an error.
- **min_req_pts_track** (*int, optional*) – Minimum number of arcs per (track_id, track_epoch) bucket required to keep that epoch. When None, the station JSON value of vwc_min_req_pts_track is used if present, otherwise 30. The resolved value is persisted back to the station JSON.
- **minvalperday** (*int, optional*) – how many unique tracks are needed to compute a valid VWC measurement for a given day

- **extension** (*str, optional*) – strategy extension value (same as used in gnssir, subdaily etc)
- **tmin** (*float, optional*) – minimum soil moisture value
- **tmax** (*float, optional*) – maximum soil moisture value
- **warning_value** (*float, optional*) – for removing low quality satellite tracks when running vwc phase units, degrees
- **year_end** (*int, optional*) – End year (inclusive). Defaults to year. Not supported with legacy=True.
- **apriori_rh_ndays** (*int, optional*) – Window size, in days, counted backwards from the end of the date range, over which apriori_RH and RH_std are averaged. Default 365.
- **legacy** (*bool, optional*) – When True, use the legacy GPS-only apriori_RH flow (writes apriori_rh_{fr}.txt from gnssir result files). Default: False.

Returns

- *Default path* – Files/{station}/{extension}/vwc_tracks.json
- *Legacy path* – $REFL_CODE/input/<station>/[<extension>/]<station>_phaseRH_<C>_<label>.txt (the per-freq apriori_rh_{fr}.txt consumed by legacy phase/vwc; <C> is the constellation char and <label> the signal label, e.g. _G_L2C)
- *Both paths also persist the VWC-specific parameters (vwc_minvalperday,*
- *vwc_min_soil_texture, vwc_max_soil_texture, vwc_min_req_pts_track,*
- *vwc_warning_value) into the gnssir analysis JSON.*

gnssrefl.vwc_input.**vwc_input_legacy**(*station*, *year*, *fr*, *min_req_pts_track*, *minvalperday*, *bin_hours*, *minvalperbin*, *bin_offset*, *extension*, *tmin*, *tmax*, *warning_value*)

Legacy GPS-only VWC input pipeline.

Loads the per-day gnssir result files for year, clusters arcs into tracks by satellite + azimuth, and writes the selected tracks to apriori_rh_{fr}.txt (the file name depends on the single requested frequency). Downstream phase -legacy T / vwc -legacy T consume that file to get each arc's apriori RH.

gnssrefl.xyz2llh module

gnssrefl.xyz2llh.**main**()

Converts Cartesian coordinates to latitude, longitude, ellipsoidal height. Prints to screen

Example

xyz2llh -1283634.1615 -4726427.8931 4074798.0429
returns 39.949492042 -105.194266387 1728.856

Parameters

- **x** (*float*) – X coordinate (m)
- **y** (*float*) – Y coordinate (m)
- **z** (*float*) – Z coordinate (m)

gnssrefl.ydoy module

gnssrefl.ydoy.**main**()

converts year/day of year to month and day. prints to the screen

Parameters

- **year** (*int*) – full year
- **doy** (*int*) – day of year

Returns

- **month** (*int*)
- **day** (*int*)

gnssrefl.ymd module

gnssrefl.ymd.**main**()

converts year,month, day to day of year and prints it to the screen

MJD is an optional output

Parameters

- **year** (*int*) – 4 ch year
- **month** (*int*) – calendar month
- **day** (*int*) – calendar day
- **mjd** (*str*) – use T or True to get MJD printed to the screen

Returns

doy – three character day of the year

Return type

str

Note: These docs are still under development; feedback via pull requests is appreciated.

PYTHON MODULE INDEX

g

gnssrefl, 51
gnssrefl.advanced_vegetation_correction, 51
gnssrefl.check_rinex_file, 54
gnssrefl.computemp1mp2, 54
gnssrefl.daily_avg, 56
gnssrefl.daily_avg_cl, 58
gnssrefl.download_ioc, 61
gnssrefl.download_noaa, 61
gnssrefl.download_orbits, 64
gnssrefl.download_psmsl, 65
gnssrefl.download_rinex, 65
gnssrefl.download_teqc, 67
gnssrefl.download_tides, 68
gnssrefl.download_unr, 69
gnssrefl.download_wsv, 69
gnssrefl.EGM96, 51
gnssrefl.extract_arcs, 69
gnssrefl.filesizes, 74
gnssrefl.fundy, 75
gnssrefl.gnss_frequencies, 75
gnssrefl.gnssir_cl, 77
gnssrefl.gnssir_input, 80
gnssrefl.gnssir_v2, 83
gnssrefl.gps, 89
gnssrefl.gpsweek, 126
gnssrefl.highrate, 126
gnssrefl.installexe_cl, 129
gnssrefl.invsnr_cl, 130
gnssrefl.invsnr_input, 132
gnssrefl.karnak_libraries, 133
gnssrefl.kelly, 137
gnssrefl.llh2xyz, 137
gnssrefl.make_meta, 137
gnssrefl.max_resolve_RH_cl, 139
gnssrefl.mjd, 140
gnssrefl.nmea2snr, 140
gnssrefl.nmea2snr_cl, 144
gnssrefl.nyquist_libs, 146
gnssrefl.phase_functions, 146
gnssrefl.pickle_dilemma, 159
gnssrefl.prn2gps, 159
gnssrefl.query_unr, 159
gnssrefl.quicklib, 165
gnssrefl.quickLook_cl, 159
gnssrefl.quickLook_function2, 161
gnssrefl.quickPhase, 163
gnssrefl.quickplt, 165
gnssrefl.read_snr_files, 168
gnssrefl.refl_zones, 169
gnssrefl.refl_zones_cl, 173
gnssrefl.refraction, 174
gnssrefl.retrieve_rh, 181
gnssrefl.rh_plot, 182
gnssrefl.rinex2snr, 182
gnssrefl.rinex2snr_cl, 188
gnssrefl.rinex3_rinex2, 193
gnssrefl.rinex3_snr, 193
gnssrefl.rinex_coords, 193
gnssrefl.rinpy, 194
gnssrefl.rt_rinex3_snr, 196
gnssrefl.sd_libs, 197
gnssrefl.simple_vegetation_correction, 204
gnssrefl.smoosh, 205
gnssrefl.smoosh_snr, 205
gnssrefl.snow_functions, 206
gnssrefl.snowdepth_cl, 208
gnssrefl.snrfile_functions, 209
gnssrefl.spline_functions, 210
gnssrefl.subdaily, 217
gnssrefl.subdaily_cl, 221
gnssrefl.tracks, 224
gnssrefl.tracks_cl, 227
gnssrefl.tracks_qc, 228
gnssrefl.utils, 230
gnssrefl.veg_multiyr, 234
gnssrefl.vwc_cl, 234
gnssrefl.vwc_hourly, 237
gnssrefl.vwc_input, 239
gnssrefl.xyz2llh, 241
gnssrefl.ydoy, 242
gnssrefl.ymd, 242

INDEX

A

a (*gnssrefl.gps.wgs84 attribute*), 121
active_epoch_days() (*in module gnssrefl.tracks*), 224
advanced_vegetation_filter() (*in module gnssrefl.advanced_vegetation_correction*), 51
all_default_frequencies() (*in module gnssrefl.gnss_frequencies*), 75
all_frequencies() (*in module gnssrefl.gnss_frequencies*), 75
angle_range_positive() (*in module gnssrefl.nmea2snr*), 140
another_gfz_orbits() (*in module gnssrefl.gps*), 89
apply_auto_removal() (*in module gnssrefl.tracks_qc*), 228
apply_new_azim_mask() (*in module gnssrefl.fundy*), 75
apply_new_constraints() (*in module gnssrefl.subdaily*), 217
apply_refraction() (*in module gnssrefl.extract_arcs*), 69
apply_vegetation_model() (*in module gnssrefl.advanced_vegetation_correction*), 52
apply_vwc_leveling() (*in module gnssrefl.phase_functions*), 146
apriori_file_exist() (*in module gnssrefl.phase_functions*), 147
apriori_rh_file (*gnssrefl.utils.FileTypes attribute*), 232
arc_plots() (*in module gnssrefl.spline_functions*), 210
arcs_directory (*gnssrefl.utils.FileTypes attribute*), 232
asknewet() (*in module gnssrefl.refraction*), 176
assign_tracks() (*in module gnssrefl.tracks*), 224
attach_gnssir_processing_results() (*in module gnssrefl.extract_arcs*), 69
attach_legacy_apriori() (*in module gnssrefl.tracks*), 224
attach_phase_processing_results() (*in module gnssrefl.extract_arcs*), 70
attach_track_id() (*in module gnssrefl.tracks*), 224
attach_vwc_track_results() (*in module gnssrefl.extract_arcs*), 70
avoid_cddis() (*in module gnssrefl.gps*), 89
azimuth_angle() (*in module gnssrefl.gps*), 90
azimuth_diff() (*in module gnssrefl.nmea2snr*), 140
azimuth_diff1() (*in module gnssrefl.nmea2snr*), 140
azimuth_diff2() (*in module gnssrefl.nmea2snr*), 140
azimuth_mean() (*in module gnssrefl.nmea2snr*), 140

B

back2thefuture() (*in module gnssrefl.gps*), 90
better_high_low_tide() (*in module gnssrefl.fundy*), 75
bfg_data() (*in module gnssrefl.gps*), 90
bfg_password() (*in module gnssrefl.gps*), 90
big_Disk_in_DC_hourly() (*in module gnssrefl.gps*), 91
big_Disk_work_hard() (*in module gnssrefl.gps*), 91
binary() (*in module gnssrefl.gps*), 91
bkg_highrate() (*in module gnssrefl.highrate*), 126
bkg_highrate_tar() (*in module gnssrefl.highrate*), 127
build_lookup_index() (*in module gnssrefl.tracks*), 225
build_tracks() (*in module gnssrefl.tracks*), 225
build_vwc_tracks() (*in module gnssrefl.vwc_input*), 239

C

c (*gnssrefl.gps.constants attribute*), 94
c (*gnssrefl.snrfile_functions.constants attribute*), 209
calcAzEl_new() (*in module gnssrefl.refl_zones*), 169
calcAzEl_newish() (*in module gnssrefl.refl_zones*), 169
calculate_avg_phase() (*in module gnssrefl.phase_functions*), 147
cdate2nums() (*in module gnssrefl.gps*), 91
cdate2ydoy() (*in module gnssrefl.gps*), 91
cddis_download_2022B() (*in module gnssrefl.gps*), 91
cddis_download_2022B_new() (*in module gnssrefl.gps*), 92
cddis_highrate() (*in module gnssrefl.highrate*), 127
cddis_highrate_tar() (*in module gnssrefl.highrate*), 127
cddis_password() (*in module gnssrefl.gps*), 92
cddis_restriction() (*in module gnssrefl.gps*), 92
char_month_converter() (*in module gnssrefl.gps*), 92

check_arc_quality() (*in module gnssrefl.utils*), 233
check_azim_compliance() (*in module gnssrefl.gnssir_v2*), 83
check_azimuth_compliance() (*in module gnssrefl.extract_arcs*), 70
check_directories() (*in module gnssrefl.computemp1mp2*), 54
check_environ_variables() (*in module gnssrefl.gps*), 93
check_environment() (*in module gnssrefl.utils*), 233
check_inputs() (*in module gnssrefl.gps*), 93
check_l2() (*in module gnssrefl.check_rinex_file*), 54
check_navexistence() (*in module gnssrefl.gps*), 93
check_offsets() (*in module gnssrefl.make_meta*), 137
check_rinex_file() (*in module gnssrefl.check_rinex_file*), 54
check_track_exists() (*in module gnssrefl.tracks_qc*), 228
checkEGM() (*in module gnssrefl.gps*), 92
checkexist() (*in module gnssrefl.installexe_cl*), 129
checkFiles() (*in module gnssrefl.gps*), 93
circular_distance_deg() (*in module gnssrefl.utils*), 233
circular_mean_deg() (*in module gnssrefl.utils*), 233
collapse_rinex3_obs() (*in module gnssrefl.rinpy*), 194
colorful() (*in module gnssrefl.quickLook_function2*), 161
combine_and_level_vwc_data() (*in module gnssrefl.vwc_hourly*), 237
compute_tracks_stats() (*in module gnssrefl.tracks_qc*), 228
confused_obstimes() (*in module gnssrefl.gps*), 93
constants (*class in gnssrefl.gps*), 94
constants (*class in gnssrefl.snrfile_functions*), 209
conv2snr() (*in module gnssrefl.rinex2snr*), 182
convert_Hdates_mjd() (*in module gnssrefl.gnssir_v2*), 83
corr_el_angles() (*in module gnssrefl.refraction*), 176
correct_elevations() (*in module gnssrefl.refraction*), 176
count_result_arcs() (*in module gnssrefl.gnssir_cl*), 77
count_result_arcs() (*in module gnssrefl.quickPhase*), 163
crx2rnx() (*in module gnssrefl.gps*), 94

D

daily_avg() (*in module gnssrefl.daily_avg_cl*), 58
daily_avg_phase_results (*gnssrefl.utils.FileTypes attribute*), 232
daily_avg_stat_plots() (*in module gnssrefl.daily_avg*), 56
datestring_mjd() (*in module gnssrefl.gps*), 94
datetime2gps() (*in module gnssrefl.spline_functions*), 210
deactivate_epoch() (*in module gnssrefl.tracks_qc*), 228
dec31() (*in module gnssrefl.gps*), 94
define_inputfile() (*in module gnssrefl.spline_functions*), 211
define_logdir() (*in module gnssrefl.gps*), 94
define_quick_filename() (*in module gnssrefl.gps*), 94
define_track_clusters() (*in module gnssrefl.vwc_input*), 239
delete_track() (*in module gnssrefl.tracks_qc*), 228
dH_curve() (*in module gnssrefl.refraction*), 177
diffraction_correction() (*in module gnssrefl.gps*), 95
directory (*gnssrefl.utils.FileTypes attribute*), 232
dmpf_dh() (*in module gnssrefl.refraction*), 177
download_chmod_move() (*in module gnssrefl.installexe_cl*), 129
download_ioc() (*in module gnssrefl.download_ioc*), 61
download_noaa() (*in module gnssrefl.download_noaa*), 61
download_orbits() (*in module gnssrefl.download_orbits*), 64
download_prn_gps() (*in module gnssrefl.prn2gps*), 159
download_psmsl() (*in module gnssrefl.download_psmsl*), 65
download_qld() (*in module gnssrefl.download_noaa*), 62
download_rinex() (*in module gnssrefl.download_rinex*), 65
download_teqc() (*in module gnssrefl.download_teqc*), 67
download_tides() (*in module gnssrefl.download_tides*), 68
download_unr() (*in module gnssrefl.download_unr*), 69
download_wsv() (*in module gnssrefl.download_wsv*), 69
doy2ymd() (*in module gnssrefl.gps*), 95
doy_hour_to_mjd() (*in module gnssrefl.tracks*), 226
dumb_high_low_tide() (*in module gnssrefl.fundy*), 75

E

e (*gnssrefl.gps.wgs84 attribute*), 121
EGM96geoid (*class in gnssrefl.EGM96*), 51
elev_angle() (*in module gnssrefl.gps*), 95
elev_limits() (*in module gnssrefl.snrfile_functions*), 209
Equivalent_Angle_Corr_mpf() (*in module gnssrefl.refraction*), 174
Equivalent_Angle_Corr_NITE() (*in module gnssrefl.refraction*), 174
esp_highrate() (*in module gnssrefl.highrate*), 128
extract_arcs() (*in module gnssrefl.extract_arcs*), 70

extract_arcs_from_file() (*in module gnssrefl.extract_arcs*), 71
extract_arcs_from_station() (*in module gnssrefl.extract_arcs*), 72
extract_arcs_from_tracks() (*in module gnssrefl.extract_arcs*), 73
extract_snr() (*in module gnssrefl.rinex2snr*), 183

F

f (*gnssrefl.gps.wgs84 attribute*), 121
fbias_daily_avg() (*in module gnssrefl.daily_avg*), 56
fdoy2mjd() (*in module gnssrefl.gps*), 95
FileManagement (*class in gnssrefl.utils*), 230
filename_plus() (*in module gnssrefl.karnak_libraries*), 133
FileTypes (*class in gnssrefl.utils*), 232
filter_by_ignored_ranges() (*in module gnssrefl.tracks_qc*), 229
final_gfz_orbits() (*in module gnssrefl.gps*), 96
find_active_epoch_containing() (*in module gnssrefl.tracks_qc*), 229
find_apriori_rh_file() (*gnssrefl.utils.FileManagement method*), 230
find_daily_avg_phase_file() (*gnssrefl.utils.FileManagement method*), 230
find_json_file() (*gnssrefl.utils.FileManagement method*), 231
find_mgnss_satlist() (*in module gnssrefl.gnssir_v2*), 84
find_ortho_height() (*in module gnssrefl.sd_libs*), 197
find_satlist_wdate() (*in module gnssrefl.gps*), 96
find_snr_file() (*gnssrefl.utils.FileManagement method*), 231
find_start_stop() (*in module gnssrefl.download_ioc*), 61
find_volumetric_water_content_file() (*gnssrefl.utils.FileManagement method*), 231
findConstell() (*in module gnssrefl.gps*), 96
fit_segment() (*in module gnssrefl.tracks*), 226
fix_angle_azimuth() (*in module gnssrefl.nmea2snr*), 141
flipit() (*in module gnssrefl.subdaily*), 217
flipit2() (*in module gnssrefl.subdaily*), 218
flipit3() (*in module gnssrefl.sd_libs*), 197
format_qc_summary() (*in module gnssrefl.utils*), 233
fract_to_obstimes() (*in module gnssrefl.subdaily*), 218
freq_out() (*in module gnssrefl.gps*), 97
freq_out() (*in module gnssrefl.spline_functions*), 211
FresnelZone() (*in module gnssrefl.refl_zones*), 169
fundyplt1() (*in module gnssrefl.sd_libs*), 198

G

ga_highrate() (*in module gnssrefl.gps*), 97
ga_stuff() (*in module gnssrefl.karnak_libraries*), 133
ga_stuff_highrate() (*in module gnssrefl.karnak_libraries*), 133
gbm_orbits_direct() (*in module gnssrefl.gps*), 97
generate_rolling_vwc_from_tracks() (*in module gnssrefl.vwc_hourly*), 237
geoidCorrection() (*in module gnssrefl.gps*), 97
get_bin_schedule_info() (*in module gnssrefl.phase_functions*), 148
get_cddis_navfile() (*in module gnssrefl.gps*), 98
get_constellation() (*in module gnssrefl.gnss_frequencies*), 75
get_coords() (*in module gnssrefl.make_meta*), 138
get_directory_path() (*gnssrefl.utils.FileManagement method*), 231
get_display_label() (*in module gnssrefl.gnss_frequencies*), 75
get_es_sdk_headers() (*in module gnssrefl.make_meta*), 138
get_esa_navfile() (*in module gnssrefl.gps*), 98
get_favourite_obs_dict() (*in module gnssrefl.rinpy*), 194
get_file_path() (*gnssrefl.utils.FileManagement method*), 232
get_file_suffix() (*in module gnssrefl.gnss_frequencies*), 75
get_files() (*in module gnssrefl.computemp1mp2*), 54
get_glonass_channel() (*in module gnssrefl.gnss_frequencies*), 76
get_glonass_wavelength() (*in module gnssrefl.gnss_frequencies*), 76
get_individual_tracks_path() (*gnssrefl.utils.FileManagement method*), 232
get_local_rinexfile() (*in module gnssrefl.rinex2snr*), 183
get_noaa_obstimes() (*in module gnssrefl.gps*), 98
get_noaa_obstimes_plus() (*in module gnssrefl.gps*), 98
get_obstimes() (*in module gnssrefl.gps*), 99
get_obstimes_plus() (*in module gnssrefl.gps*), 99
get_ofac_hifac() (*in module gnssrefl.gps*), 99
get_ofac_hifac() (*in module gnssrefl.spline_functions*), 211
get_orbits_setexe() (*in module gnssrefl.gps*), 99
get_sat_list() (*in module gnssrefl.gnss_frequencies*), 76
get_sat_range() (*in module gnssrefl.gnss_frequencies*), 76
get_scale_factor() (*in module gnssrefl.gnss_frequencies*), 76
get_signal_label() (*in module gnssrefl.gnss_frequencies*), 76
get_snr_column() (*in module gnssrefl.gnss_frequencies*), 76

get_sopac_navfile() (*in module gnssrefl.gps*), 100
get_sopac_navfile_cron() (*in module gnssrefl.gps*), 100
get_sys() (*in module gnssrefl.utils*), 233
get_temporal_suffix() (*in module gnssrefl.phase_functions*), 148
get_tracks_file_path() (*gnssrefl.utils.FileManagement method*), 232
get_vwc_frequency() (*in module gnssrefl.phase_functions*), 148
get_vwc_tracks_file_path() (*gnssrefl.utils.FileManagement method*), 232
get_vwc_tracks_freqs() (*in module gnssrefl.vwc_cl*), 234
get_wavelength() (*in module gnssrefl.gnss_frequencies*), 76
get_wuhan_orbits() (*in module gnssrefl.gps*), 100
getMJD() (*in module gnssrefl.gps*), 97
getnavfile() (*in module gnssrefl.gps*), 101
getnavfile_archive() (*in module gnssrefl.gps*), 101
getrinexversion() (*in module gnssrefl.rinpy*), 194
getseries() (*in module gnssrefl.gps*), 101
getsp3file() (*in module gnssrefl.gps*), 101
getsp3file_flex() (*in module gnssrefl.gps*), 102
getsp3file_mgex() (*in module gnssrefl.gps*), 102
gfz_version() (*in module gnssrefl.gps*), 102
gmf_deriv() (*in module gnssrefl.refraction*), 178
gnssir() (*in module gnssrefl.gnssir_cl*), 77
gnssir_failqc_result (*gnssrefl.utils.FileTypes attribute*), 232
gnssir_guts_v2() (*in module gnssrefl.gnssir_v2*), 84
gnssir_result (*gnssrefl.utils.FileTypes attribute*), 232
gnssrefl
    module, 51
gnssrefl.advanced_vegetation_correction
    module, 51
gnssrefl.check_rinex_file
    module, 54
gnssrefl.computemp1mp2
    module, 54
gnssrefl.daily_avg
    module, 56
gnssrefl.daily_avg_cl
    module, 58
gnssrefl.download_ioc
    module, 61
gnssrefl.download_noaa
    module, 61
gnssrefl.download_orbits
    module, 64
gnssrefl.download_psmsl
    module, 65
gnssrefl.download_rinex
    module, 65
gnssrefl.download_teqc
    module, 67
gnssrefl.download_tides
    module, 68
gnssrefl.download_unr
    module, 69
gnssrefl.download_wsv
    module, 69
gnssrefl.EGM96
    module, 51
gnssrefl.extract_arcs
    module, 69
gnssrefl.filesizes
    module, 74
gnssrefl.fundy
    module, 75
gnssrefl.gnss_frequencies
    module, 75
gnssrefl.gnssir_cl
    module, 77
gnssrefl.gnssir_input
    module, 80
gnssrefl.gnssir_v2
    module, 83
gnssrefl.gps
    module, 89
gnssrefl.gpsweek
    module, 126
gnssrefl.highrate
    module, 126
gnssrefl.installexe_cl
    module, 129
gnssrefl.invsnr_cl
    module, 130
gnssrefl.invsnr_input
    module, 132
gnssrefl.karnak_libraries
    module, 133
gnssrefl.kelly
    module, 137
gnssrefl.llh2xyz
    module, 137
gnssrefl.make_meta
    module, 137
gnssrefl.max_resolve_RH_cl
    module, 139
gnssrefl.mjd
    module, 140
gnssrefl.nmea2snr
    module, 140
gnssrefl.nmea2snr_cl
    module, 144
gnssrefl.nyquist_libs
    module, 146

gnssrefl.phase_functions
    module, 146
gnssrefl.pickle_dilemma
    module, 159
gnssrefl.prn2gps
    module, 159
gnssrefl.query_unr
    module, 159
gnssrefl.quicklib
    module, 165
gnssrefl.quickLook_cl
    module, 159
gnssrefl.quickLook_function2
    module, 161
gnssrefl.quickPhase
    module, 163
gnssrefl.quickplt
    module, 165
gnssrefl.read_snr_files
    module, 168
gnssrefl.refl_zones
    module, 169
gnssrefl.refl_zones_cl
    module, 173
gnssrefl.refraction
    module, 174
gnssrefl.retrieve_rh
    module, 181
gnssrefl.rh_plot
    module, 182
gnssrefl.rinex2snr
    module, 182
gnssrefl.rinex2snr_cl
    module, 188
gnssrefl.rinex3_rinex2
    module, 193
gnssrefl.rinex3_snr
    module, 193
gnssrefl.rinex_coords
    module, 193
gnssrefl.rinpy
    module, 194
gnssrefl.rt_rinex3_snr
    module, 196
gnssrefl.sd_libs
    module, 197
gnssrefl.simple_vegetation_correction
    module, 204
gnssrefl.smoosh
    module, 205
gnssrefl.smoosh_snr
    module, 205
gnssrefl.snow_functions
    module, 206
gnssrefl.snowdepth_cl
    module, 208
gnssrefl.snrfile_functions
    module, 209
gnssrefl.spline_functions
    module, 210
gnssrefl.subdaily
    module, 217
gnssrefl.subdaily_cl
    module, 221
gnssrefl.tracks
    module, 224
gnssrefl.tracks_cl
    module, 227
gnssrefl.tracks_qc
    module, 228
gnssrefl.utils
    module, 230
gnssrefl.veg_multiyr
    module, 234
gnssrefl.vwc_cl
    module, 234
gnssrefl.vwc_hourly
    module, 237
gnssrefl.vwc_input
    module, 239
gnssrefl.xyz2llh
    module, 241
gnssrefl.ydoy
    module, 242
gnssrefl.ymd
    module, 242
go_from_crxgz_to_rnx() (*in module gnssrefl.rinex2snr*), 183
gogetit() (*in module gnssrefl.karnak_libraries*), 133
goodbad() (*in module gnssrefl.quickLook_function2*), 161
gps2datenum() (*in module gnssrefl.spline_functions*), 211
gps2datetime() (*in module gnssrefl.spline_functions*), 211
gps_default_frequencies() (*in module gnssrefl.gnss_frequencies*), 76
gpt2_1w() (*in module gnssrefl.refraction*), 178
greenland_rinex3() (*in module gnssrefl.gps*), 102
gsi_data() (*in module gnssrefl.karnak_libraries*), 134
gzip_migration() (*in module gnssrefl.gnssir_v2*), 85

H

hatanaka_version() (*in module gnssrefl.gps*), 103
hatanaka_warning() (*in module gnssrefl.gps*), 103
height() (*gnssrefl.EGM96.EGM96geoid method*), 51
help_debug() (*in module gnssrefl.phase_functions*), 148
highrate_nz() (*in module gnssrefl.gps*), 103

Hv_Hr_ratio() (*in module gnssrefl.refraction*), 175

I

ign_orbits() (*in module gnssrefl.gps*), 103
ign_rinex3() (*in module gnssrefl.gps*), 103
ignore_range() (*in module gnssrefl.tracks_qc*), 229
igsname() (*in module gnssrefl.gps*), 104
in_winter() (*in module gnssrefl.veg_multiyr*), 234
individual_tracks (*gnssrefl.utils.FileTypes attribute*), 232
inout() (*in module gnssrefl.gps*), 104
installexe() (*in module gnssrefl.installexe_cl*), 129
invsnr() (*in module gnssrefl.invsnr_cl*), 130
invsnr_header() (*in module gnssrefl.spline_functions*), 212
invsnr_input() (*in module gnssrefl.invsnr_input*), 132
is_valid_frequency() (*in module gnssrefl.gnss_frequencies*), 76
iso_to_mjd() (*in module gnssrefl.tracks*), 226
iter_vwc_epochs() (*in module gnssrefl.vwc_cl*), 234

J

just_bkg() (*in module gnssrefl.karnak_libraries*), 134

K

kadaster_highrate() (*in module gnssrefl.highrate*), 128
kgpsweek() (*in module gnssrefl.gps*), 104
kgpsweekC() (*in module gnssrefl.gps*), 105
kinda_qc() (*in module gnssrefl.phase_functions*), 149
kristine_dictionary() (*in module gnssrefl.spline_functions*), 212

L

l1c_list() (*in module gnssrefl.gps*), 105
l2c_l5_list() (*in module gnssrefl.gps*), 105
llh2xyz() (*in module gnssrefl.gps*), 105
load_arcs() (*in module gnssrefl.tracks*), 226
load_avg_phase() (*in module gnssrefl.phase_functions*), 149
load_clara_model() (*in module gnssrefl.advanced_vegetation_correction*), 52
load_gnssir_results_from_tracks() (*in module gnssrefl.extract_arcs*), 73
load_phase() (*in module gnssrefl.phase_functions*), 149
load_results_with_failqc() (*in module gnssrefl.extract_arcs*), 73
load_sat_phase() (*in module gnssrefl.phase_functions*), 150
load_snr_time_filtered() (*in module gnssrefl.read_snr_files*), 168
load_tracks_json() (*in module gnssrefl.tracks*), 226
load_vwc_tracks() (*in module gnssrefl.vwc_cl*), 234
loadrinexfromnpz() (*in module gnssrefl.rinpy*), 194
loadsnrfile() (*in module gnssrefl.spline_functions*), 212
local_update_plot() (*in module gnssrefl.gnssir_v2*), 85
look_for_pickle_file() (*in module gnssrefl.refraction*), 179
lookup_arc() (*in module gnssrefl.tracks*), 226
low_pct() (*in module gnssrefl.phase_functions*), 150
lsp_header() (*in module gnssrefl.gps*), 106
LSPresult_name() (*in module gnssrefl.gps*), 89

M

main() (*in module gnssrefl.check_rinex_file*), 54
main() (*in module gnssrefl.computemp1mp2*), 55
main() (*in module gnssrefl.daily_avg_cl*), 60
main() (*in module gnssrefl.download_orbits*), 65
main() (*in module gnssrefl.download_rinex*), 67
main() (*in module gnssrefl.download_teqc*), 67
main() (*in module gnssrefl.download_tides*), 68
main() (*in module gnssrefl.download_unr*), 69
main() (*in module gnssrefl.download_wsv*), 69
main() (*in module gnssrefl.filesizes*), 74
main() (*in module gnssrefl.gnssir_cl*), 79
main() (*in module gnssrefl.gnssir_input*), 80
main() (*in module gnssrefl.gpsweek*), 126
main() (*in module gnssrefl.installexe_cl*), 129
main() (*in module gnssrefl.invsnr_cl*), 132
main() (*in module gnssrefl.invsnr_input*), 132
main() (*in module gnssrefl.llh2xyz*), 137
main() (*in module gnssrefl.make_meta*), 138
main() (*in module gnssrefl.max_resolve_RH_cl*), 139
main() (*in module gnssrefl.mjd*), 140
main() (*in module gnssrefl.nmea2snr_cl*), 144
main() (*in module gnssrefl.pickle_dilemma*), 159
main() (*in module gnssrefl.prn2gps*), 159
main() (*in module gnssrefl.query_unr*), 159
main() (*in module gnssrefl.quickLook_cl*), 159
main() (*in module gnssrefl.quickPhase*), 163
main() (*in module gnssrefl.quickplt*), 165
main() (*in module gnssrefl.refl_zones_cl*), 173
main() (*in module gnssrefl.rh_plot*), 182
main() (*in module gnssrefl.rinex2snr_cl*), 188
main() (*in module gnssrefl.rinex3_rinex2*), 193
main() (*in module gnssrefl.rinex3_snr*), 193
main() (*in module gnssrefl.rinex_coords*), 193
main() (*in module gnssrefl.rt_rinex3_snr*), 196
main() (*in module gnssrefl.smoosh*), 205
main() (*in module gnssrefl.smoosh_snr*), 205
main() (*in module gnssrefl.snowdepth_cl*), 208
main() (*in module gnssrefl.subdaily_cl*), 221
main() (*in module gnssrefl.tracks_cl*), 227
main() (*in module gnssrefl.veg_multiyr*), 234
main() (*in module gnssrefl.vwc_cl*), 234

main() (*in module gnssrefl.vwc_hourly*), 238
main() (*in module gnssrefl.vwc_input*), 240
main() (*in module gnssrefl.xyz2llh*), 241
main() (*in module gnssrefl.ydoy*), 242
main() (*in module gnssrefl.ymd*), 242
main_hourly() (*in module gnssrefl.vwc_hourly*), 238
makan_warning() (*in module gnssrefl.rinex2snr*), 183
make_azim_choices() (*in module gnssrefl.gps*), 106
make_FZ_kml() (*in module gnssrefl.refl_zones*), 170
make_gnssir_input() (*in module gnssrefl.gnssir_input*), 80
make_json (*gnssrefl.utils.FileTypes attribute*), 233
make_meta() (*in module gnssrefl.make_meta*), 138
make_nav_dirs() (*in module gnssrefl.gps*), 106
make_parallel_proc_lists() (*in module gnssrefl.gnssir_v2*), 86
make_parallel_proc_lists_mjd() (*in module gnssrefl.gnssir_v2*), 86
make_rinex2_ofiles() (*in module gnssrefl.karnak_libraries*), 134
make_snow_filter() (*in module gnssrefl.phase_functions*), 150
make_snrdir() (*in module gnssrefl.gps*), 106
make_wavelength_column() (*in module gnssrefl.spline_functions*), 212
makeEllipse_latlon() (*in module gnssrefl.refl_zones*), 170
makeFresnelEllipse() (*in module gnssrefl.refl_zones*), 170
max_resolve_RH() (*in module gnssrefl.max_resolve_RH_cl*), 139
merge_epochs() (*in module gnssrefl.tracks_qc*), 229
mergerinexfiles() (*in module gnssrefl.rinpy*), 194
meta_man_input() (*in module gnssrefl.make_meta*), 139
mirror_plot() (*in module gnssrefl.sd_libs*), 198
mjd() (*in module gnssrefl.gps*), 106
mjd_more() (*in module gnssrefl.gps*), 107
mjd_to_date() (*in module gnssrefl.gps*), 107
mjd_to_datetime() (*in module gnssrefl.gps*), 107
mjd_to_file_columns() (*in module gnssrefl.phase_functions*), 151
mjd_to_iso_ceil() (*in module gnssrefl.tracks*), 227
mjd_to_iso_floor() (*in module gnssrefl.tracks*), 227
mjd_to_obstimes() (*in module gnssrefl.sd_libs*), 198
modjul_to_ydoy() (*in module gnssrefl.gps*), 107
module
    gnssrefl, 51
    gnssrefl.advanced_vegetation_correction, 51
    gnssrefl.check_rinex_file, 54
    gnssrefl.computemp1mp2, 54
    gnssrefl.daily_avg, 56
    gnssrefl.daily_avg_cl, 58
    gnssrefl.download_ioc, 61
    gnssrefl.download_noaa, 61
    gnssrefl.download_orbits, 64
    gnssrefl.download_psmsl, 65
    gnssrefl.download_rinex, 65
    gnssrefl.download_teqc, 67
    gnssrefl.download_tides, 68
    gnssrefl.download_unr, 69
    gnssrefl.download_wsv, 69
    gnssrefl.EGM96, 51
    gnssrefl.extract_arcs, 69
    gnssrefl.filesizes, 74
    gnssrefl.fundy, 75
    gnssrefl.gnss_frequencies, 75
    gnssrefl.gnssir_cl, 77
    gnssrefl.gnssir_input, 80
    gnssrefl.gnssir_v2, 83
    gnssrefl.gps, 89
    gnssrefl.gpsweek, 126
    gnssrefl.highrate, 126
    gnssrefl.installexe_cl, 129
    gnssrefl.invsnr_cl, 130
    gnssrefl.invsnr_input, 132
    gnssrefl.karnak_libraries, 133
    gnssrefl.kelly, 137
    gnssrefl.llh2xyz, 137
    gnssrefl.make_meta, 137
    gnssrefl.max_resolve_RH_cl, 139
    gnssrefl.mjd, 140
    gnssrefl.nmea2snr, 140
    gnssrefl.nmea2snr_cl, 144
    gnssrefl.nyquist_libs, 146
    gnssrefl.phase_functions, 146
    gnssrefl.pickle_dilemma, 159
    gnssrefl.prn2gps, 159
    gnssrefl.query_unr, 159
    gnssrefl.quicklib, 165
    gnssrefl.quickLook_cl, 159
    gnssrefl.quickLook_function2, 161
    gnssrefl.quickPhase, 163
    gnssrefl.quickplt, 165
    gnssrefl.read_snr_files, 168
    gnssrefl.refl_zones, 169
    gnssrefl.refl_zones_cl, 173
    gnssrefl.refraction, 174
    gnssrefl.retrieve_rh, 181
    gnssrefl.rh_plot, 182
    gnssrefl.rinex2snr, 182
    gnssrefl.rinex2snr_cl, 188
    gnssrefl.rinex3_rinex2, 193
    gnssrefl.rinex3_snr, 193
    gnssrefl.rinex_coords, 193
    gnssrefl.rinpy, 194
    gnssrefl.rt_rinex3_snr, 196

gnssrefl.sd_libs, 197
gnssrefl.simple_vegetation_correction, 204
gnssrefl.smoosh, 205
gnssrefl.smoosh_snr, 205
gnssrefl.snow_functions, 206
gnssrefl.snowdepth_cl, 208
gnssrefl.snrfile_functions, 209
gnssrefl.spline_functions, 210
gnssrefl.subdaily, 217
gnssrefl.subdaily_cl, 221
gnssrefl.tracks, 224
gnssrefl.tracks_cl, 227
gnssrefl.tracks_qc, 228
gnssrefl.utils, 230
gnssrefl.veg_multiyr, 234
gnssrefl.vwc_cl, 234
gnssrefl.vwc_hourly, 237
gnssrefl.vwc_input, 239
gnssrefl.xyz2llh, 241
gnssrefl.ydoy, 242
gnssrefl.ymd, 242
month_converter() (*in module gnssrefl.gps*), 108
more_confused_obstimes() (*in module gnssrefl.gps*), 108
move_arc_to_failqc() (*in module gnssrefl.extract_arcs*), 74
mpf_tot() (*in module gnssrefl.refraction*), 179
mpfile_unavco() (*in module gnssrefl.download_teqc*), 67
mu (*gnssrefl.gps.constants attribute*), 94
mu (*gnssrefl.snrfile_functions.constants attribute*), 209
multi_freq_amp() (*in module gnssrefl.daily_avg*), 56
multimonthdownload() (*in module gnssrefl.download_noaa*), 62
my_percentile() (*in module gnssrefl.subdaily*), 218
myfavoritegpsobs() (*in module gnssrefl.gps*), 108
myfavoriteobs() (*in module gnssrefl.gps*), 108
myfindephem() (*in module gnssrefl.gps*), 108
myreadnav() (*in module gnssrefl.gps*), 108
myscan() (*in module gnssrefl.gps*), 108

N

N_layer() (*in module gnssrefl.refraction*), 175
nav_name() (*in module gnssrefl.gps*), 108
navfile_retrieve() (*in module gnssrefl.gps*), 109
new_rinex3_rinex2() (*in module gnssrefl.gps*), 109
new_rise_set() (*in module gnssrefl.gnssir_v2*), 86
new_ultra_gfz_orbits() (*in module gnssrefl.gps*), 109
newchip_gfzrnx() (*in module gnssrefl.installexe_cl*), 130
newchip_hatanaka() (*in module gnssrefl.installexe_cl*), 130
newish_gfz_orbits() (*in module gnssrefl.gps*), 110
newvegplot() (*in module gnssrefl.veg_multiyr*), 234
nextdoy() (*in module gnssrefl.gps*), 110
nicerTime() (*in module gnssrefl.gps*), 110
nmea2snr() (*in module gnssrefl.nmea2snr_cl*), 144
nmea_apriori_coords() (*in module gnssrefl.nmea2snr*), 141
nmea_sp3_azel() (*in module gnssrefl.nmea2snr*), 141
nmea_translate() (*in module gnssrefl.nmea2snr*), 142
noaa2me() (*in module gnssrefl.download_noaa*), 62
noaa2me() (*in module gnssrefl.gps*), 110
noaa_command() (*in module gnssrefl.download_noaa*), 62
noaatime_to_obstime() (*in module gnssrefl.gps*), 111
norm() (*in module gnssrefl.gps*), 111
norm_zero_vxyz() (*in module gnssrefl.advanced_vegetation_correction*), 53
normAmp() (*in module gnssrefl.phase_functions*), 151
numsats_plot() (*in module gnssrefl.sd_libs*), 198
ny_plot() (*in module gnssrefl.nyquist_libs*), 146
nyquist_simple() (*in module gnssrefl.refl_zones*), 171

O

old_quad() (*in module gnssrefl.phase_functions*), 151
omegaEarth (*gnssrefl.gps.constants attribute*), 94
omegaEarth (*gnssrefl.snrfile_functions.constants attribute*), 209
one_gfz_archive_to_rule_them_all() (*in module gnssrefl.gps*), 111
open_gnssir_logfile() (*in module gnssrefl.gnssir_v2*), 86
open_outputfile() (*in module gnssrefl.gps*), 111
open_plot() (*in module gnssrefl.gps*), 112
orbfile_cddis() (*in module gnssrefl.gps*), 112
output_names() (*in module gnssrefl.subdaily*), 218

P

padClara() (*in module gnssrefl.advanced_vegetation_correction*), 53
parse_arguments() (*in module gnssrefl.daily_avg_cl*), 60
parse_arguments() (*in module gnssrefl.download_orbits*), 65
parse_arguments() (*in module gnssrefl.download_rinex*), 67
parse_arguments() (*in module gnssrefl.download_teqc*), 67
parse_arguments() (*in module gnssrefl.download_tides*), 68
parse_arguments() (*in module gnssrefl.download_unr*), 69
parse_arguments() (*in module gnssrefl.download_wsv*), 69
parse_arguments() (*in module gnssrefl.gnssir_cl*), 79

parse_arguments() (*in module gnssrefl.gnssir_input*), 83
parse_arguments() (*in module gnssrefl.installexe_cl*), 130
parse_arguments() (*in module gnssrefl.invsnr_cl*), 132
parse_arguments() (*in module gnssrefl.invsnr_input*), 132
parse_arguments() (*in module gnssrefl.make_meta*), 139
parse_arguments() (*in module gnssrefl.max_resolve_RH_cl*), 140
parse_arguments() (*in module gnssrefl.nmea2snr_cl*), 145
parse_arguments() (*in module gnssrefl.quickLook_cl*), 159
parse_arguments() (*in module gnssrefl.quickPhase*), 163
parse_arguments() (*in module gnssrefl.quickplt*), 165
parse_arguments() (*in module gnssrefl.refl_zones_cl*), 173
parse_arguments() (*in module gnssrefl.rh_plot*), 182
parse_arguments() (*in module gnssrefl.rinex2snr_cl*), 188
parse_arguments() (*in module gnssrefl.snowdepth_cl*), 208
parse_arguments() (*in module gnssrefl.subdaily_cl*), 221
parse_arguments() (*in module gnssrefl.tracks_cl*), 227
parse_arguments() (*in module gnssrefl.vwc_cl*), 234
parse_arguments() (*in module gnssrefl.vwc_input*), 240
parse_arguments_hourly() (*in module gnssrefl.vwc_hourly*), 238
phase_file (*gnssrefl.utils.FileTypes attribute*), 233
phase_file_fmt() (*in module gnssrefl.phase_functions*), 151
phase_file_header() (*in module gnssrefl.phase_functions*), 151
phase_tracks() (*in module gnssrefl.phase_functions*), 151
pickup_files_nyquist() (*in module gnssrefl.nyquist_libs*), 146
pickup_from_noaa() (*in module gnssrefl.download_noaa*), 63
pickup_subdaily_json_defaults() (*in module gnssrefl.sd_libs*), 199
plot2screen() (*in module gnssrefl.gnssir_v2*), 87
plot_baseline_leveling() (*in module gnssrefl.phase_functions*), 152
plot_hourly_vs_daily_vwc() (*in module gnssrefl.vwc_hourly*), 238
plot_tracks() (*in module gnssrefl.spline_functions*), 212
pre_check_arc() (*in module gnssrefl.utils*), 233
prepare_track_dir() (*in module gnssrefl.phase_functions*), 152
prevdoy() (*in module gnssrefl.gps*), 112
print_archives() (*in module gnssrefl.rinex2snr*), 183
print_badpoints() (*in module gnssrefl.sd_libs*), 199
print_file_stats() (*in module gnssrefl.gps*), 112
print_version_to_screen() (*in module gnssrefl.gps*), 112
process_day_worker() (*in module gnssrefl.gnssir_cl*), 79
process_jobs_multi() (*in module gnssrefl.nmea2snr_cl*), 145
process_jobs_nopar() (*in module gnssrefl.nmea2snr_cl*), 145
process_phase_day() (*in module gnssrefl.quickPhase*), 163
process_phase_day_worker() (*in module gnssrefl.quickPhase*), 163
process_phase_sequential() (*in module gnssrefl.quickPhase*), 163
process_vwc_from_tracks() (*in module gnssrefl.vwc_cl*), 234
process_year() (*in module gnssrefl.gnssir_cl*), 79
processrinexfile() (*in module gnssrefl.rinpy*), 195
propagate() (*in module gnssrefl.gps*), 112
propagate_and_azel_sp3() (*in module gnssrefl.snrfile_functions*), 210

Q

query_coordinate_file() (*in module gnssrefl.gps*), 113
queryUNR_modern() (*in module gnssrefl.gps*), 112
quick_plot() (*in module gnssrefl.gps*), 113
quick_raw() (*in module gnssrefl.daily_avg*), 56
quick_refraction() (*in module gnssrefl.quickLook_function2*), 162
quickazel() (*in module gnssrefl.gps*), 113
quicklook() (*in module gnssrefl.quickLook_cl*), 159
quickLook_function() (*in module gnssrefl.quickLook_function2*), 162
quickname() (*in module gnssrefl.nmea2snr*), 142
quickname() (*in module gnssrefl.rinex2snr*), 183
quickp() (*in module gnssrefl.gps*), 113
quickphase() (*in module gnssrefl.quickPhase*), 163
quickTr() (*in module gnssrefl.sd_libs*), 199

R

random() (*in module gnssrefl.gps*), 113
randomfilename() (*in module gnssrefl.gps*), 113
rapid_gfz_orbits() (*in module gnssrefl.gps*), 113
raw_phase_file (*gnssrefl.utils.FileTypes attribute*), 233
read_4by5() (*in module gnssrefl.refraction*), 179
read_apriori_rh() (*in module gnssrefl.phase_functions*), 152

read_file() (*gnssrefl.utils.FileManagement method*), 232
read_files() (*in module gnssrefl.gps*), 114
read_files_in_dir() (*in module gnssrefl.utils*), 233
read_jpl_file() (*in module gnssrefl.prn2gps*), 159
read_json_file() (*in module gnssrefl.gnssir_v2*), 87
read_leapsecond_file() (*in module gnssrefl.gps*), 114
read_nmea() (*in module gnssrefl.nmea2snr*), 142
read_predicts() (*in module gnssrefl.fundy*), 75
read_simon_williams() (*in module gnssrefl.gps*), 114
read_snr() (*in module gnssrefl.read_snr_files*), 168
read_sp3() (*in module gnssrefl.gps*), 114
read_sp3file() (*in module gnssrefl.gps*), 114
read_the_orbits() (*in module gnssrefl.nyquist_libs*), 146
readheader() (*in module gnssrefl.rinpy*), 196
readin_and_plot() (*in module gnssrefl.subdaily*), 219
readin_plot_daily() (*in module gnssrefl.daily_avg*), 56
readklsnrtxt() (*in module gnssrefl.spline_functions*), 213
readoutmp() (*in module gnssrefl.computemp1mp2*), 55
ReadRecAnt() (*in module gnssrefl.computemp1mp2*), 54
readSNRval() (*in module gnssrefl.rinex2snr*), 184
readWrite_gpt2_1w() (*in module gnssrefl.refraction*), 179
recompute_derived_fields() (*in module gnssrefl.tracks_qc*), 229
recompute_durations() (*in module gnssrefl.tracks_qc*), 229
recompute_metadata_aggregates() (*in module gnssrefl.tracks_qc*), 229
recompute_n_arcs() (*in module gnssrefl.tracks_qc*), 229
refit_active_epochs() (*in module gnssrefl.tracks_qc*), 229
reflzones() (*in module gnssrefl.refl_zones_cl*), 173
refrc_Rueger() (*in module gnssrefl.refraction*), 180
remove_dc_component() (*in module gnssrefl.extract_arcs*), 74
removeDC() (*in module gnssrefl.gps*), 115
rename_vals() (*in module gnssrefl.phase_functions*), 153
renumber_epoch_ids() (*in module gnssrefl.tracks_qc*), 229
reorder_epoch_keys() (*in module gnssrefl.tracks_qc*), 229
replace_wget() (*in module gnssrefl.gps*), 115
require_active_epoch() (*in module gnssrefl.tracks_qc*), 229
residuals_cubspl_js() (*in module gnssrefl.spline_functions*), 213
residuals_cubspl_spectral() (*in module gnssrefl.spline_functions*), 213
result_directories() (*in module gnssrefl.gps*), 116
results_dir_has_files() (*in module gnssrefl.tracks*), 227
retrieve_Hdates() (*in module gnssrefl.gnssir_v2*), 87
retrieve_rh() (*in module gnssrefl.retrieve_rh*), 181
rewrite_azel() (*in module gnssrefl.gnssir_v2*), 87
RH_ortho_plot2() (*in module gnssrefl.sd_libs*), 197
rh_plot() (*in module gnssrefl.rh_plot*), 182
rh_plots() (*in module gnssrefl.sd_libs*), 200
rhdot_correction2() (*in module gnssrefl.subdaily*), 220
rhdot_plots() (*in module gnssrefl.sd_libs*), 200
rinex2_highrate() (*in module gnssrefl.karnak_libraries*), 134
rinex2names() (*in module gnssrefl.karnak_libraries*), 135
rinex2snr() (*in module gnssrefl.rinex2snr_cl*), 188
rinex2snr_day_worker() (*in module gnssrefl.rinex2snr_cl*), 192
rinex3_nav() (*in module gnssrefl.gps*), 116
rinex_ga_highrate() (*in module gnssrefl.gps*), 116
rinex_jp() (*in module gnssrefl.gps*), 116
rinex_name() (*in module gnssrefl.gps*), 116
rinex_nrcan_highrate() (*in module gnssrefl.gps*), 116
rinex_unavco() (*in module gnssrefl.gps*), 117
rinex_unavco_highrate() (*in module gnssrefl.gps*), 117
RinexError, 194
rising_setting_new() (*in module gnssrefl.refl_zones*), 171
rnx2snr() (*in module gnssrefl.rinex2snr*), 184
rnx2snr_v3() (*in module gnssrefl.rinex2snr*), 184
rolling_window() (*in module gnssrefl.advanced_vegetation_correction*), 53
rot3() (*in module gnssrefl.gps*), 117
run_nmea2snr() (*in module gnssrefl.nmea2snr*), 143
run_quickplt() (*in module gnssrefl.quickplt*), 165
run_rinex2snr() (*in module gnssrefl.rinex2snr*), 184
run_teqc() (*in module gnssrefl.computemp1mp2*), 55

S

saastam2() (*in module gnssrefl.refraction*), 180
satfreq2waveL() (*in module gnssrefl.spline_functions*), 213
satorb() (*in module gnssrefl.rinex2snr*), 185
satorb_prop() (*in module gnssrefl.rinex2snr*), 185
satorb_prop_sp3() (*in module gnssrefl.rinex2snr*), 186
save_arc() (*in module gnssrefl.extract_arcs*), 74
save_individual_track_data() (*in module gnssrefl.simple_vegetation_correction*), 204

save_lsp_results() (*in module gnssrefl.spline_functions*), 213
save_plot() (*in module gnssrefl.gps*), 117
save_plot() (*in module gnssrefl.quicklib*), 165
save_reflzone_orbits() (*in module gnssrefl.refl_zones*), 171
save_tracks() (*in module gnssrefl.tracks_qc*), 229
save_vwc_plot() (*in module gnssrefl.phase_functions*), 154
saverinextonpz() (*in module gnssrefl.rinpy*), 196
separateobservables() (*in module gnssrefl.rinpy*), 196
serial_cddis_files() (*in module gnssrefl.karnak_libraries*), 135
set_azlist_multi_regions() (*in module gnssrefl.refl_zones*), 171
set_environment() (*in module gnssrefl.utils*), 233
set_final_azlist() (*in module gnssrefl.refl_zones*), 172
set_labels() (*in module gnssrefl.quickLook_function2*), 163
set_parameters() (*in module gnssrefl.phase_functions*), 154
set_refraction_model() (*in module gnssrefl.spline_functions*), 214
set_rinex2snr_logs() (*in module gnssrefl.rinex2snr*), 186
set_subdir() (*in module gnssrefl.gps*), 117
set_system() (*in module gnssrefl.refl_zones*), 172
set_xlimits_ydoy() (*in module gnssrefl.quicklib*), 165
setup_arcs_directory() (*in module gnssrefl.extract_arcs*), 74
sfilename() (*in module gnssrefl.computemp1mp2*), 55
signal2list() (*in module gnssrefl.spline_functions*), 214
signal_label_to_freq() (*in module gnssrefl.gnss_frequencies*), 76
simple_vegetation_filter() (*in module gnssrefl.simple_vegetation_correction*), 204
simpleLSP() (*in module gnssrefl.spline_functions*), 214
simpleTime() (*in module gnssrefl.gps*), 117
sita_Earth() (*in module gnssrefl.refraction*), 180
sita_Satellite() (*in module gnssrefl.refraction*), 181
smarterWay() (*in module gnssrefl.spline_functions*), 215
snow_azimuthal() (*in module gnssrefl.snow_functions*), 206
snow_simple() (*in module gnssrefl.snow_functions*), 206
snowdepth() (*in module gnssrefl.snowdepth_cl*), 208
snowplot() (*in module gnssrefl.snow_functions*), 207
snr2arcs() (*in module gnssrefl.spline_functions*), 215
snr2spline() (*in module gnssrefl.spline_functions*), 216
snr_exist() (*in module gnssrefl.gps*), 118
snr_file (*gnssrefl.utils.FileTypes attribute*), 233
snr_file_size() (*in module gnssrefl.rinex2snr_cl*), 192
snr_name() (*in module gnssrefl.gps*), 118
sp3_name() (*in module gnssrefl.gps*), 118
spline_in_out() (*in module gnssrefl.subdaily*), 220
split_epoch() (*in module gnssrefl.tracks_qc*), 230
stack_two_more() (*in module gnssrefl.sd_libs*), 200
store_orbitfile() (*in module gnssrefl.gps*), 119
store_snrfile() (*in module gnssrefl.gps*), 119
str2bool() (*in module gnssrefl.utils*), 233
strip_compute() (*in module gnssrefl.gps*), 119
strip_rinexfile() (*in module gnssrefl.karnak_libraries*), 135
subdaily() (*in module gnssrefl.subdaily_cl*), 221
subdaily_phase_plot() (*in module gnssrefl.phase_functions*), 155
subdaily_resids_last_stage() (*in module gnssrefl.sd_libs*), 201
swapRS() (*in module gnssrefl.karnak_libraries*), 135

T

teqc_version() (*in module gnssrefl.gps*), 120
test_func() (*in module gnssrefl.phase_functions*), 155
test_func_new() (*in module gnssrefl.phase_functions*), 155
testing_nvals() (*in module gnssrefl.sd_libs*), 201
the_kelly_simple_way() (*in module gnssrefl.kelly*), 137
the_last_plot() (*in module gnssrefl.sd_libs*), 201
the_makan_option() (*in module gnssrefl.rinex2snr*), 186
time_limits() (*in module gnssrefl.snow_functions*), 207
tracks_file (*gnssrefl.utils.FileTypes attribute*), 233
trans_time() (*in module gnssrefl.quicklib*), 165
translate_dates() (*in module gnssrefl.gps*), 120
trignet() (*in module gnssrefl.gps*), 120
two_stacked_plots() (*in module gnssrefl.sd_libs*), 202

U

Ulich_Bending_Angle() (*in module gnssrefl.refraction*), 175
Ulich_Bending_Angle_original() (*in module gnssrefl.refraction*), 176
ultra_gfz_orbits() (*in module gnssrefl.gps*), 120
unignore_range() (*in module gnssrefl.tracks_qc*), 230
universal() (*in module gnssrefl.karnak_libraries*), 135
universal_all() (*in module gnssrefl.karnak_libraries*), 136
universal_rinex2() (*in module gnssrefl.karnak_libraries*), 136
unr_database() (*in module gnssrefl.gps*), 120

UNR_highrate() (*in module gnssrefl.gps*), 89
unused() (*in module gnssrefl.snow_functions*), 207
unwrap_az() (*in module gnssrefl.tracks*), 227
up() (*in module gnssrefl.gps*), 121
update_plot() (*in module gnssrefl.gps*), 121
update_quick_plot() (*in module gnssrefl.gps*), 121

V

validate_epoch_ids() (*in module gnssrefl.tracks_qc*), 230
validate_input_datatypes() (*in module gnssrefl.utils*), 233
variableArchives() (*in module gnssrefl.highrate*), 128
vary_Hortho() (*in module gnssrefl.sd_libs*), 202
vegoutfile() (*in module gnssrefl.veg_multiyr*), 234
vegplt() (*in module gnssrefl.computemp1mp2*), 55
volumetric_water_content (*gnssrefl.utils.FileTypes attribute*), 233
vwc() (*in module gnssrefl.vwc_cl*), 235
vwc_hourly() (*in module gnssrefl.vwc_hourly*), 238
vwc_input() (*in module gnssrefl.vwc_input*), 240
vwc_input_legacy() (*in module gnssrefl.vwc_input*), 241
vwc_legacy() (*in module gnssrefl.vwc_cl*), 236
vwc_outputs (*gnssrefl.utils.FileTypes attribute*), 233
vwc_plot() (*in module gnssrefl.phase_functions*), 155
vwc_tracks_file (*gnssrefl.utils.FileTypes attribute*), 233

W

warn_legacy_apriori_and_exit() (*in module gnssrefl.tracks*), 227
wgs84 (*class in gnssrefl.gps*), 121
what_is_today() (*in module gnssrefl.gps*), 121
whichquad() (*in module gnssrefl.quickLook_function2*), 163
window_data() (*in module gnssrefl.gps*), 122
window_new() (*in module gnssrefl.gnssir_v2*), 88
wl() (*in module gnssrefl.gnss_frequencies*), 76
write_all_phase() (*in module gnssrefl.phase_functions*), 156
write_apriori_rh() (*in module gnssrefl.phase_functions*), 156
write_avg_phase() (*in module gnssrefl.phase_functions*), 156
write_coords() (*in module gnssrefl.refl_zones*), 172
write_out_all() (*in module gnssrefl.daily_avg*), 57
write_out_data() (*in module gnssrefl.download_noaa*), 63
write_out_header() (*in module gnssrefl.subdaily*), 221
write_out_raw_phase() (*in module gnssrefl.phase_functions*), 157
write_out_RH_file() (*in module gnssrefl.daily_avg*), 57
write_phase_for_advanced() (*in module gnssrefl.phase_functions*), 157
write_QC_fails() (*in module gnssrefl.gps*), 123
write_rolling_vwc_output() (*in module gnssrefl.phase_functions*), 157
write_snr_from_nav() (*in module gnssrefl.rinex2snr*), 187
write_snr_from_sp3() (*in module gnssrefl.rinex2snr*), 187
write_spline_output() (*in module gnssrefl.sd_libs*), 203
write_subdaily() (*in module gnssrefl.subdaily*), 221
write_track_file() (*in module gnssrefl.phase_functions*), 158
write_tracks_json() (*in module gnssrefl.tracks*), 227
write_vwc_output() (*in module gnssrefl.phase_functions*), 158
writejsonfile() (*in module gnssrefl.sd_libs*), 203
writeout_azim() (*in module gnssrefl.snow_functions*), 207
writeout_snowdepth_v0() (*in module gnssrefl.snow_functions*), 208
writeout_spline_outliers() (*in module gnssrefl.sd_libs*), 203

X

xyz2llh() (*in module gnssrefl.gps*), 123
xyz2llhd() (*in module gnssrefl.gps*), 123

Y

ydoy2datetime() (*in module gnssrefl.gps*), 124
ydoy2mjd() (*in module gnssrefl.gps*), 124
ydoy2useful() (*in module gnssrefl.gps*), 124
ydoy2ymd() (*in module gnssrefl.gps*), 124
ydoych() (*in module gnssrefl.gps*), 124
ymd2ch() (*in module gnssrefl.gps*), 125
ymd2doy() (*in module gnssrefl.gps*), 125
ymd_hhmmss() (*in module gnssrefl.gps*), 125

Z

z_process_jobs() (*in module gnssrefl.rinex2snr_cl*), 192
zenithdelay() (*in module gnssrefl.gps*), 126