asilib

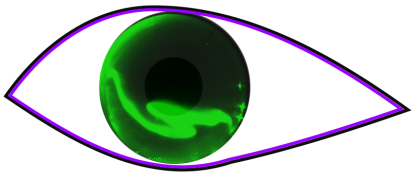
Mykhaylo Shumko

Apr 21, 2024

TABLE OF CONTENTS:

aurora-asi-lib

Aurora All-Sky Imager Library



Last Built: Apr 21, 2024 | **Version**: 0.23.0 | **Source**: github | **Archive**: zenodo.

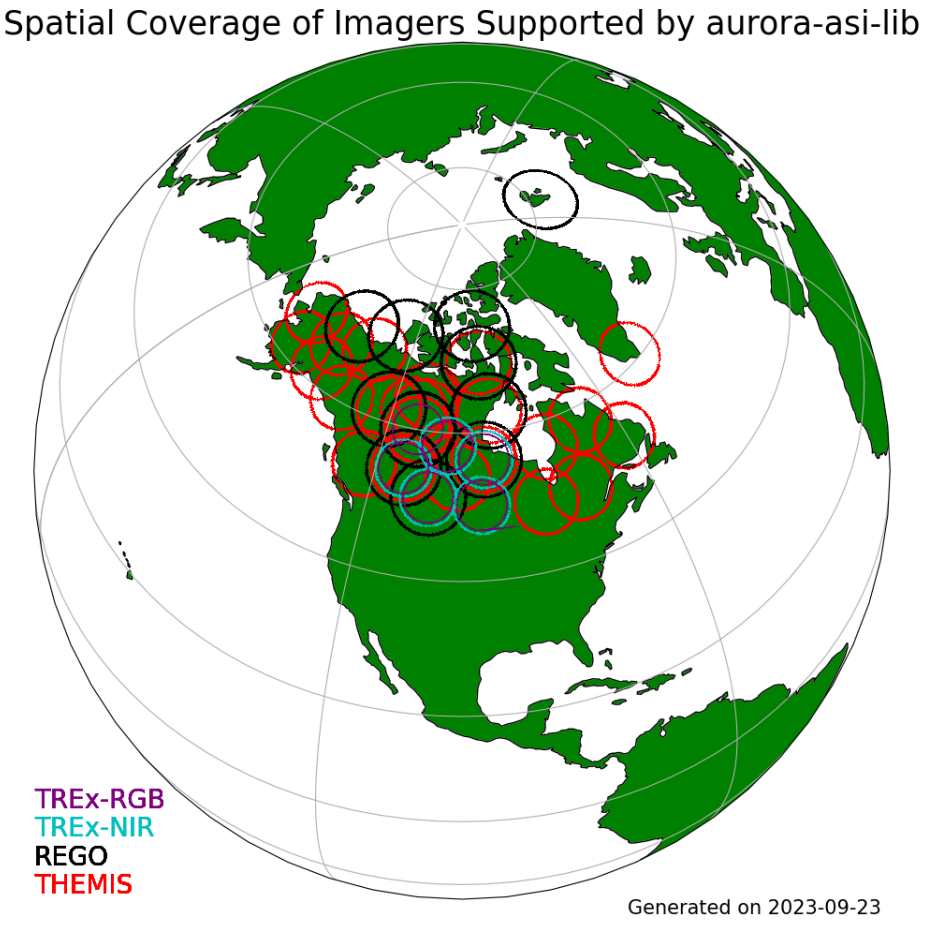
asilib is an open source package providing data access and analysis tools for the world's all-sky imager (ASI) data.

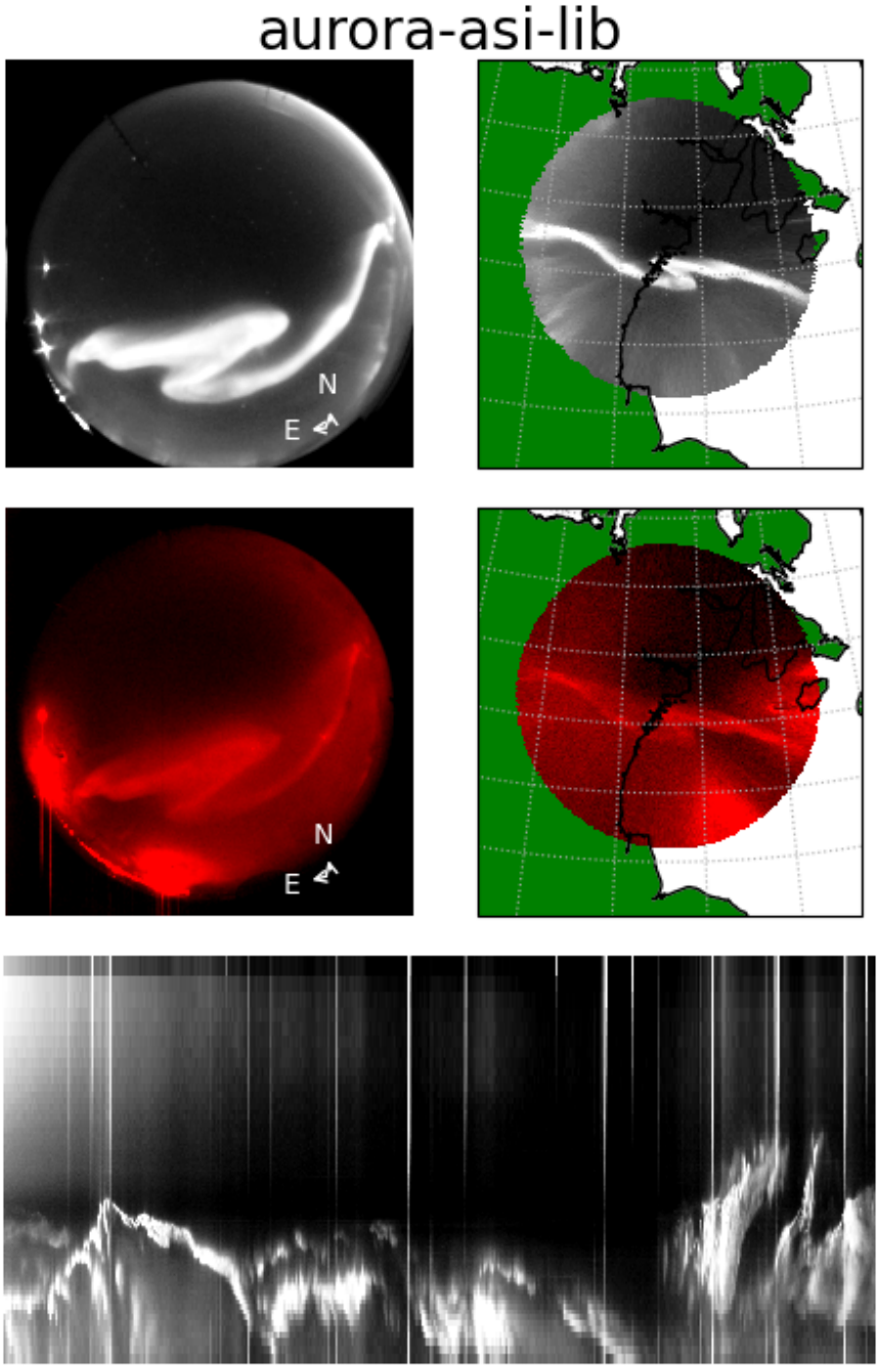
Note: The asilib code on PyPI moved from aurora-asi-lib to asilib. To update to version >=0.22.0, run

1. *python3 -m pip uninstall aurora-asi-lib* and
2. *python3 -m pip install asilib*.

The purpose of this project is to combine data from numerous observational ASI arrays into a single unified framework and is thus not associated with the development and operations of all sky cameras, or the curation of ASI datasets. All data is publicly available and is provided as-is. Please give appropriate credit and coordinate with instrument teams with regards to data issues and/or interpretation. See the *Acknowledgements* section for more information.

- *Red-line Emission Geospace Observatory (REGO)*,
- *Time History of Events and Macroscale Interactions during Substorms (THEMIS)*,
- *Transition Region Explorer (TREx)*


Spatial Coverage of Imagers Supported by aurora-asi-lib
TREx-RGB
TREx-NIR
REGO
THEMIS
Generated on 2023-09-23


aurora-asi-lib
N
E
N
E

CHAPTER

ONE

ACKNOWLEDGEMENTS

asilib is not associated with the development and operations of all sky cameras, or the curation of ASI datasets. All data accessed by asilib is publicly available from the home institution responsible for the instrumentation. We recommend data users coordinate with instrument teams with regards to data issues and/or interpretation. Users are responsible to appropriately acknowledge the data sources they utilize. Required acknowledgements are contained in the descriptions of each instrument network.

If asilib significantly contributed to your research, and you would like to acknowledge it in your academic publication, please consider including the asilib developers as co-authors, and/or citing the following paper on asilib:

- Shumko M, Chaddock D, Gallardo-Lacourt B, Donovan E, Spanswick EL, Halford AJ, Thompson I and Murphy KR (2022), AuroraX, PyAuroraX, and aurora-asi-lib: A user-friendly auroral all-sky imager analysis framework. Front. Astron. Space Sci. 9:1009450. doi: 10.3389/fspas.2022.1009450

Lastly, asilib will not be possible without 1) everyone involved with designing, building, and maintaining all-sky imaging systems, and 2) everyone who contributed to the dependencies used by *asilib*. Some of the dependencies include:

- numpy: Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357-362 (2020). DOI: 10.1038/s41586-020-2649-2. ([Publisher link](https://www.nature.com/articles/s41586-020-2649-2)).
- Scipy: Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.
- aacgm2: Angeline Burrell for the Python source and Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501-7521, doi:10.1002/2014JA020264.
- pandas: Jeff Reback, Wes McKinney, jbrockmendel, Joris Van den Bossche, Tom Augspurger, Phillip Cloud, gfyoung, Sinhrks, Adam Klein, Matthew Roeschke, Simon Hawkins, Jeff Tratner, Chang She, William Ayd, Terji Petersen, Marc Garcia, Jeremy Schendel, Andy Hayden, MomIsBestFriend, … Mortada Mehyar. (2020). pandas-dev/pandas: Pandas 1.0.3 (v1.0.3). Zenodo. https://doi.org/10.5281/zenodo.3715232
- cartopy: Phil Elson, Elliott Sales de Andrade, Greg Lucas, Ryan May, Richard Hattersley, Ed Campbell, Andrew Dawson, Stephane Raynaud, scmc72, Bill Little, Alan D. Snow, Kevin Donkers, Byron Blay, Peter Killick, Nat Wilson, Patrick Peglar, lbdreyer, Andrew, Jon Szymaniak, … Mark Hedley. (2022). SciTools/cartopy: v0.20.2 (v0.20.2). Zenodo. https://doi.org/10.5281/zenodo.5842769
- IRBEM: Boscher, D., Bourdarie, S., O'Brien, P., Guild, T., & Shumko, M. (2012). IRBEM-lib library. https://zenodo.org/doi/10.5281/zenodo.6867552
- themis-imager-readfile: https://github.com/ucalgary-aurora/themis-imager-readfile

- rego-imager-readfile: https://github.com/ucalgary-aurora/rego-imager-readfile
- trex-imager-readfile: https://github.com/ucalgary-aurora/trex-imager-readfile

1.1 Get Started

1.1.1 Install

Installing asilib is as simple as:

```
python3 -m pip install asilib
```

Anaconda

asilib can also be installed with pip inside Anaconda. In a new environment install scipy first and then install asilib using the above instructions.

Note:

- By default, asilib saves the ASI data, movie images, and movies in the *~/asilib-data/* directory. To override the default directory, run asilib as a module, *python3 -m asilib config*. See the Configuration section below for more details.
- If you get the "*ERROR: Could not build wheels for pymap3d which use PEP 517 and cannot be installed directly*" error when installing, you need to upgrade your pip, setuptools, and wheel libaries via `` `python3 -m pip install --upgrade pip setuptools wheel` ``.

Dependencies

There are three optional dependencies that you may want to install if you want to use certain *asilib* functions.

Dependency	Purpose	asilib methods
ffmpeg	Animating Images	asilib.Imager.animate_fisheye() asilib.Imager.animate_map() asilib.Imager.animate_fisheye_gen() asilib.Imager.animate_map_gen()
IRBEM	Magnetic field footprint	asilib.Conjunction.lla_footprint()
cartopy	Projecting images onto a map	asilib.map.create_map()*

**create_map() will fallback to a simple map function if cartopy is not installed.*

Configuration

asilib writes the data and movie files to the *asilib.config['ASI_DATA_DIR']* directory. By default *ASI_DATA_DIR* is pointed at *~/asilib-data* and it is configurable. To configure *ASI_DATA_DIR*, and other asilib settings, run *python3 -m asilib config* and answer the prompts. The prompt answer in [brackets] is the default if you don't enter anything.

1.1.2 Core Concepts

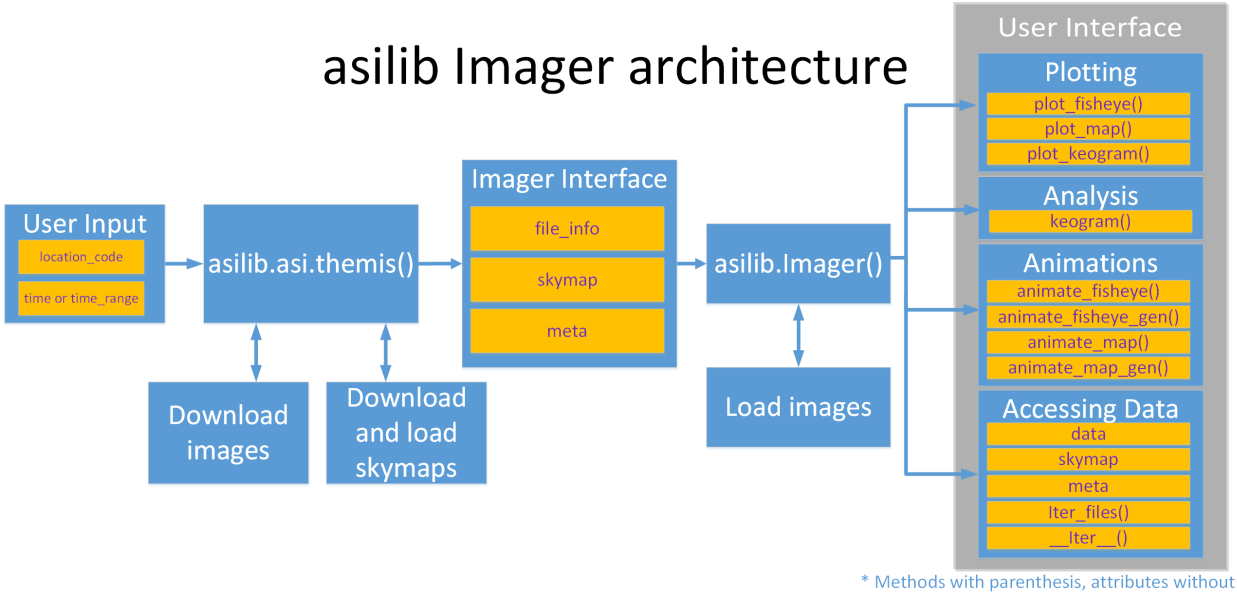
The core of the user interface is the `Imager()` class. It is invoked using an *entry function* such as `asilib.asi.themis()`, `asilib.asi.rego()`, or `asilib.asi.trex.trex_nir()`.

The entry function downloads the necessary image and skymap files, and passes the skymap arrays and image file paths to `Imager()`. The entry function also specifies a loader function that loads one file given it's path. `Imager()` uses these paths to load data as needed—also referred to as the "lazy mode"—to maintain a low money usage (necessary if working with high speed ASIs or simultaneously with multiple ASIs). If memory is not an issue, you can load all of the ASI data at once—also referred to as the "greedy mode".

Once initiated, `Imager()` exposes an intuitive user API to load, plot, animate, and analyze ASI data.

Note: Considering that some ASIs produce enough data to overwhelm your computer's memory, for example the Phantom ASIs in support of the LAMP sounding rocket produced 190 GB/hour of data, by default asilib loads data as needed. This is the "lazy" mode that prioritizes memory at the expense of longer run time. Alternatively, if memory is not a concern, asilib supports an "eager" mode that loads all of the data into memory. Eager mode is triggered by calling the *Imager().data* attribute.

The architecture described so far is illustrated in the flowchart below.



asilib also implements two classes to extend `Imager()`. First, `Conjunction()` finds and calculates auroral intensity near a satellite's footprint. Second, `Imagers()` plots and animates images from multiple `Imager()` instances.

`Conjunction()`: Often ASI observations need to be combined with in-situ measurements such as low Earth orbiting satellites. This involves mapping the ASI pixels and the satellite location to an assumed emission altitude (e.g., 110 km). This mapping is done via line-of-sight for the ASI pixels, and along magnetic field lines for the satellite (to the satellite's footprint).

Imagers() plots and animates images from multiple *Imager()* instances. This is useful for creating mosaics (multiple images mapped onto a map). While you can call `plot_map()` for each Imager, any overlapping fields of view will be overplotted except the final imager (see the zorder concept). `plot_map()` overcomes this issue for overlapping fields of view by plotting only the higher elevation pixels (the ones that have the least amount of spatial distortion.) Another reason to use *Imagers()* is to synchronize animating multiple *Imager()* fisheye or mapped images.

1.1.3 Examples

See the *Examples* gallery for fully-functioning examples of the fundamental asilib functionality.

1.1.4 Tutorial

See the *Tutorials* for comprehensive walk-throughs of the asilib functionality.

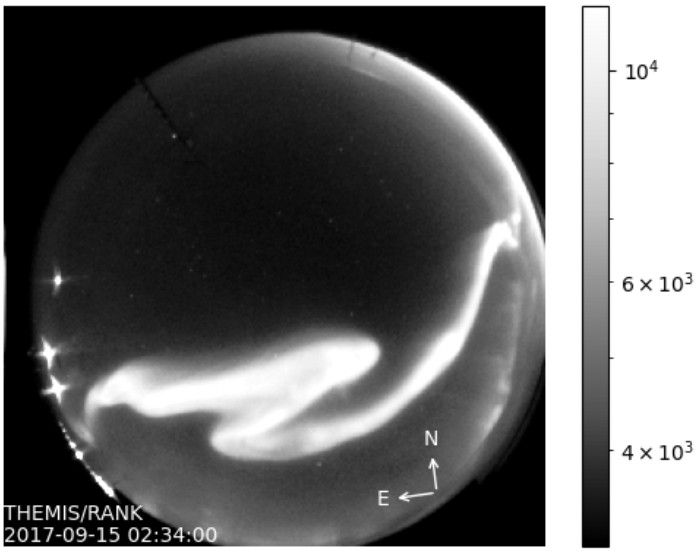
1.2 Examples

This example gallery using the best practices and illustrates functionality throughout *asilib*. These are complete examples that are also included in the `asilib/examples/` directory on GitHub.

1.2.1 Fisheye Lens View of an Arc

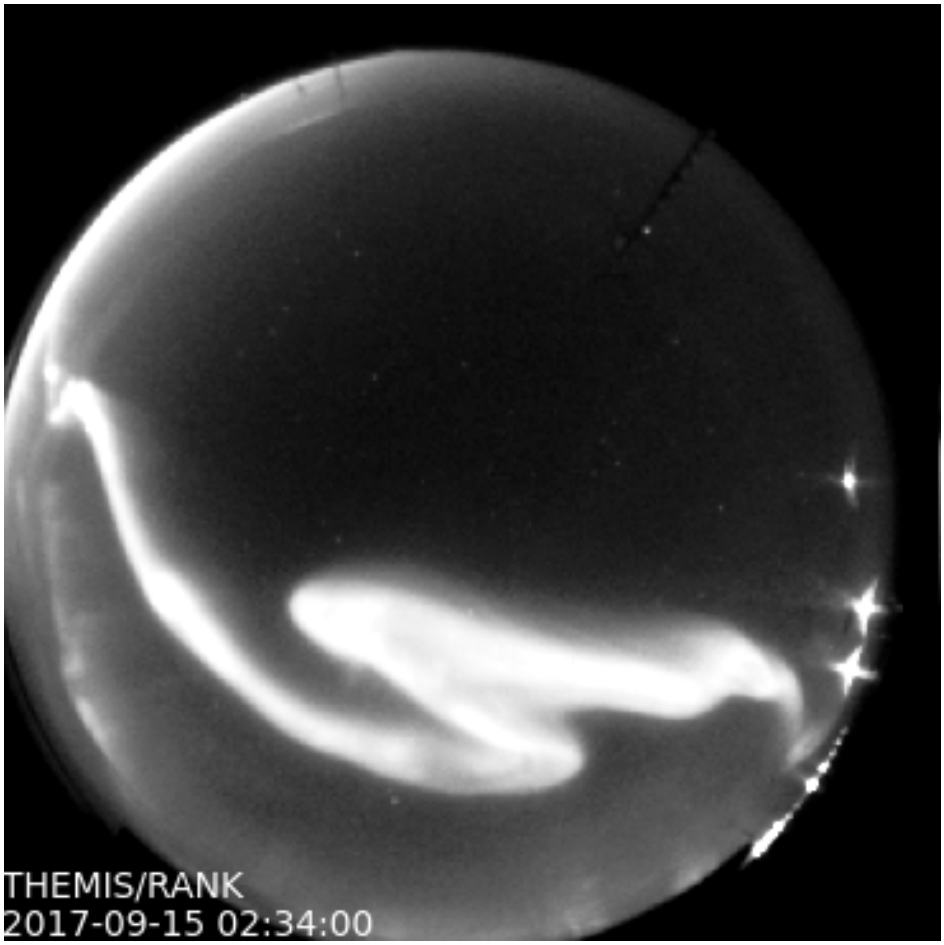
A bright auroral arc that was analyzed by Imajo et al. 2021 "Active auroral arc powered by accelerated electrons from very high altitudes"

Current Interface



```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib.asi

location_code = 'RANK'
time = datetime(2017, 9, 15, 2, 34, 0)

asi = asilib.asi.themis(location_code, time=time)
ax, im = asi.plot_fisheye()
plt.colorbar(im)
ax.axis('off')
plt.show()
```

Legacy Interface




```
from datetime import datetime

import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```
import asilib

asi_array_code = 'THEMIS'
location_code = 'RANK'
time = datetime(2017, 9, 15, 2, 34, 0)

# A bright auroral arc that was analyzed by Imajo et al., 2021 "Active
# auroral arc powered by accelerated electrons from very high altitudes"
image_time, image, ax, im = asilib.plot_fisheye(
    asi_array_code, location_code, time, color_norm='log', redownload=False
)
plt.colorbar(im)
ax.axis('off')
plt.show()
```

1.2.2 STEVE projected onto a map

Maps an image of STEVE (the thin band). Reproduced from http://themis.igpp.ucla.edu/nuggets/nuggets_2018/Gallardo-Lacourt/fig2.jpg

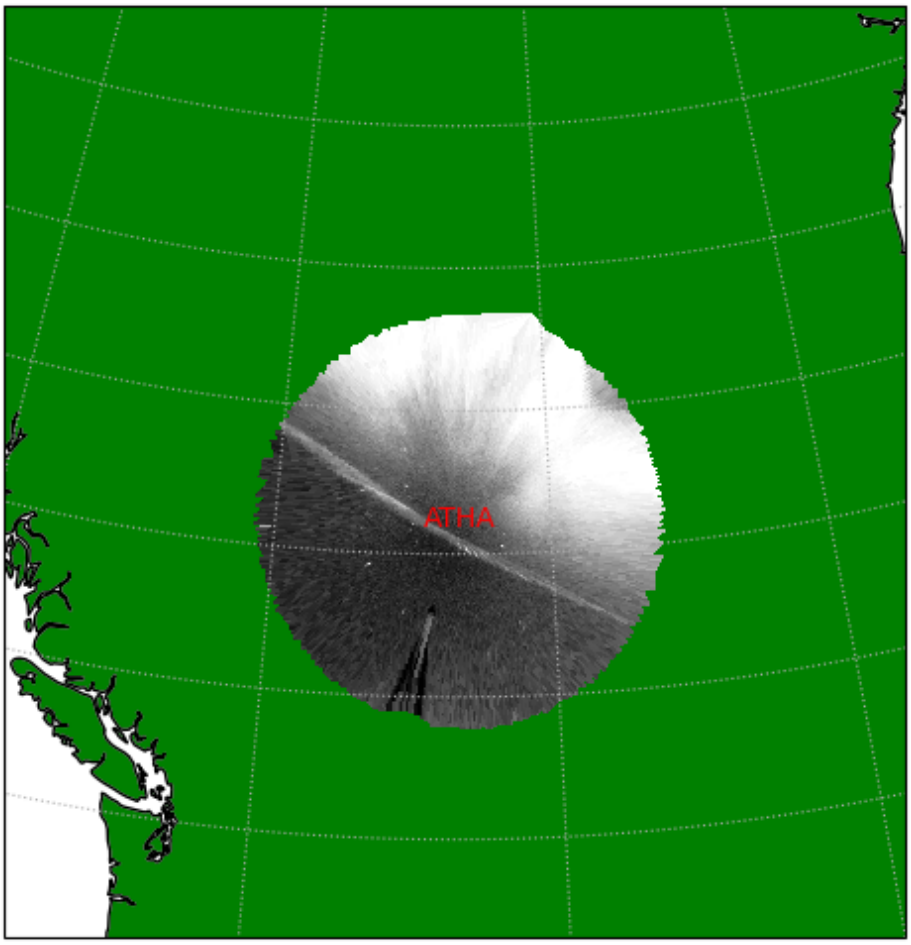
Current Interface

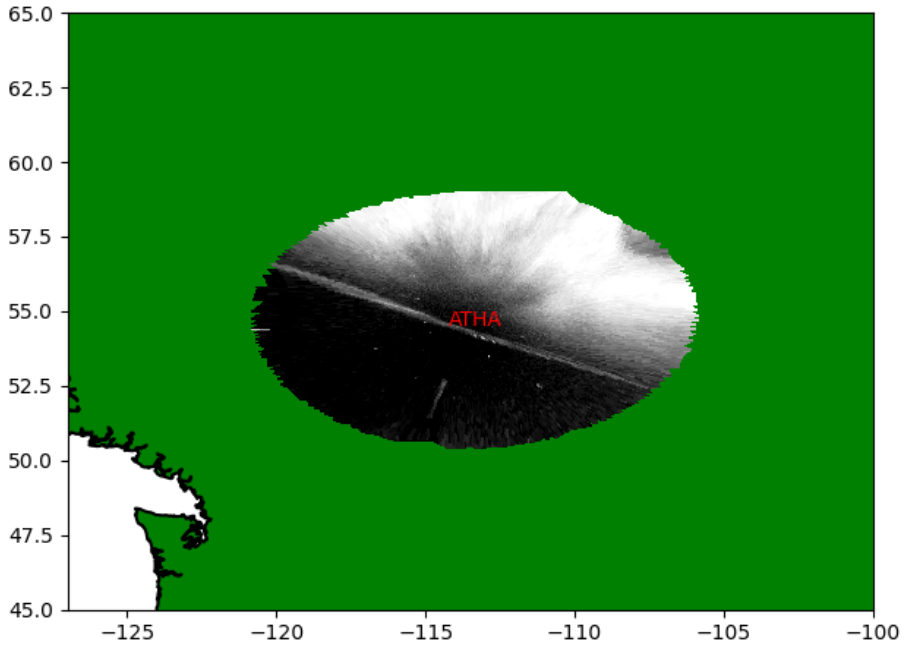
```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib.asi
import asilib.map

ax = asilib.map.create_map(lon_bounds=(-127, -100), lat_bounds=(45, 65))

asi = asilib.asi.themis('ATHA', time=datetime(2010, 4, 5, 6, 7, 0), alt=110)
asi.plot_map(ax=ax)
plt.tight_layout()
plt.show()
```

Legacy Interface

```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib

ax = asilib.make_map(lon_bounds=(-127, -100), lat_bounds=(45, 65))

image_time, image, skymap, ax, p = asilib.plot_map(
```

(continues on next page)


ATHA


65.0
62.5
60.0
57.5
55.0
52.5
50.0
47.5
45.0
−125
−120
−115
−110
−105
−100
ATHA

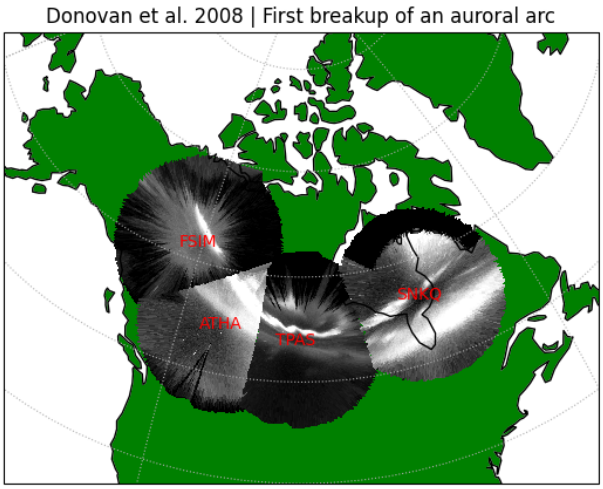
(continued from previous page)

```
    'THEMIS', 'ATHA', datetime(2010, 4, 5, 6, 7, 0), 110, ax=ax
)
plt.tight_layout()
plt.show()
```

1.2.3 Auroral arc projected onto a map

The first breakup of an auroral arc during a substorm analyzed by Donovan et al. 2008 “Simultaneous THEMIS in situ and auroral observations of a small substorm”

Current Interface



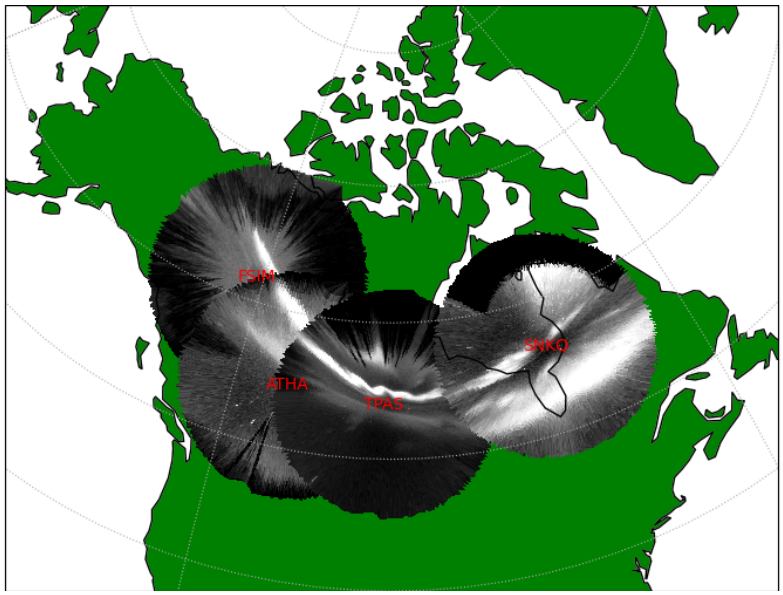

```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib
import asilib.map
import asilib.asi

time = datetime(2007, 3, 13, 5, 8, 45)
location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
map_alt = 110
min_elevation = 2

ax = asilib.map.create_simple_map(lon_bounds=(-140, -60), lat_bounds=(40, 82))

_imagers = []

for location_code in location_codes:
```

(continues on next page)

(continued from previous page)

```
    _imagers.append(asilib.asi.themis(location_code, time=time, alt=map_alt))

asis = asilib.Imagers(_imagers)
asis.plot_map(ax=ax, overlap=False, min_elevation=min_elevation)

ax.set_title('Donovan et al. 2008 | First breakup of an auroral arc')
plt.show()
```

Legacy Interface

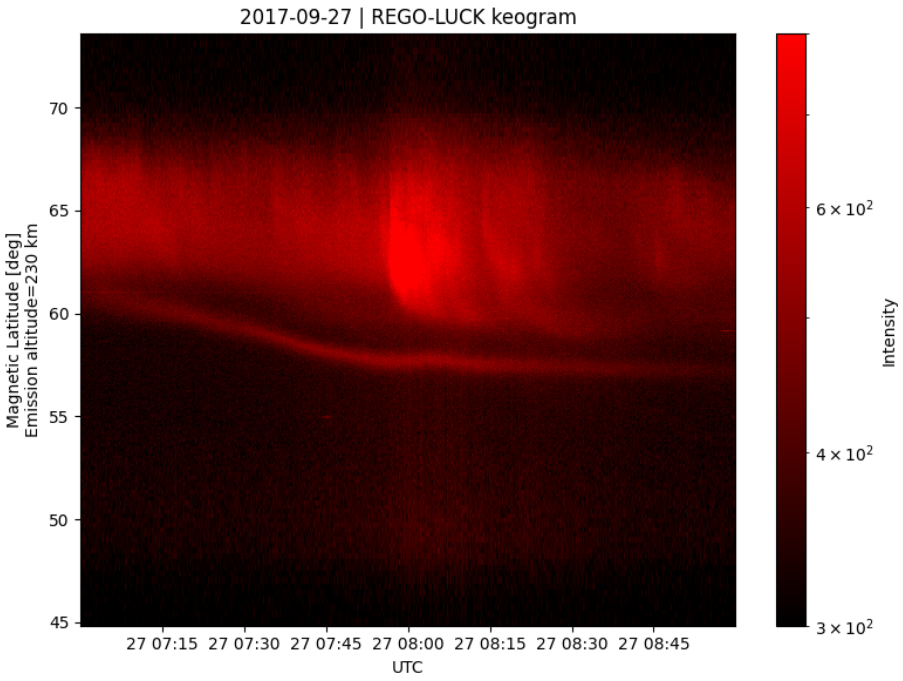



```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib

time = datetime(2007, 3, 13, 5, 8, 45)
asi_array_code = 'THEMIS'
location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
map_alt = 110
min_elevation = 2

ax = asilib.make_map(lon_bounds=(-160, -52), lat_bounds=(40, 82))

for location_code in location_codes:
    asilib.plot_map(
        asi_array_code, location_code, time, map_alt, ax=ax, min_elevation=min_elevation
    )

ax.set_title('Donovan et al. 2008 | First breakup of an auroral arc')
plt.show()
```

1.2.4 A keogram of STEVE

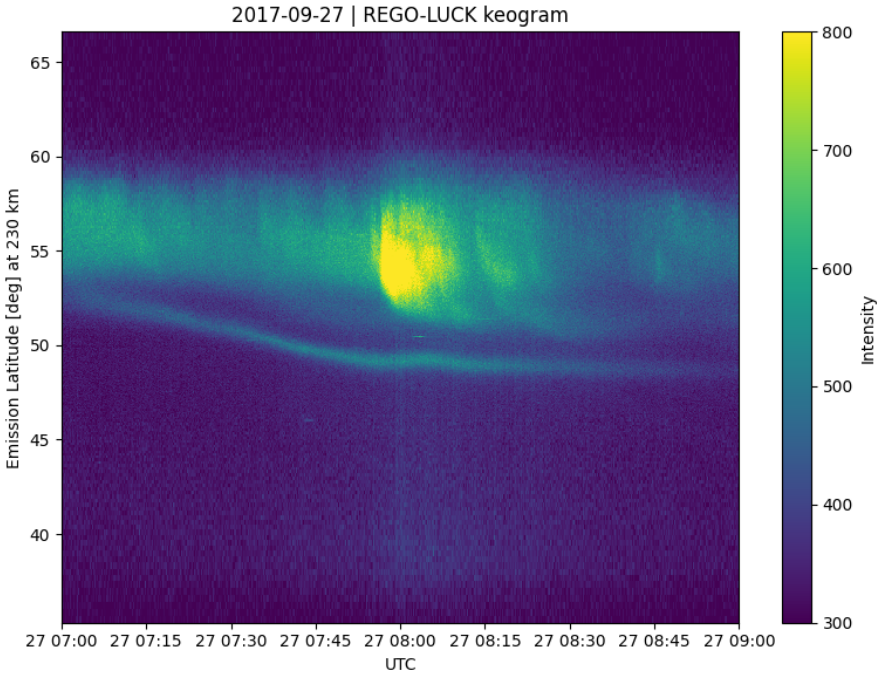
A keogram with a STEVE event that moved towards the equator. This event was analyzed in Gallardo-Lacourt et al. 2018 "A statistical analysis of STEVE"

Current Interface




```
import matplotlib.pyplot as plt

import asilib.asi

location_code = 'LUCK'
time_range = ['2017-09-27T07', '2017-09-27T09']
map_alt_km = 230

fig, ax = plt.subplots(figsize=(8, 6))
asi = asilib.asi.rego(location_code, time_range=time_range, alt=map_alt_km)
ax, p = asi.plot_keogram(ax=ax, color_bounds=(300, 800), aacgm=True)
plt.colorbar(p, label='Intensity')
ax.set_xlabel('UTC')
ax.set_ylabel(f'Magnetic Latitude [deg]\nEmission altitude={map_alt_km} km')
plt.tight_layout()
plt.show()
```

Legacy Interface

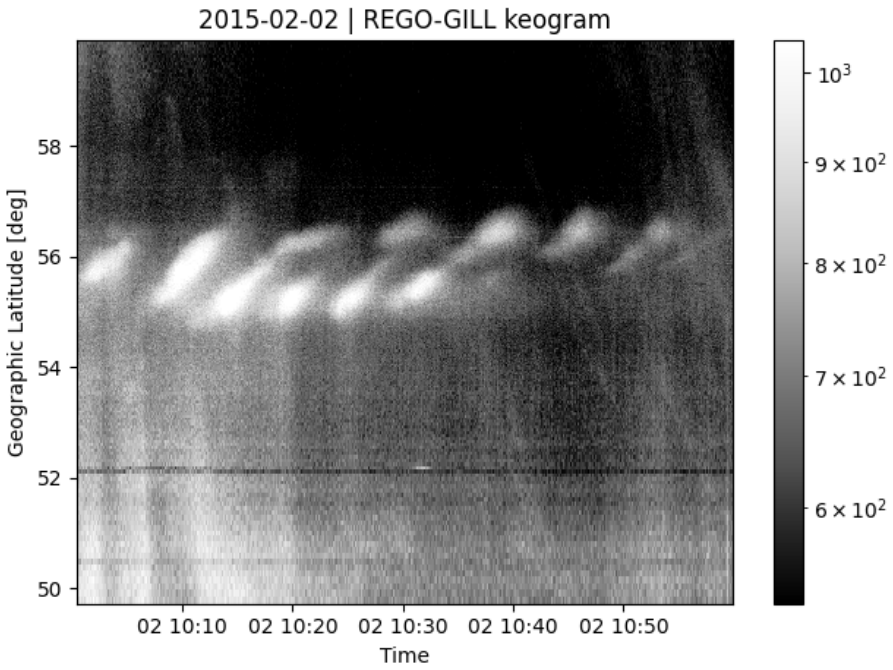



```
import matplotlib.pyplot as plt

import asilib

asi_array_code = 'REGO'
location_code = 'LUCK'
time_range = ['2017-09-27T07', '2017-09-27T09']
map_alt_km = 230

fig, ax = plt.subplots(figsize=(8, 6))
ax, im = asilib.plot_keogram(
    asi_array_code,
    location_code,
    time_range,
    ax=ax,
    map_alt=map_alt_km,
    color_bounds=(300, 800),
)
plt.colorbar(im, label='Intensity')
ax.set_xlabel('UTC')
ax.set_ylabel(f'Emission Latitude [deg] at {map_alt_km} km')
plt.tight_layout()
plt.show()
```

1.2.5 Keogram of a field line resonance

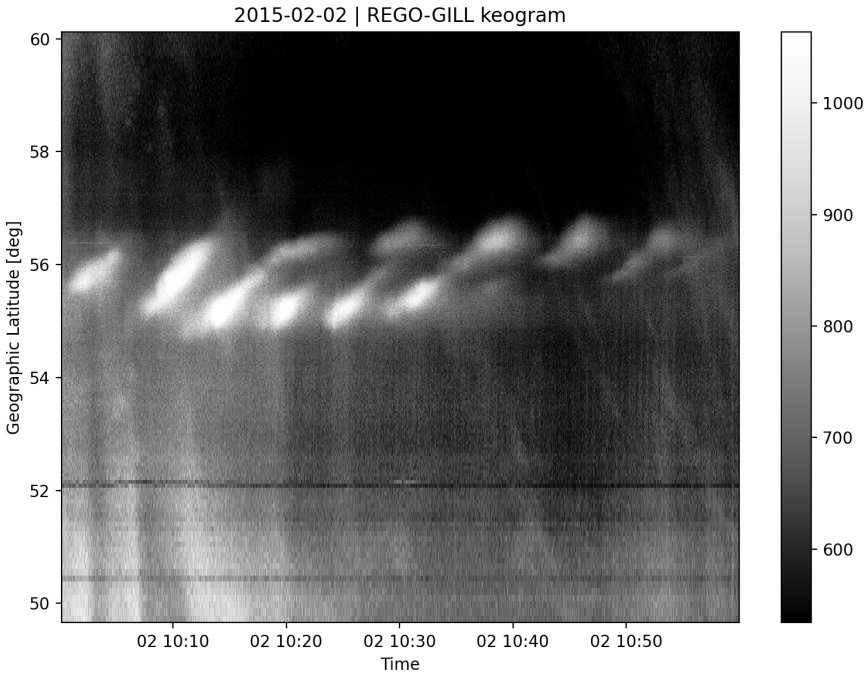
A field line resonance studied in: Gillies, D. M., Knudsen, D., Rankin, R., Milan, S., & Donovan, E. (2018). A statistical survey of the 630.0-nm optical signature of periodic auroral arcs resulting from magnetospheric field line resonances. Geophysical Research Letters, 45(10), 4648-4655.

Current Interface




```
import matplotlib.pyplot as plt

import asilib.asi

location_code = 'GILL'
time_range = ['2015-02-02T10', '2015-02-02T11']

asi = asilib.asi.rego(location_code, time_range=time_range, alt=230)
ax, p = asi.plot_keogram(color_map='Greys_r')
plt.xlabel('Time')
plt.ylabel('Geographic Latitude [deg]')
plt.colorbar(p)
plt.tight_layout()
plt.show()
```

Legacy Interface




```
import matplotlib.pyplot as plt

import asilib

asi_array_code = 'REGO'
location_code = 'GILL'
time_range = ['2015-02-02T10', '2015-02-02T11']

fig, ax = plt.subplots(figsize=(8, 6))
ax, im = asilib.plot_keogram(
    asi_array_code,
    location_code,
    time_range,
    ax=ax,
    map_alt=230,
    pcolormesh_kwargs={'cmap': 'Greys_r'},
)
plt.xlabel('Time')
plt.ylabel('Geographic Latitude [deg]')
plt.colorbar(im)
plt.tight_layout()
plt.show()
```

1.2.6 Fisheye Movie

Current Interface

```
from datetime import datetime

import asilib.asi

location_code = 'FSMI'
time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 30))
asi = asilib.asi.themis(location_code, time_range=time_range)
asi.animate_fisheye()

print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" / asi.animation_
↪name}')
```

Legacy Interface

```
from datetime import datetime

import asilib

asi_array_code = 'THEMIS'
location_code = 'FSMI'
time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 30))

asilib.animate_fisheye(asi_array_code, location_code, time_range, overwrite=True)
print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "animations"}')
```

1.2.7 Map movie

Current Interface

```
from datetime import datetime

import asilib.asi
import asilib.map

time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
location_code = 'FSMI'

asi = asilib.asi.themis(location_code, time_range=time_range, alt=110)

lat_bounds = (asi.meta['lat'] - 7, asi.meta['lat'] + 7)
lon_bounds = (asi.meta['lon'] - 20, asi.meta['lon'] + 20)
ax = asilib.map.create_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)

asi.animate_map(ax=ax)
```

(continues on next page)

(continued from previous page)

```
print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" / asi.animation_
↪name}')
```

Legacy Interface

```
from datetime import datetime

import asilib

time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
asi_array_code = 'THEMIS'
location_code = 'FSMI'

# We need the skymap only to center the map on the projected image.
skymap = asilib.load_skymap(asi_array_code, location_code, time_range[0])
lat_bounds = (skymap['SITE_MAP_LATITUDE']-7, skymap['SITE_MAP_LATITUDE']+7)
lon_bounds = (skymap['SITE_MAP_LONGITUDE']-20, skymap['SITE_MAP_LONGITUDE']+20)

ax = asilib.make_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
asilib.animate_map(asi_array_code, location_code, time_range, 110, overwrite=True, ax=ax)

print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "animations"}')
```

1.2.8 Animate Mosaic

```
import asilib
import asilib.asi

time_range = ('2021-11-04T06:55', '2021-11-04T07:05')
asis = asilib.Imagers(
    [asilib.asi.trex_rgb(location_code, time_range=time_range)
    for location_code in ['LUCK', 'PINA', 'GILL', 'RABB']]
    )
asis.animate_map(lon_bounds=(-115, -85), lat_bounds=(43, 63), overwrite=True)
```

1.2.9 ASI-satellite conjunction movie

A comprehensive example that maps a hypothetical satellite track to an image and calculates the mean ASI intensity in a 20x20 km box around the satellite's 100 km altitude footprint.

Current Interface

```
from datetime import datetime

import numpy as np
import matplotlib.pyplot as plt

import asilib
import asilib.asi


# ASI parameters
location_code = 'RANK'
alt=110  # km
time_range = (datetime(2017, 9, 15, 2, 32, 0), datetime(2017, 9, 15, 2, 35, 0))

fig, ax = plt.subplots(
    3, 1, figsize=(7, 10), gridspec_kw={'height_ratios': [4, 1, 1]}, constrained_
↪layout=True
)

asi = asilib.asi.themis(location_code, time_range=time_range, alt=alt)

# Create the fake satellite track coordinates: latitude, longitude, altitude (LLA).
# This is a north-south satellite track oriented to the east of the THEMIS/RANK
# imager.
n = int((time_range[1] - time_range[0]).total_seconds() / 3)  # 3 second cadence.
lats = np.linspace(asi.meta["lat"] + 5, asi.meta["lat"] - 5, n)
lons = (asi.meta["lon"] - 0.5) * np.ones(n)
alts = alt * np.ones(n)  # Altitude needs to be the same as the skymap.
sat_lla = np.array([lats, lons, alts]).T
# Normally the satellite time stamps are not the same as the ASI.
# You may need to call Conjunction.interp_sat() to find the LLA coordinates
# at the ASI timestamps.
sat_time = asi.data.time

conjunction_obj = asilib.Conjunction(asi, (sat_time, sat_lla))

# Map the satellite track to the imager's azimuth and elevation coordinates and
# image pixels. NOTE: the mapping is not along the magnetic field lines! You need
# to install IRBEM and then use conjunction.lla_footprint() before
# calling conjunction_obj.map_azel.
sat_azel, sat_azel_pixels = conjunction_obj.map_azel()

# Calculate the auroral intensity near the satellite and mean intensity within a 10x10␣
↪km area.
nearest_pixel_intensity = conjunction_obj.intensity(box=None)
```

(continues on next page)

(continued from previous page)

```
area_intensity = conjunction_obj.intensity(box=(10, 10))
area_mask = conjunction_obj.equal_area(box=(10,10))

# Need to change masked NaNs to 0s so we can plot the rectangular area contours.
area_mask[np.where(np.isnan(area_mask))] = 0

# Initiate the animation generator function.
gen = asi.animate_fisheye_gen(
    ax=ax[0], azel_contours=True, overwrite=True, cardinal_directions='NE'
)

for i, (time, image, _, im) in enumerate(gen):
    # Plot the entire satellite track, its current location, and a 20x20 km box
    # around its location.
    ax[0].plot(sat_azel_pixels[:, 0], sat_azel_pixels[:, 1], 'red')
    ax[0].scatter(sat_azel_pixels[i, 0], sat_azel_pixels[i, 1], c='red', marker='o',␣
↪s=50)
    ax[0].contour(area_mask[i, :, :], levels=[0.99], colors=['yellow'])

    if 'vline1' in locals():
        vline1.remove()  # noqa: F821
        vline2.remove()  # noqa: F821
        text_obj.remove()  # noqa: F821
    else:
        # Plot the ASI intensity along the satellite path
        ax[1].plot(sat_time, nearest_pixel_intensity)
        ax[2].plot(sat_time, area_intensity)
    vline1 = ax[1].axvline(time, c='b')
    vline2 = ax[2].axvline(time, c='b')

    # Annotate the location_code and satellite info in the top-left corner.
    location_code_str = (
        f'THEMIS/{location_code} '
        f'LLA=({asi.meta["lat"]:.2f}, '
        f'{asi.meta["lon"]:.2f}, {asi.meta["alt"]:.2f})'
    )
    satellite_str = f'Satellite LLA=({sat_lla[i, 0]:.2f}, {sat_lla[i, 1]:.2f}, {sat_
↪lla[i, 2]:.2f})'
    text_obj = ax[0].text(
        0,
        1,
        location_code_str + '\n' + satellite_str,
        va='top',
        transform=ax[0].transAxes,
        color='red',
    )
    ax[1].set(ylabel='ASI intensity\nnearest pixel [counts]')
    ax[2].set(xlabel='Time', ylabel='ASI intensity\n10x10 km area [counts]')

print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" / asi.animation_
↪name}')
```

Legacy Interface

```
from datetime import datetime

import numpy as np
import matplotlib.pyplot as plt

import asilib


# ASI parameters
asi_array_code = 'THEMIS'
location_code = 'RANK'
time_range = (datetime(2017, 9, 15, 2, 32, 0), datetime(2017, 9, 15, 2, 35, 0))

fig, ax = plt.subplots(
    2, 1, figsize=(7, 10), gridspec_kw={'height_ratios': [4, 1]}, constrained_layout=True
)

# Load the skymap calibration data. This is only necessary to create a fake satellite␣
↪track.
skymap_dict = asilib.load_skymap(asi_array_code, location_code, time_range[0])

# Create the fake satellite track coordinates: latitude, longitude, altitude (LLA).
# This is a north-south satellite track oriented to the east of the THEMIS/RANK
# imager.
n = int((time_range[1] - time_range[0]).total_seconds() / 3)  # 3 second cadence.
lats = np.linspace(skymap_dict["SITE_MAP_LATITUDE"] + 5, skymap_dict["SITE_MAP_LATITUDE
↪"] - 5, n)
lons = (skymap_dict["SITE_MAP_LONGITUDE"] - 0.5) * np.ones(n)
alts = 110 * np.ones(n)
lla = np.array([lats, lons, alts]).T

# Map the satellite track to the imager's azimuth and elevation coordinates and
# image pixels. NOTE: the mapping is not along the magnetic field lines! You need
# to install IRBEM and then use asilib.lla2footprint() before
# lla2azel() is called.
sat_azel, sat_azel_pixels = asilib.lla2azel(asi_array_code, location_code, time_range[0],
↪ lla)

# Initiate the movie generator function. Any errors with the data will be raised here.
movie_generator = asilib.animate_fisheye_generator(
    asi_array_code, location_code, time_range, azel_contours=True, overwrite=True,␣
↪ax=ax[0]
)

# Use the generator to get the images and time stamps to estimate mean the ASI
# brightness along the satellite path and in a (20x20 km) box.
image_data = movie_generator.send('data')

# Calculate what pixels are in a box_km around the satellite, and convolve it
# with the images to pick out the ASI intensity in that box.
area_box_mask = asilib.equal_area(
```

(continues on next page)

(continued from previous page)

```
    asi_array_code, location_code, time_range[0], lla, box_km=(20, 20)
)
asi_brightness = np.nanmean(image_data.images * area_box_mask, axis=(1, 2))
area_box_mask[np.isnan(area_box_mask)] = 0  # To play nice with plt.contour()

for i, (time, image, _, im) in enumerate(movie_generator):
    # Note that because we are drawing different data in each frame (a unique ASI
    # image in ax[0] and the ASI time series + a guide in ax[1], we need
    # to redraw everything at every iteration.

    ax[1].clear() # ax[0] cleared by asilib.animate_fisheye_generator()
    # Plot the entire satellite track, its current location, and a 20x20 km box
    # around its location.
    ax[0].plot(sat_azel_pixels[:, 0], sat_azel_pixels[:, 1], 'red')
    ax[0].scatter(sat_azel_pixels[i, 0], sat_azel_pixels[i, 1], c='red', marker='o',␣
↪s=50)
    ax[0].contour(area_box_mask[i, :, :], levels=[0.99], colors=['yellow'])

    # Plot the time series of the mean ASI intensity along the satellite path
    ax[1].plot(image_data.time, asi_brightness)
    ax[1].axvline(time, c='k')

    # Annotate the location_code and satellite info in the top-left corner.
    location_code_str = (
        f'{asi_array_code}/{location_code} '
        f'LLA=({skymap_dict["SITE_MAP_LATITUDE"]:.2f}, '
        f'{skymap_dict["SITE_MAP_LONGITUDE"]:.2f}, {skymap_dict["SITE_MAP_ALTITUDE"]:.2f}
↪)'
    )
    satellite_str = f'Satellite LLA=({lla[i, 0]:.2f}, {lla[i, 1]:.2f}, {lla[i, 2]:.2f})'
    ax[0].text(
        0,
        1,
        location_code_str + '\n' + satellite_str,
        va='top',
        transform=ax[0].transAxes,
        color='red',
    )
    ax[1].set(xlabel='Time', ylabel='Mean ASI intensity [counts]')

print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "animations"}')
```

1.3 Tutorials

1.3.1 Basics

Welcome! This tutorial will guide you though the main functions in `asilib`.

First off, we need to import the necessary packages.

[1]:
```
from datetime import datetime, timedelta
from IPython.display import Video
import numpy as np

import matplotlib.pyplot as plt
import asilib
import asilib.asi
import asilib.map

plt.style.use('dark_background')

print(f'asilib version: {asilib.__version__}')
```

```
asilib version: 0.20.5
```

First of all, you should know where the data and movies are saved to. This information is in `asilib.config` and can be changed with `python3 -m asilib config` to configure asilib.

[2]:
```
asilib.config
```

[2]:
```
{'ASILIB_DIR': WindowsPath('C:/Users/shumkms1/Documents/research/aurora-asi-lib/asilib'),
 'ASI_DATA_DIR': WindowsPath('C:/Users/shumkms1/asilib-data')}
```

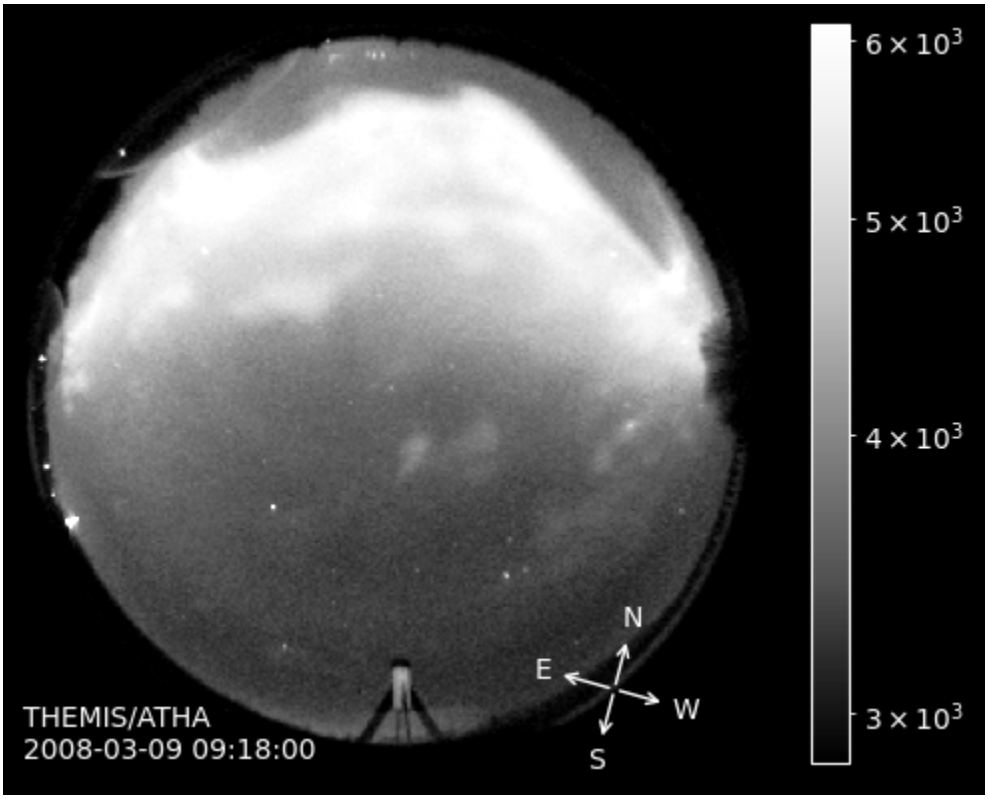
As you can guess, `asilib.config['ASILIB_DIR']` is the directory where this code resides, `asilib.config['ASI_DATA_DIR']` is the directory where the data is saved to.

Plot a single image

The core of asilib is the `asilib.Imager` class. It provides an intuitive interface to load, plot, animate, and analyze auroral images. Normally, you do not need to call `asilib.Imager()` directly. Instead, you initialize an `asilib.Imager` object via a ASI interface function, such as `asilib.asi.themis`, `asilib.asi.rego`, or `asilib.asi.trex_nir`. Let's see how this works by plotting a fisheye lens image from a THEMIS imager at Athabasca (ATHA). We will show the aurora studied in:

Liu, J., Lyons, L. R., Archer, W. E., Gallardo-Lacourt, B., Nishimura, Y., Zou, Y., … Weygand, J. M. (2018). Flow shears at the poleward boundary of omega bands observed during conjunctions of Swarm and THEMIS ASI. Geophysical Research Letters, 45, 1218– 1227. https://doi.org/10.1002/2017GL076485

[3]:
```
location_code = 'ATHA'
time = datetime(2008, 3, 9, 9, 18, 0)  # You can supply a datetime object or a ISO-
↪formatted time string.

asi = asilib.asi.themis(location_code, time=time)
ax, im = asi.plot_fisheye(cardinal_directions='news')
plt.colorbar(im)
ax.axis('off');
```

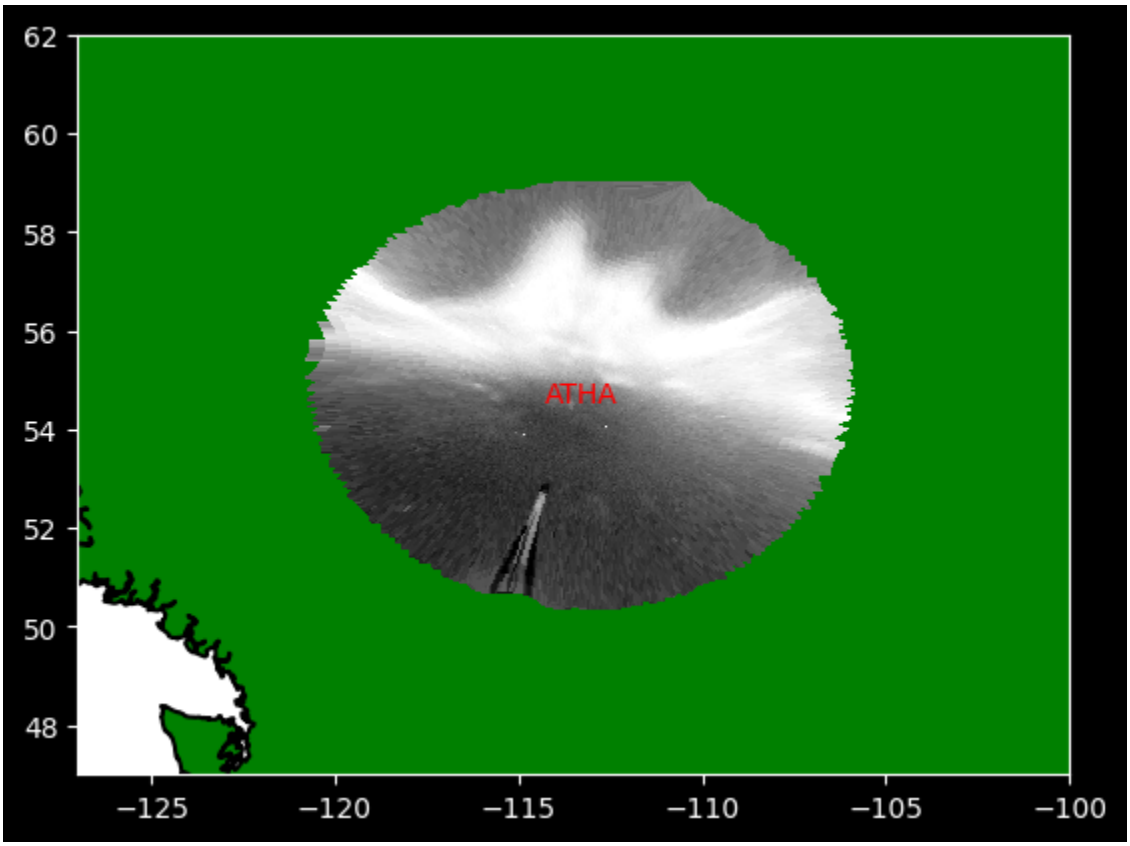


That is it! By calling the `asilib.asi.themis()` function, we create an `asilib.Imager()` object—with uniform interface. In other words, plotting a fisheye image (`Imager.plot_fisheye()`) is the same for THEMIS, REGO, TREx, or any other ASI supported by asilib. This is what makes asilib so powerful.

It is also easy to map the ASI fisheye lens image to a geographic map using the `Imager.plot_map()` method. In the code box below, the first line creates a geographic map centered on Athabasca and the second line projects the image onto the map.

Note: If latitude or longitude bounds are not provided, `Imager.plot_map()` defaults to a map of North America.

[4]:
```
ax = asilib.map.create_simple_map(lon_bounds=(-127, -100), lat_bounds=(47, 62))
asi.plot_map(ax=ax);
```



Notice that you did not need to explicitly download or load the data. The ASI Interface Function downloads the necessary image and skymap files, while `asilib.Imager()` loads data as needed, also known as the "lazy" mode. This preserves your computer's memory at the expense of processing speed. Alternatively, you can load all of the data into memory at once with the "eager" mode.

[5]:
```
asi_data = asi.data
```

[6]:
```
asi_data.time
```

[6]:
```
datetime.datetime(2008, 3, 9, 9, 18, 0, 50605)
```

[7]:
```
asi_data.image.shape
```

[7]:
```
(256, 256)
```

[8]:
```
asi_data.image
```

[8]:
```
array([[2536, 2616, 2554, ..., 2572, 2537, 2546],
       [2582, 2582, 2620, ..., 2562, 2613, 2608],
       [2544, 2560, 2568, ..., 2588, 2526, 2550],
       ...,
       [2525, 2546, 2553, ..., 2612, 2541, 2629],
       [2545, 2596, 2698, ..., 2510, 2568, 2569],
       [2502, 2577, 2602, ..., 2514, 2617, 2576]], dtype=uint16)
```

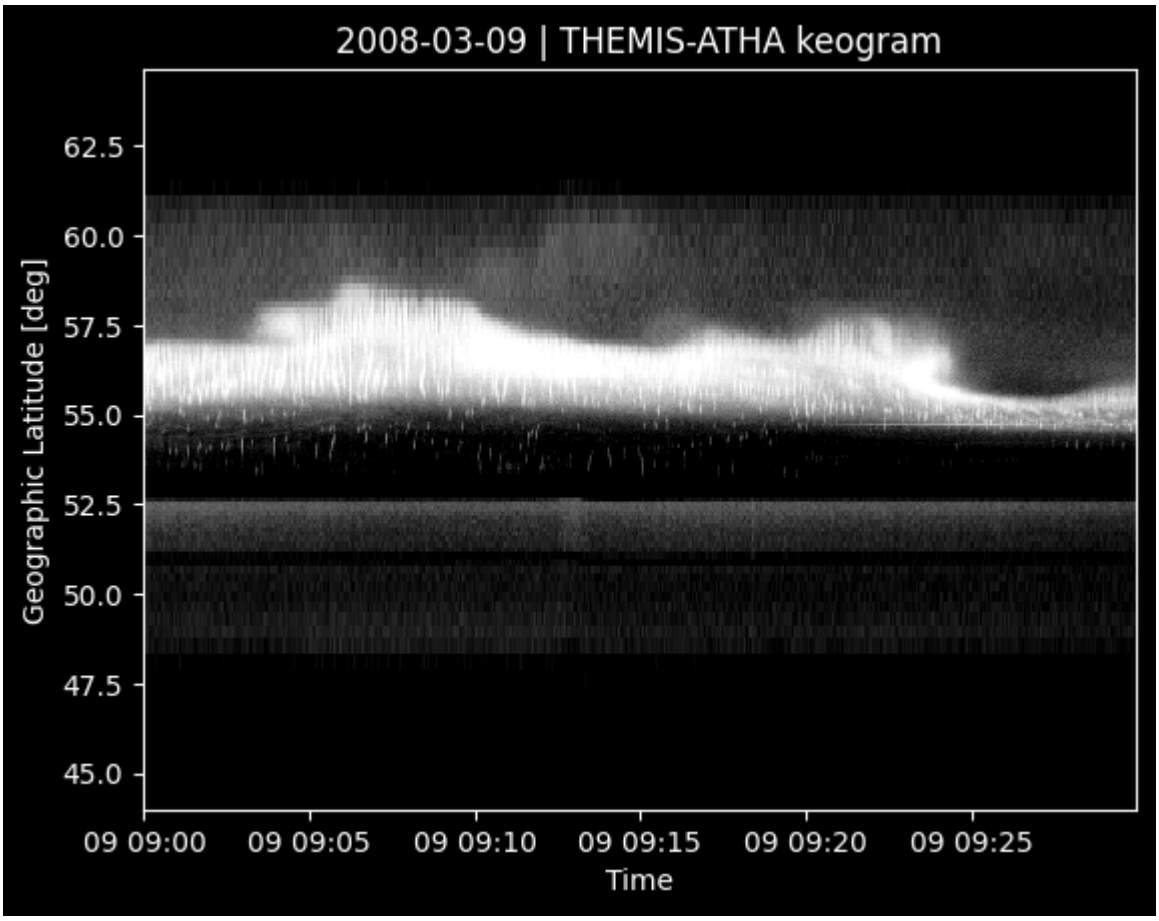
Plot a Keogram

`Imager.plot_keogram()` plots a keogram through the meridian. Alternatively, you can specify a custom path using (latitude, longitude coordinates).

By default, the y-axis is geographic latitude. If you set `aacgm=True`, the keogram's vertical axis will be magnetic latitude estimated using the aacgmv2 Python package. It implements the Altitude-adjusted corrected geomagnetic coordinates defined in Shepherd 2014.

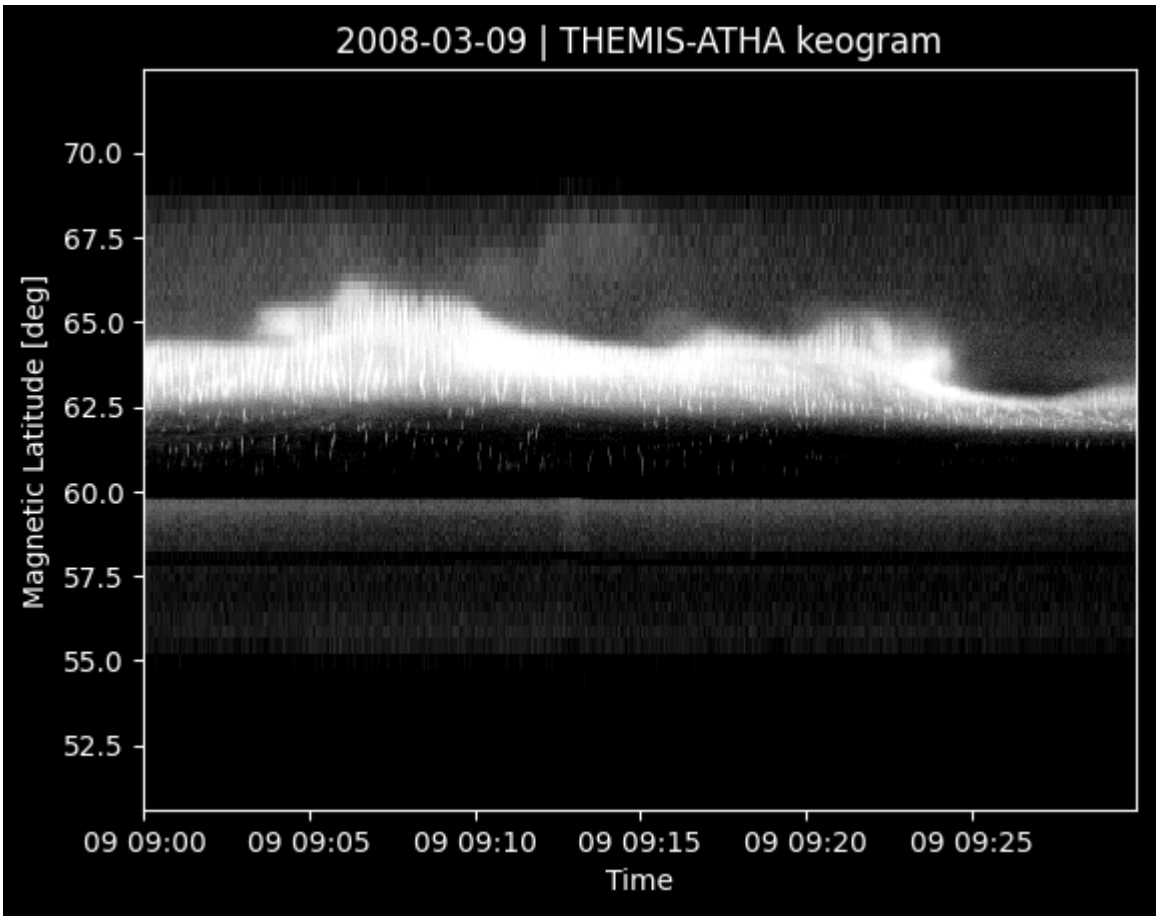
[9]:
```
time_range = [datetime(2008, 3, 9, 9, 0, 0), datetime(2008, 3, 9, 9, 30, 0)]
asi2 = asilib.asi.themis(location_code, time_range=time_range)
asi2.plot_keogram()
plt.xlabel('Time'); plt.ylabel('Geographic Latitude [deg]');
```

```
THEMIS ATHA keogram: |##################################################| 100%
```



[10]:
```
asi2.plot_keogram(aacgm=True)
plt.xlabel('Time'); plt.ylabel('Magnetic Latitude [deg]');
```

```
THEMIS ATHA keogram: |##################################################| 100%
```



In making the last three plots, the geographic latitudes correspond to pixels mapped to a specified auroral emission altitude. This altitude is set with the `alt` kwarg passed into the ASI Interface Function. The default altitude is 110 km for THEMIS and 230 km for REGO.

If you pick an incorrect `alt`, you will get an error.

[11]:
```
try:
    asilib.asi.themis(location_code, time_range=time_range, alt=100)
except AssertionError as err:
    print('AssetionError:', err)
```

```
AssetionError: 100 km is not in the valid skymap altitudes: [ 90. 110. 150.] km. If you␣
↪want a custom altitude with less percision, please use the custom_alt keyword
```

These auroral emission altitudes are defined in the skymap files.

Skymap calibration files

You may wonder how the image's pixel values were mapped to geographic latitude. This is done via the skymap calibration files that are provided by the instrument teams. They contain four arrays that map the ASI's pixels to:

- `el` - elevation
- `az` - azimuth
- `lat` - latitude
- `lon` - longitude

These skymaps are essential for mapping images onto a geographic map, and for calculating the auroral intensity for conjunction studies.

[12]:
```
asi2.skymap.keys()
```

[12]:
```
dict_keys(['lat', 'lon', 'alt', 'el', 'az', 'path'])
```

The ASI maintainers often move or adjust their ASIs, after which a new skymap is often produced. Therefore, the relevant skymap is the one taken right before the images that were loaded. By default, asilib downloads all of the skymaps. The skymap file name below indicates that it is valid for images taken between 1 March 2007 and 22 May 2009.

[13]:
```
asi2.skymap['path'].name
```

[13]:
```
'themis_skymap_atha_20070301-20090522_vXX.sav'
```

Lastly, the `Imager` instance has a `meta` attribute that contains the ASI metadata such as location, pixel resolution, and cadence.

[14]:
```
asi2.meta
```

[14]:
```
{'array': 'THEMIS',
 'location': 'ATHA',
 'lat': 54.720001220703125,
 'lon': -113.30999755859375,
 'alt': 0.676,
 'cadence': 3,
 'resolution': (256, 256)}
```

You can also see human-readable summary of the ASI by printing it.

[15]:
```
print(asi2)
```

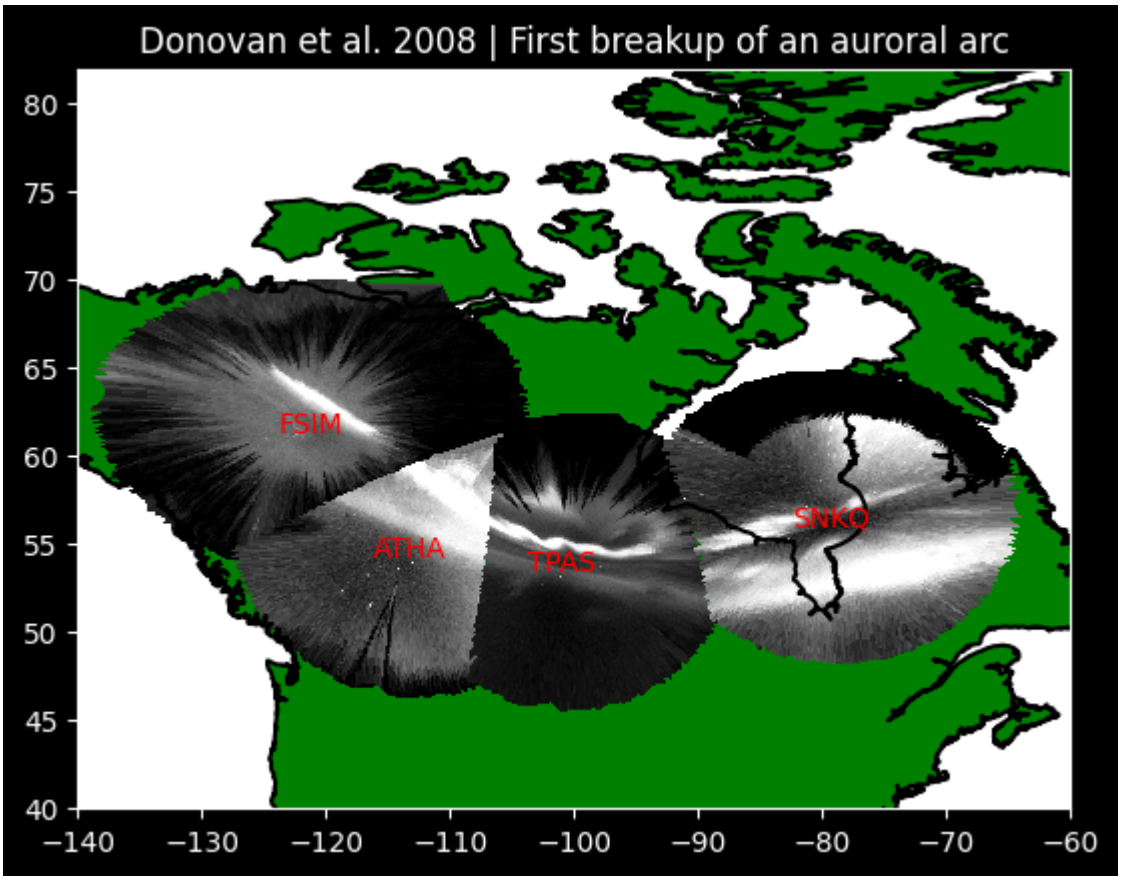
```
A THEMIS-ATHA Imager. time_range=[datetime.datetime(2008, 3, 9, 9, 0), datetime.
↪datetime(2008, 3, 9, 9, 30)]
```

Mapping multiple all-sky images

You can plot one image from multiple ASI locations using a for-loop. In the following example, we will replicate Fig. 2b from:

Donovan, E., Liu, W., Liang, J., Spanswick, E., Voronkov, I., Connors, M., … & Rae, I. J. (2008). Simultaneous THEMIS in situ and auroral observations of a small substorm. Geophysical Research Letters, 35(17).

[18]:
```
time = datetime(2007, 3, 13, 5, 8, 45)
location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
map_alt = 110
min_elevation = 2  # Plot only pixels observed above some minimum elevation.

bx = asilib.map.create_simple_map()

asis = asilib.Imagers(
    [asilib.asi.themis(location_code, time=time, alt=map_alt) for location_code in␣
↪location_codes]
    )
asis.plot_map(ax=bx, min_elevation=min_elevation)

bx.set_title('Donovan et al. 2008 | First breakup of an auroral arc')
plt.show()
```

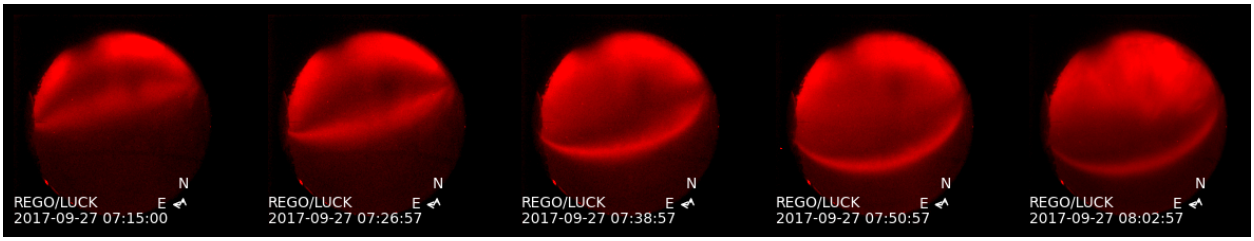


Working with multiple images

`asilib.Imager` supports slicing by time. In the example below, we plot a few fisheye and mapped images of STEVE that was observed by the REGO imagers.

Gallardo-Lacourt, B., Nishimura, Y., Donovan, E., Gillies, D. M., Perry, G. W., Archer, W. E., et al. (2018). A statistical analysis of STEVE. Journal of Geophysical Research: Space Physics, 123, 9893– 9905. https://doi.org/10.1029/2018JA025368

[19]:
```
time_range = [datetime(2017, 9, 27, 7, 15), datetime(2017, 9, 27, 8, 15)]
n_plots = 5
dt = int((time_range[1]-time_range[0]).total_seconds()/n_plots)
image_times = [time_range[0]+timedelta(seconds=i*dt) for i in range(n_plots)]

asi = asilib.asi.rego('LUCK', time_range=time_range)

fig, cx = plt.subplots(1, n_plots, figsize=(15, 8))

for montage_time, cx_i in zip(image_times, cx):
    filtered_asi = asi[montage_time]
    filtered_asi.plot_fisheye(ax=cx_i)
    cx_i.axis('off')
```



Animate images

Let's now make a simple fisheye lens movie of a substorm using `Imager.animate_fisheye()`.

[20]:
```
location_code = 'FSMI'
time_range = [datetime(2015, 3, 26, 6, 7, 0), datetime(2015, 3, 26, 6, 30, 0)]

# loglevel is to suppress the verbose ffmpeg output.
asi = asilib.asi.themis(location_code, time_range=time_range)
asi.animate_fisheye(overwrite=True, ffmpeg_params={'loglevel':'quiet'})
plt.close()  # To show a clean output in this tutorial---it is often unnecessary.

# When you run this, you should see the video below in your asilib-data/movies directory.
Video('https://github.com/mshumko/asilib/blob/e6147ff1a3309c602c1aa48711ebc8a90a7863e1/
↪docs/_static/'
    '20150326_060700_062957_themis_fsmi.mp4?raw=true')
```

```
20150326_060700_063000_themis_fsmi_fisheye.mp4: |######################| 100%
Animation saved to C:\Users\shumkms1\asilib-data\animations\20150326_060700_063000_
↪themis_fsmi_fisheye.mp4
```

[20]:
```
<IPython.core.display.Video object>
```

Animating images projected onto a map is also straightforward.

[21]:
```
location_code = 'FSMI'
time_range = [datetime(2015, 3, 26, 6, 7, 0), datetime(2015, 3, 26, 6, 30, 0)]

# loglevel is to suppress the verbose ffmpeg output.
asi = asilib.asi.themis(location_code, time_range=time_range)

lat_bounds = (asi.meta['lat']-7, asi.meta['lat']+7)
lon_bounds = (asi.meta['lon']-10, asi.meta['lon']+10)

dx = asilib.make_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
plt.subplots_adjust(top=0.99, bottom=0.05, left=0.05, right=0.99)

asi.animate_map(overwrite=True, ax=dx, ffmpeg_params={'loglevel':'quiet'})

plt.close()  # To show a clean output in this tutorial---it is often unnecessary.

# When you run this, you should see the video below in your asilib-data/animations␣
↪directory.
Video('https://github.com/mshumko/asilib/docs/_static/v1/20150326_060700_061200_themis_
↪fsmi_map.mp4?raw=true')
```

```
20150326_060700_063000_themis_fsmi_map.mp4: |##########################| 100%
Animation saved to C:\Users\shumkms1\asilib-data\animations\20150326_060700_063000_
↪themis_fsmi_map.mp4
```

[21]:
```
<IPython.core.display.Video object>
```

If you need to annotate the animation, `asilib.Imager` has `animate_fisheye_gen` and `animate_map_gen` generator functions. After it plots each auroral image, it allows you to superpose your data.

Note: in each iteration, these methods do not clear the subplot; all plot objects persist unless you explicitly remove them

[22]:
```
location_code = 'FSMI'
time_range = [datetime(2015, 3, 26, 6, 10, 0), datetime(2015, 3, 26, 6, 30, 0)]

# loglevel is to suppress the verbose ffmpeg output.
asi = asilib.asi.themis(location_code, time_range=time_range)

lat_bounds = (asi.meta['lat']-7, asi.meta['lat']+7)
lon_bounds = (asi.  meta['lon']-10, asi.meta['lon']+10)

ex = asilib.make_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
plt.subplots_adjust(top=0.99, bottom=0.05, left=0.05, right=0.99)
gen = asi.animate_map_gen(overwrite=True, ax=ex, ffmpeg_params={'loglevel':'quiet'}, asi_
↪label=False)

for time, image, ax, im in gen:
    if 'time_label' in locals():
        # This is one way I found to clean up an added plotting object.
        time_label.remove()
    time_label = ex.text(0.99, 0.99, f'location: {location_code} | time: {time}',
                         ha='right', va='top', transform=ex.transAxes, fontsize=15)
```

(continues on next page)

(continued from previous page)

```
plt.close()  # To show a clean output in this tutorial---it is often unnecessary.

# When you run this, you should see the video below in your asilib-data/animations␣
↪directory.
Video('https://github.com/mshumko/asilibin/docs/_static/v1/20150326_061000_063000_themis_
↪fsmi_map.mp4?raw=true')
```

```
20150326_061000_063000_themis_fsmi_map.mp4: |############################| 100%
Animation saved to C:\Users\shumkms1\asilib-data\animations\20150326_061000_063000_
↪themis_fsmi_map.mp4
```

[22]:
```
<IPython.core.display.Video object>
```

1.3.2 Mosaics

[1]:
```
from datetime import datetime

import matplotlib.pyplot as plt
import matplotlib.colors
import aacgmv2
import numpy as np
import scipy.interpolate

import asilib
import asilib.asi
import asilib.map

print(f'asilib version: {asilib.__version__}')
```

```
asilib version: 0.20.3
```

We first create an `asilib.Imagers()` object consisting of TREx-RGB `asilib.Imagers()` defined a list of location_codes.

[2]:
```
asilib.asi.trex.trex_rgb_info()
```

[2]:

	array	location_code	name	latitude	longitude
0	TREx_RGB	ATHA	Athabasca	54.60	-113.64
1	TREx_RGB	FSMI	Fort Smith	60.03	-111.93
2	TREx_RGB	GILL	Gillam	56.38	-94.64
3	TREx_RGB	LUCK	Lucky Lake	51.15	-107.26
4	TREx_RGB	PINA	Pinawa	50.26	-95.87
5	TREx_RGB	RABB	Rabbit Lake	58.23	-103.68

[3]:
```
time = datetime(2021, 11, 4, 7, 3, 51)
location_codes = ['FSMI', 'LUCK', 'RABB', 'PINA', 'GILL']
map_alt = 110
min_elevation = 10
```

[4]:
```
_imagers = []

for location_code in location_codes:
```

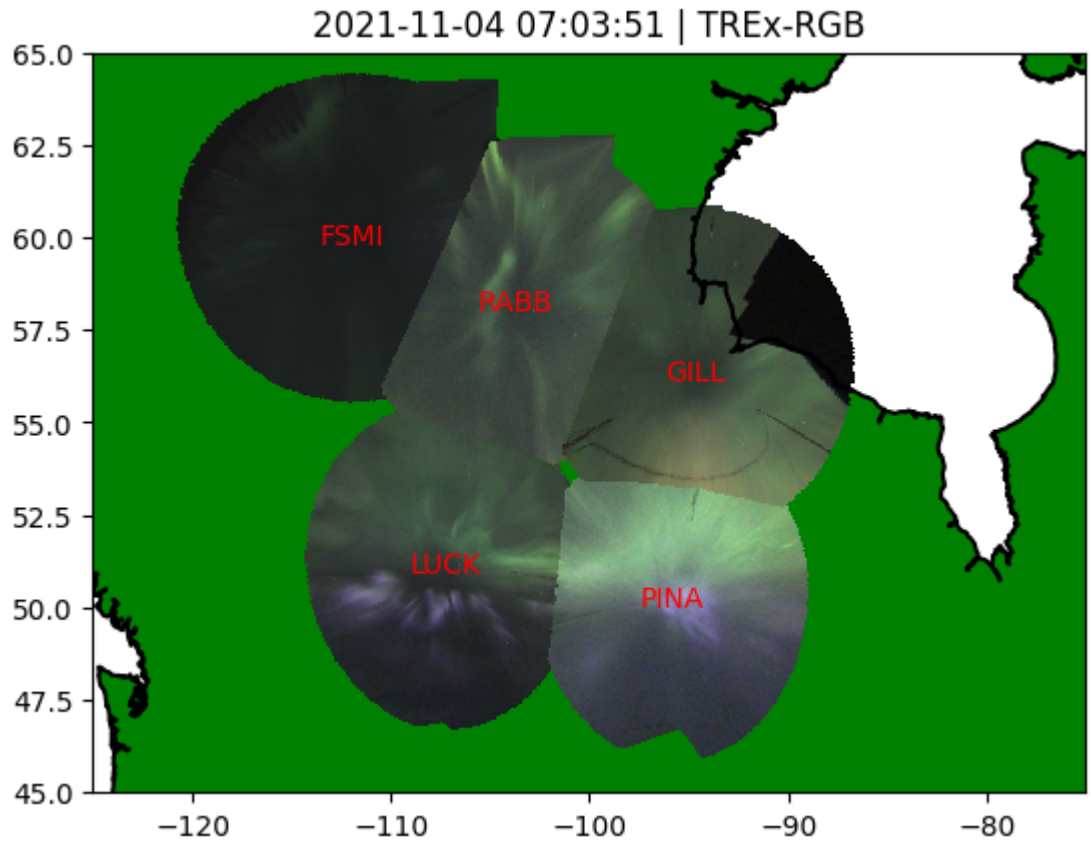
(continues on next page)

(continued from previous page)

```
    _imagers.append(asilib.asi.trex.trex_rgb(location_code, time=time, alt=map_alt))

asis = asilib.Imagers(_imagers)
```

Plot a TREx-RGB mosaic with and without AACGM magnetic latitude contours.

[5]:
```
lon_bounds=(-125, -75)
lat_bounds=(45, 65)
ax = asilib.map.create_simple_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
asis.plot_map(ax=ax, overlap=False, min_elevation=min_elevation)
plt.title(f'{time} | TREx-RGB');
```

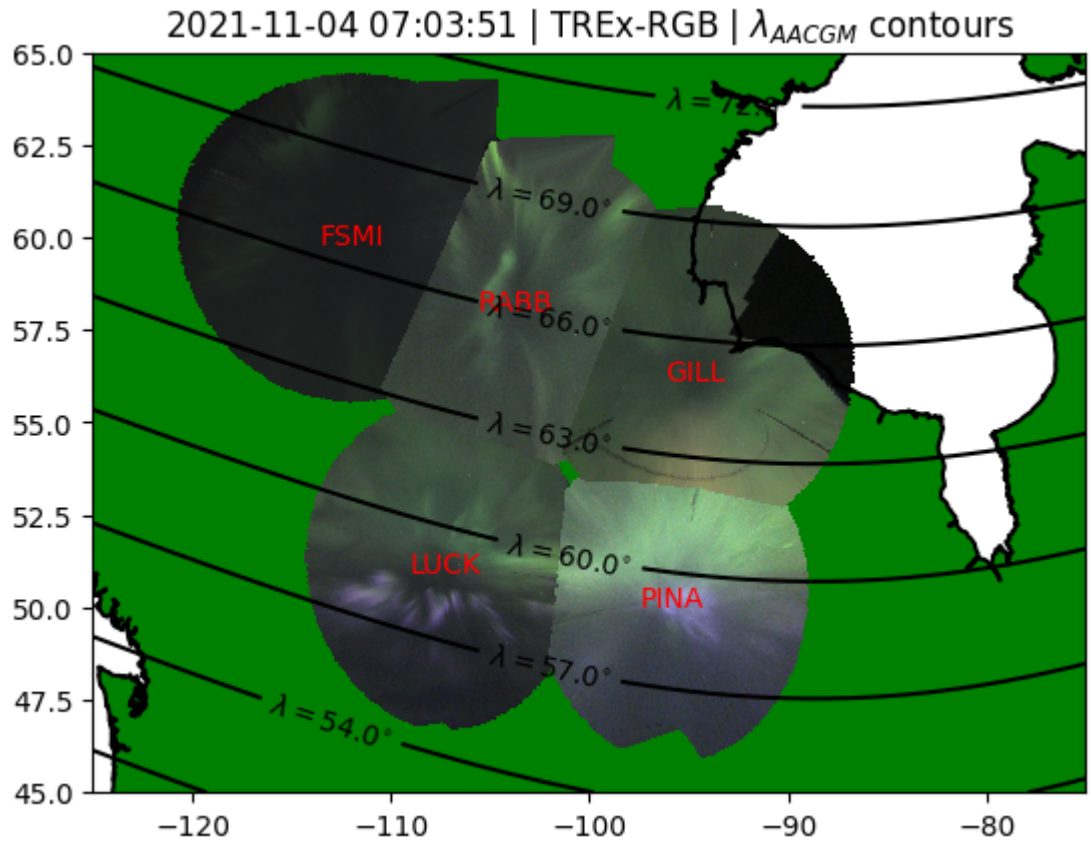


[6]:
```
lat_grid, lon_grid = np.meshgrid(np.linspace(*lat_bounds), np.linspace(*lon_bounds,␣
↪num=51))
# Need to pass flattened arrays since aacgmv2 does not work with n-D arrays.
aacgm_lat_grid, aacgm_lon_grid, _ = aacgmv2.wrapper.convert_latlon_arr(
    lat_grid.flatten(), lon_grid.flatten(), 110, time, method_code='G2A'
    )
aacgm_lat_grid = aacgm_lat_grid.reshape(lat_grid.shape)
aacgm_lon_grid = aacgm_lon_grid.reshape(lon_grid.shape)

ax = asilib.map.create_simple_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
asis.plot_map(ax=ax, overlap=False, min_elevation=min_elevation)
cs = plt.contour(lon_grid, lat_grid, aacgm_lat_grid, colors='k')
```

(continues on next page)

(continued from previous page)

```
ax.clabel(cs, inline=True, fontsize=10, fmt=lambda x: f'$\lambda = {{{x}}}^{{\circ}}$')
plt.title(f'{time} | TREx-RGB | $\lambda_{{AACGM}}$ contours');
```

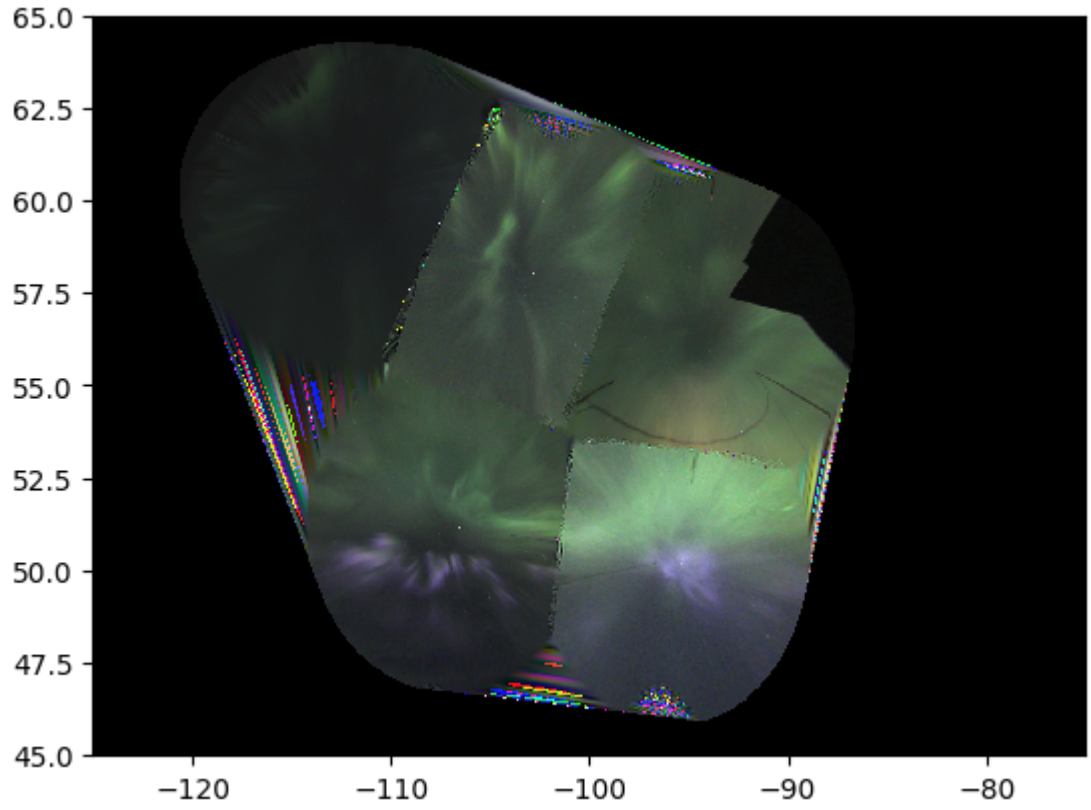


Now lets try to put the mosaic onto a custom grid.

[7]:
```
lat_lon_points, intensities = asis.get_points(min_elevation=min_elevation)
lat_grid2, lon_grid2 = np.meshgrid(np.linspace(*lat_bounds, num=1000), np.linspace(*lon_
↪bounds, num=1001))
rgb_grid = scipy.interpolate.griddata(lat_lon_points, intensities, (lat_grid2, lon_
↪grid2), method='cubic')
g_grid = scipy.interpolate.griddata(lat_lon_points, intensities[:, 1], (lat_grid2, lon_
↪grid2), method='cubic')
b_grid = scipy.interpolate.griddata(lat_lon_points, intensities[:, 2], (lat_grid2, lon_
↪grid2), method='cubic')
```

[8]:
```
plt.pcolormesh(lon_grid2, lat_grid2, rgb_grid/255)  # Need to divide by 255 to indicate␣
↪RGB channels to pcolormesh.
```

[8]:
```
<matplotlib.collections.QuadMesh at 0x2aa3e982650>
```

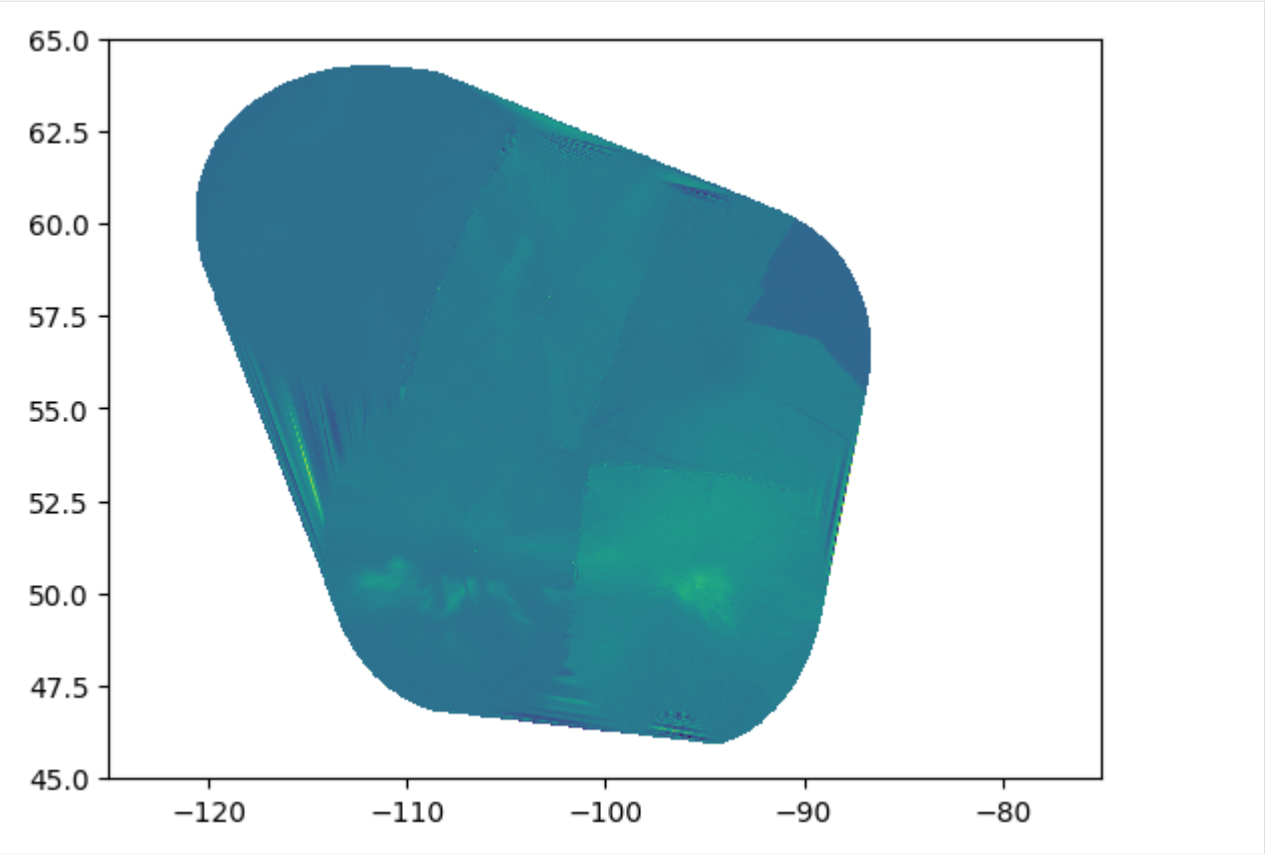


Looks good enough for a 100x101 grid! Now, how about the individual colors and green-blue ratio?

First, the blue grid:

[9]:
```
plt.pcolormesh(lon_grid2, lat_grid2, b_grid)
```

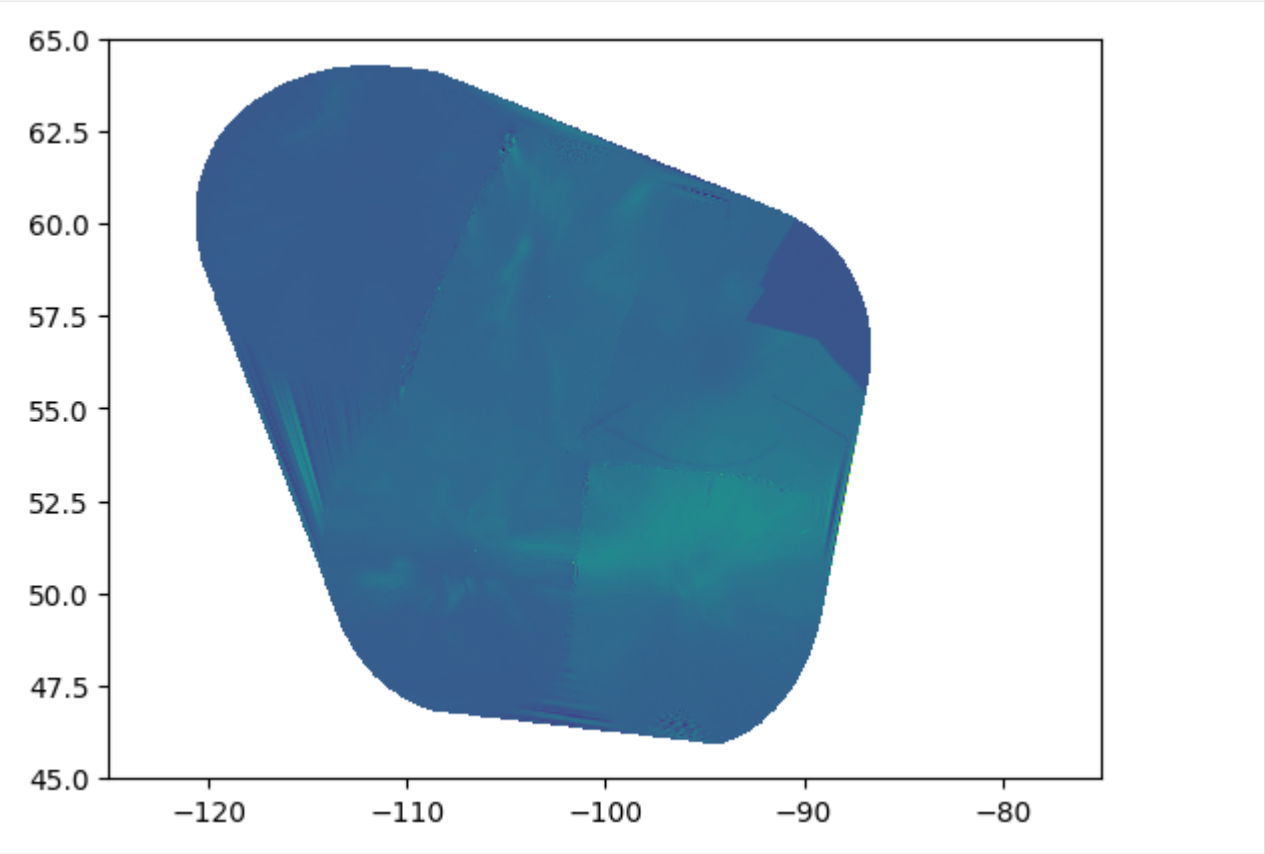
[9]:
```
<matplotlib.collections.QuadMesh at 0x2aa3e9f3350>
```



And the green grid:

[10]:
```
plt.pcolormesh(lon_grid2, lat_grid2, g_grid)
```

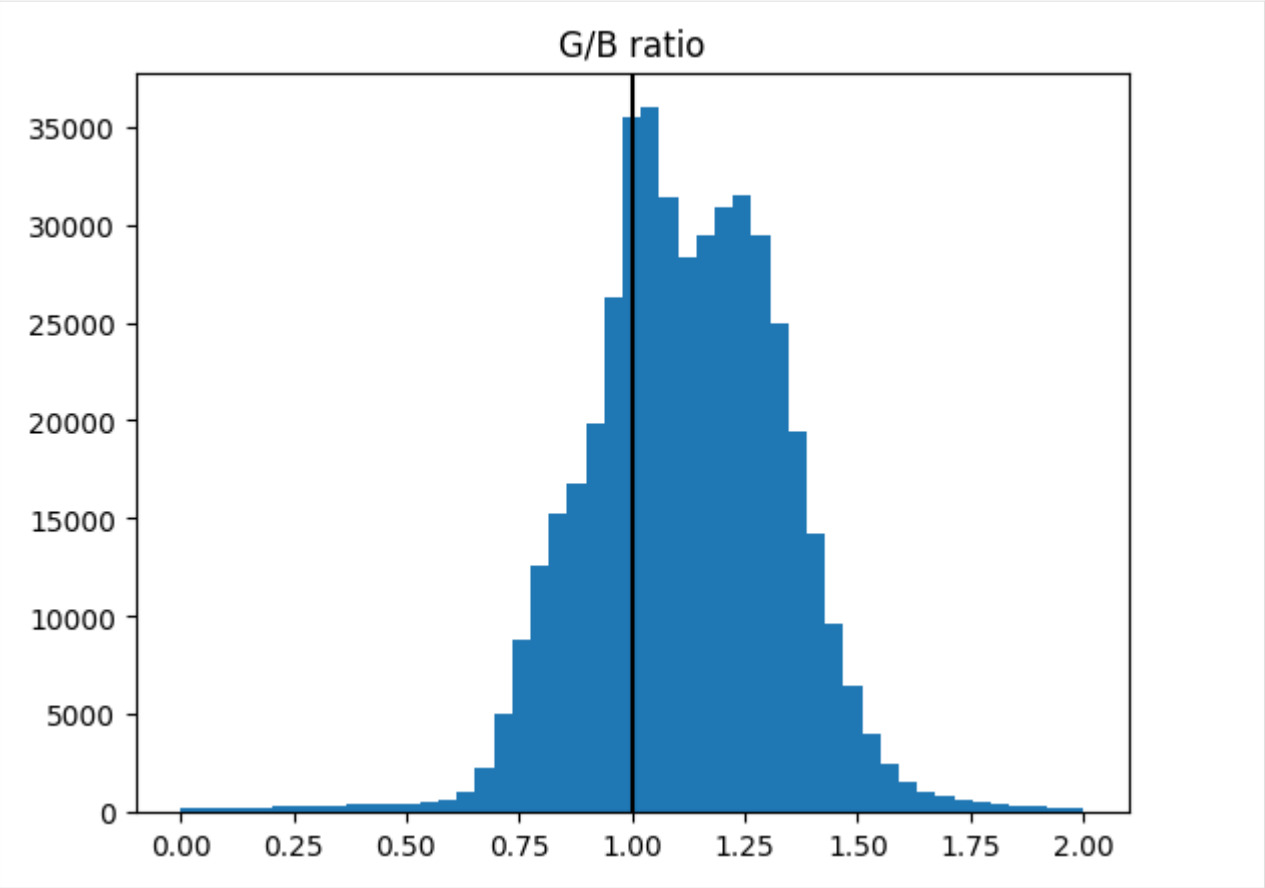
[10]:
```
<matplotlib.collections.QuadMesh at 0x2aa3ea5c610>
```



And the ratio:

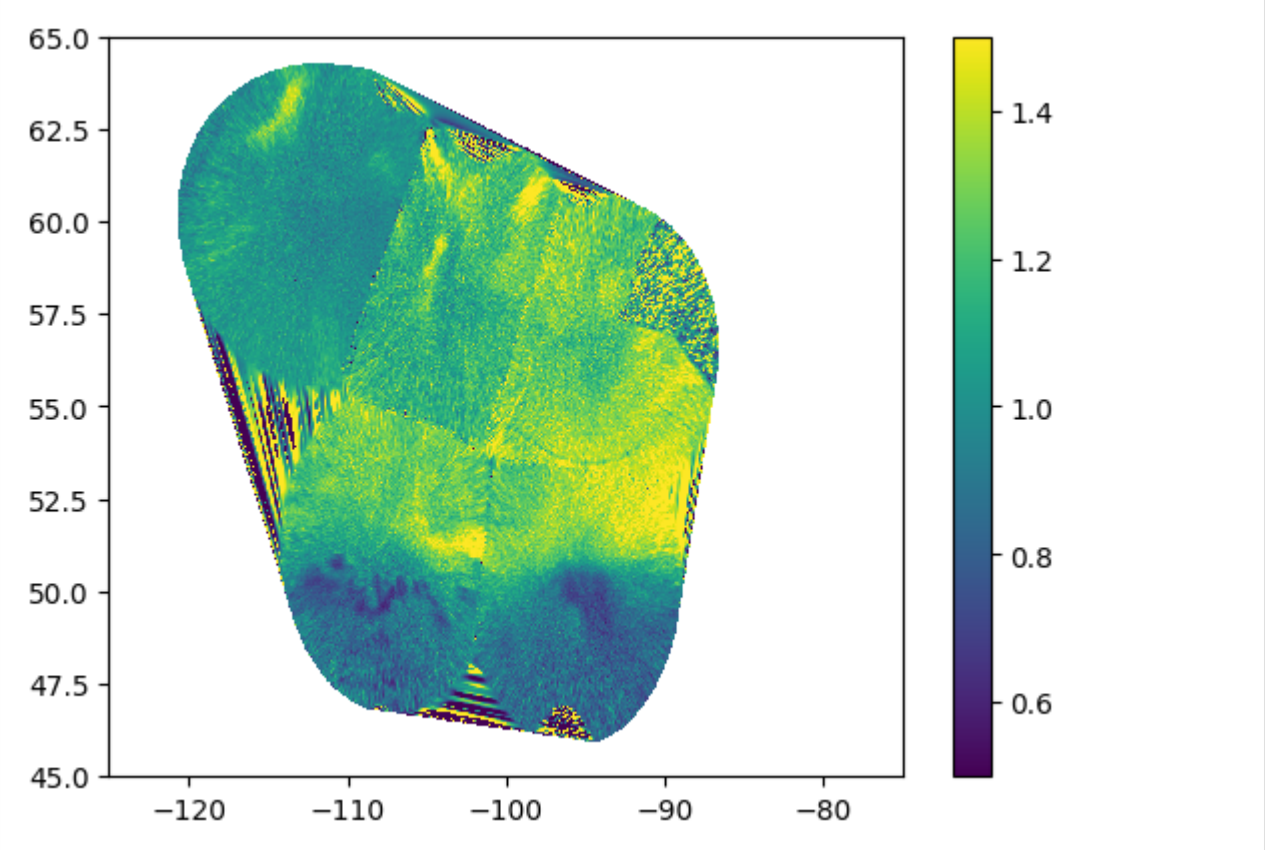
[11]:
```
plt.hist((g_grid/b_grid).flatten(), bins=np.linspace(0, 2));
plt.title('G/B ratio')
plt.axvline(1, c='k')
```

[11]:
```
<matplotlib.lines.Line2D at 0x2aa3a688690>
```



[12]:
```
plt.pcolormesh(lon_grid2, lat_grid2, g_grid/b_grid, vmin=0.5, vmax=1.5)
plt.colorbar()
```

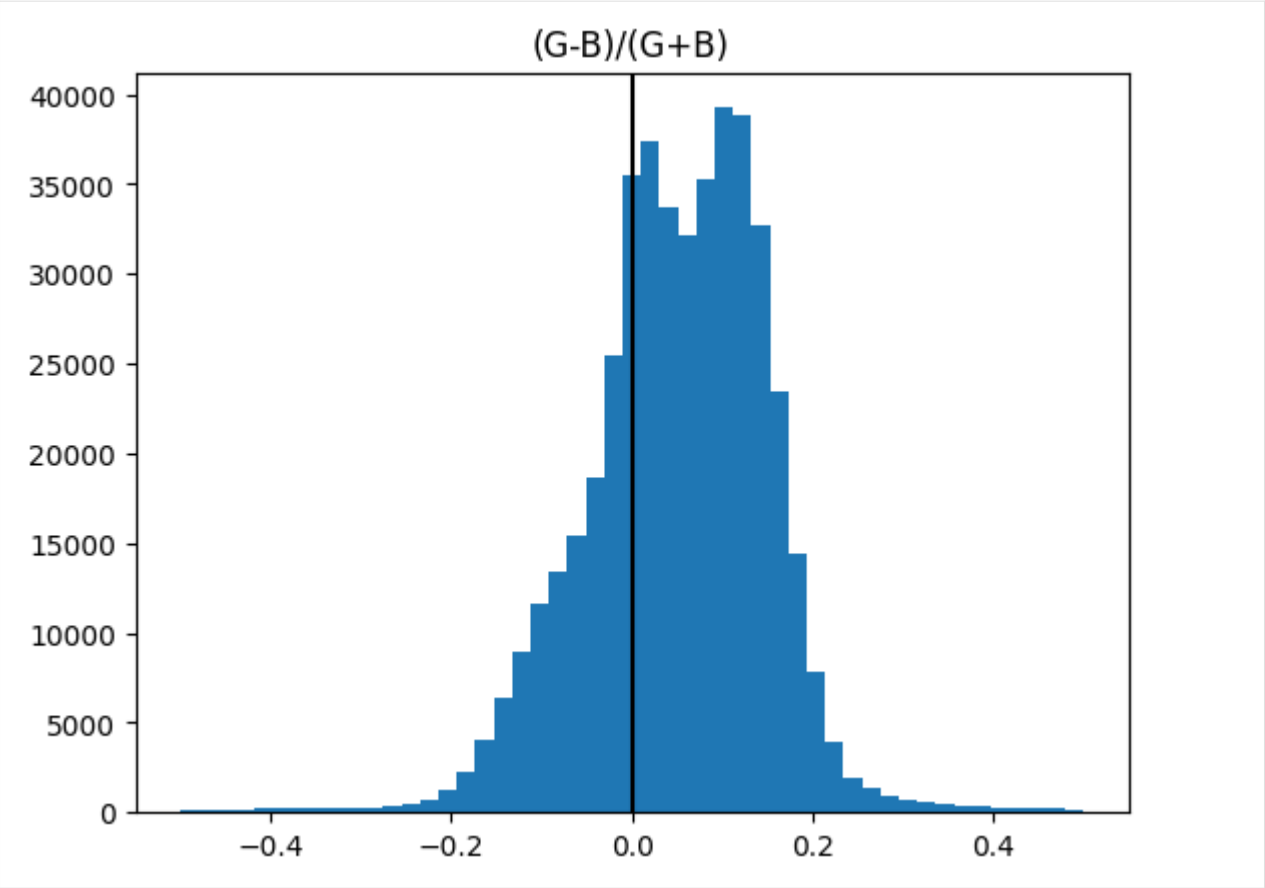
[12]:
```
<matplotlib.colorbar.Colorbar at 0x2aa3e9d26d0>
```



Normalized ratio

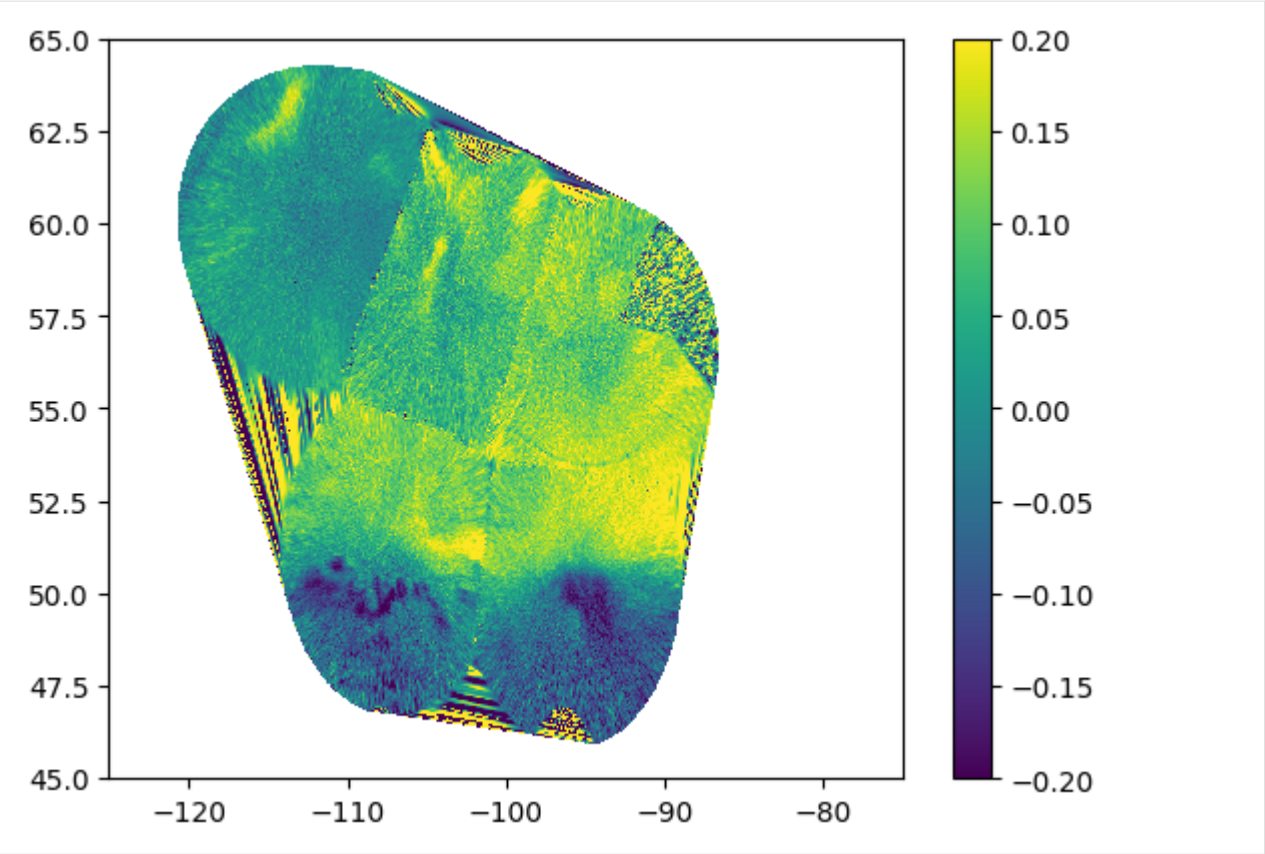
[13]:
```
plt.hist(((g_grid-b_grid)/(g_grid+b_grid)).flatten(), bins=np.linspace(-0.5, 0.5));
plt.title('(G-B)/(G+B)')
plt.axvline(0, c='k')
```

[13]:
```
<matplotlib.lines.Line2D at 0x2aa3a1cb890>
```



[14]:
```
plt.pcolormesh(lon_grid2, lat_grid2, (g_grid-b_grid)/(g_grid+b_grid), vmin=-0.2, vmax=0.
↪2)
plt.colorbar()
```

[14]:
```
<matplotlib.colorbar.Colorbar at 0x2aa3a7526d0>
```



1.3.3 Conjunctions

In this advanced tutorial we will combine a low Earth orbiting satellite ephemeris with ASIs, to calculate the mean auroral intensity at the satellite footprint as a function of time.

The mean auroral intensity is calculated two ways: 1) the nearest pixel, and 2) a (20x20) km area at 110 km emission altitude.

[2]:
```
from datetime import datetime, timedelta
from IPython.display import Video
import numpy as np

import matplotlib.pyplot as plt
import asilib
import asilib.asi
import asilib.map

plt.style.use('dark_background')

print(f'asilib version: {asilib.__version__}')
```

```
asilib version: 0.20.3
```

[3]:
```
location_code = 'RANK'
alt=110
```

(continues on next page)

(continued from previous page)

```
time_range = (datetime(2017, 9, 15, 2, 32, 0), datetime(2017, 9, 15, 2, 35, 0))
asi = asilib.asi.themis(location_code, time_range=time_range, alt=alt)
```

```
Downloaded 20170915_0232_rank_themis12_full.pgm.gz.
Downloaded 20170915_0233_rank_themis12_full.pgm.gz.
Downloaded 20170915_0234_rank_themis12_full.pgm.gz.
Downloaded themis_skymap_rank_20061101-20071203_vXX.sav.
Downloaded themis_skymap_rank_20071212-%2B_vXX.sav.
Downloaded themis_skymap_rank_20071218-%2B_vXX.sav.
Downloaded themis_skymap_rank_20071207-20090428_vXX.sav.
Downloaded themis_skymap_rank_20090911-%2B_vXX.sav.
Downloaded themis_skymap_rank_20100909-%2B_vXX.sav.
Downloaded themis_skymap_rank_20110202-%2B_vXX.sav.
Downloaded themis_skymap_rank_20120225-%2B_vXX.sav.
Downloaded themis_skymap_rank_20130107-%2B_vXX.sav.
Downloaded themis_skymap_rank_20150825-%2B_vXX.sav.
Downloaded themis_skymap_rank_20150825-20171003_vXX.sav.
Downloaded themis_skymap_rank_20170915-%2B_v02.sav.
Downloaded themis_skymap_rank_20170915-%2B_vXX.sav.
Downloaded themis_skymap_rank_20171003-%2B_v02.sav.
Downloaded themis_skymap_rank_20171003-%2B_vXX.sav.
Downloaded themis_skymap_rank_20190206-%2B_v02.sav.
Downloaded themis_skymap_rank_20190206-%2B_vXX.sav.
Downloaded themis_skymap_rank_20190827-%2B_v02.sav.
Downloaded themis_skymap_rank_20200227-%2B_v02.sav.
Downloaded themis_skymap_rank_20200919-%2B_v02.sav.
Downloaded themis_skymap_rank_20210309-%2B_v02.sav.
Downloaded themis_skymap_rank_20210829-%2B_v02.sav.
Downloaded themis_skymap_rank_20220304-%2B_v02.sav.
```

Satellite footprint

Now we define an orbit path of a low Earth orbiting satellite (i.e. its footprint). This is a north-south satellite track oriented to the east of the THEMIS/RANK imager. In this context, `lla` stands for the `(latitude, longitude, altitude)` coordinates.

[4]:
```
n = int((time_range[1] - time_range[0]).total_seconds() / 3)  # 3 second cadence.
lats = np.linspace(asi.meta["lat"] + 5, asi.meta["lat"] - 5, n)
lons = (asi.meta["lon"] - 0.5) * np.ones(n)
alts = alt * np.ones(n)
sat_lla = np.array([lats, lons, alts]).T
sat_time = asi.data.time
```

Create an `asilib.Conjunction()` object that handles mapping between the satellite and the imager. It takes in an `Imager` instance and arrays of the satellite times and LLA coordinates.

[5]:
```
conjunction_obj = asilib.Conjunction(asi, (sat_time, sat_lla))
```

Now, here are two steps that we are ignoring that you'll likely need to implement:

1. Map the satellite's LLA coordinates along the magnetic field line from the satellite altitude down to 110 km (or whatever you chose for the `alt` kwarg.) This is done via `Conjunction.lla_footprint()` that requires the

IRBEM library. IRBEM can be hard to install; in the future, I plan to change remove IRBEM in favor of geopack (or a similar package).

2. Normally the satellite LLA time stamps are not the same as the ASI. In that case you will need to call `Conjunction.interp_sat()` to interpolate the LLA coordinates to the ASI timestamps. Note: this method does not handle interpolation well across the anti-meridian (-180/180 degree longitude). If it detects that you're interpolating over it, it will issue a warning.

Nearest pixel intensity

[6]:
```
sat_azel, sat_azel_pixels = conjunction_obj.map_azel()
nearest_pixel_intensity = conjunction_obj.intensity(box=None)
```

Mean pixel intensity in a 20x20 km area.

The mean intensity is calculated with a masked array. It contains `np.nan` outside the 20x20 km area, and 1s inside.

[7]:
```
area_intensity = conjunction_obj.intensity(box=(10, 10))

# You don't need to calculate the area mask if you just need the intensity, but this is␣
↪useful if you
# want to animate and visualize the area
area_mask = conjunction_obj.equal_area(box=(10,10))
# Need to change masked NaNs to 0s so we can plot the rectangular area contours.
area_mask[np.where(np.isnan(area_mask))] = 0
```

The following code block contains many steps.

1. We create three subplots and initialize the animation generator.
2. We loop over each image.
    - In the first subplot plot the entire satellite footprint using the `sat_azel_pixels` array.
    - In the first subplot plot the instantaneous footprint.
    - In the first subplot plot the 20x20 km area contour.
    - In the second and third subplots plot the auroral intensity from a) the nearest pixel, and b) the 20x20 km area.
    - In the second and third subplots plot a vertical line at the current time.
    - Annotate the first subplot.

[8]:
```
fig, ax = plt.subplots(
    3, 1, figsize=(7, 10), gridspec_kw={'height_ratios': [4, 1, 1]}, constrained_
↪layout=True
)
ax[1].set(ylabel='ASI intensity\nnearest pixel [counts]')
ax[2].set(xlabel='Time', ylabel='ASI intensity\n10x10 km area [counts]')

gen = asi.animate_fisheye_gen(
    ax=ax[0], azel_contours=True, overwrite=True, cardinal_directions='news'
)
```

(continues on next page)

(continued from previous page)

```
for i, (time, image, _, im) in enumerate(gen):
    # Plot the entire satellite track, its current location, and a 20x20 km box
    # around its location.
    ax[0].plot(sat_azel_pixels[:, 0], sat_azel_pixels[:, 1], 'red')
    ax[0].scatter(sat_azel_pixels[i, 0], sat_azel_pixels[i, 1], c='red', marker='o',␣
↪s=50)
    ax[0].contour(area_mask[i, :, :], levels=[0.99], colors=['yellow'])

    if 'vline1' in locals():
        vline1.remove()
        vline2.remove()
        text_obj.remove()
    else:
        # Plot the ASI intensity along the satellite path
        ax[1].plot(sat_time, nearest_pixel_intensity)
        ax[2].plot(sat_time, area_intensity)
    vline1 = ax[1].axvline(time, c='b')
    vline2 = ax[2].axvline(time, c='b')

    # Annotate the location_code and satellite info in the top-left corner.
    location_code_str = (
        f'THEMIS/{location_code} '
        f'LLA=({asi.meta["lat"]:.2f}, '
        f'{asi.meta["lon"]:.2f}, {asi.meta["alt"]:.2f})'
    )
    satellite_str = f'Satellite LLA=({sat_lla[i, 0]:.2f}, {sat_lla[i, 1]:.2f}, {sat_
↪lla[i, 2]:.2f})'
    text_obj = ax[0].text(
        0,
        1,
        location_code_str + '\n' + satellite_str,
        va='top',
        transform=ax[0].transAxes,
        color='red',
    )
plt.close()
```

```
20170915_023200_023500_themis_rank_fisheye.mp4: |###################### | 97%
Animation saved to C:\Users\shumkms1\asilib-data\animations\20170915_023200_023500_
↪themis_rank_fisheye.mp4
```

[9]:
```
# When you run this, you should see the video below in your asilib-data/movies directory.
Video('https://github.com/mshumko/asilib/blob/main/docs/_static/v1/20170915_023200_
↪023500_themis_rank_fisheye.mp4?raw=true')
```

[9]:
```
<IPython.core.display.Video object>
```

1.3.4 Legacy Basics Tutorial

Overview

Welcome to aurora-asi-lib. This tutorial will guide you though the main functions in `aurora-asi-lib` (imported as `asilib`).

First off, we need to import the necessary packages.

[3]:
```
from datetime import datetime, timedelta
from IPython.display import Video
import numpy as np

import matplotlib.pyplot as plt
import asilib

plt.style.use('dark_background')

print(f'asilib version: {asilib.__version__}')
```

```
asilib version: 0.12.1
```

First of all, you should know where the data and movies are saved to. This information is in `asilib.config` and can be changed with `python3 -m asilib config` to configure asilib.

[4]:
```
asilib.config
```

[4]:
```
{'ASILIB_DIR': WindowsPath('C:/Users/mshumko/Documents/research/aurora-asi-lib/asilib'),
 'ASI_DATA_DIR': WindowsPath('C:/Users/mshumko/asilib-data')}
```

As you can guess, `asilib.config['ASILIB_DIR']` is the directory where this code resides, `asilib.config['ASI_DATA_DIR']` is the directory where the data is saved to.

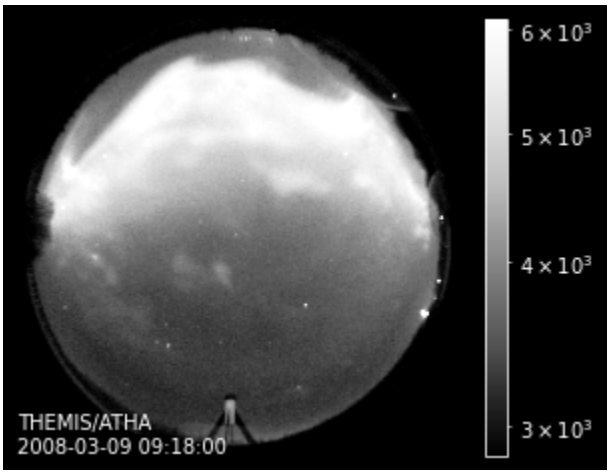
Plot a single image

Since we sold you on easily downloading, loading, plotting and analyzing auroral images with `asilib`, lets begin with a plot of an omega band studied in:

Liu, J., Lyons, L. R., Archer, W. E., Gallardo-Lacourt, B., Nishimura, Y., Zou, Y., … Weygand, J. M. (2018). Flow shears at the poleward boundary of omega bands observed during conjunctions of Swarm and THEMIS ASI. Geophysical Research Letters, 45, 1218– 1227. https://doi.org/10.1002/2017GL076485

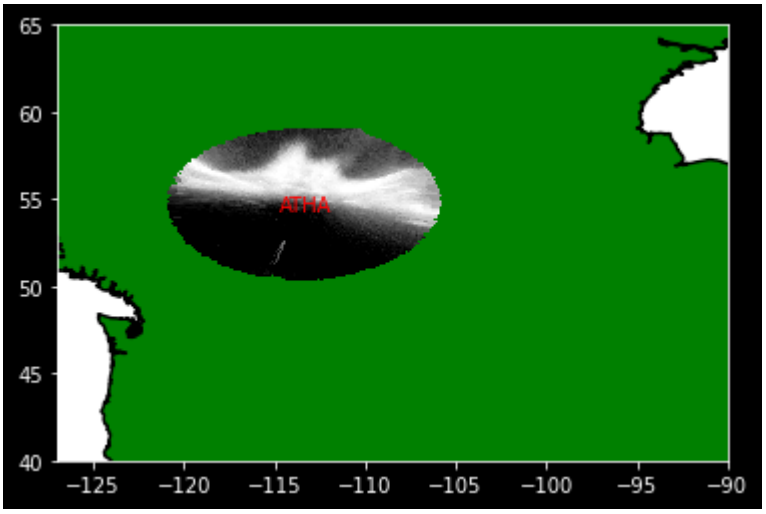
We can plot a fisheye lens image using `asilib.plot_fisheye`.

[5]:
```
asi_array_code = 'THEMIS'
location_code = 'ATHA'
time = datetime(2008, 3, 9, 9, 18, 0)  # You can supply a datetime object or a ISO-
↪formatted time string.

image_time, image, ax, im = asilib.plot_fisheye(asi_array_code, location_code, time,
                    color_norm='log', color_map='auto')
plt.colorbar(im)
ax.axis('off');
```



It is also easy to map the ASI fisheye lens image to a geographic map using `asilib.plot_map`. In the code box below, the first line creates a subplot with a geographic map centered on Athabasca. If latitude or longitude bounds are not provided, `asilib.plot_map` defaults to a map of North America.

[6]:
```
ax = asilib.make_map(lon_bounds=(-127, -90), lat_bounds=(40, 65))

map_alt_km = 110
asilib.plot_map(asi_array_code, location_code, time, map_alt_km, ax=ax);
```



Notice that you did not need to explicitly download or load the data—`asilib` takes care of that for you. If you need to explicitly download data, `asilib` comes with four functions to download the image and skymap files.

Now what if you need to analyze the image? `asilib.plot_fisheye` returns the time stamp of the image `image_time` and the 2-d image np.array `image`.

[7]:
```
image_time, image
```

[7]:
```
(datetime.datetime(2008, 3, 9, 9, 18, 0, 51000),
 array([[2546, 2537, 2572, ..., 2554, 2616, 2536],
        [2608, 2613, 2562, ..., 2620, 2582, 2582],
        [2550, 2526, 2588, ..., 2568, 2560, 2544],
        ...,
```

(continues on next page)

(continued from previous page)

```
        [2629, 2541, 2612, ..., 2553, 2546, 2525],
        [2569, 2568, 2510, ..., 2698, 2596, 2545],
        [2576, 2617, 2514, ..., 2602, 2577, 2502]], dtype=uint16))
```

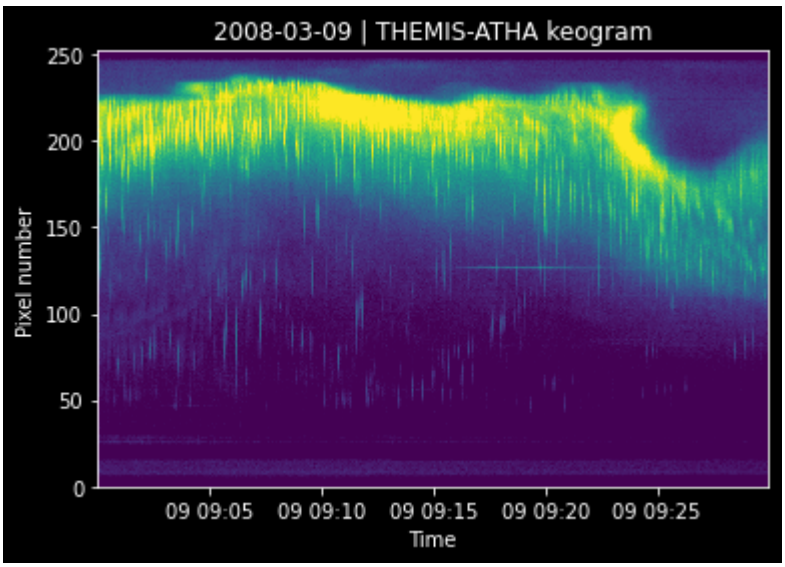
You can also just load that image by using `asilib.load_image()`.

[8]:
```
image_time, image = asilib.load_image(asi_array_code, location_code, time=time,␣
↪redownload=False)
```

Plot a Keogram

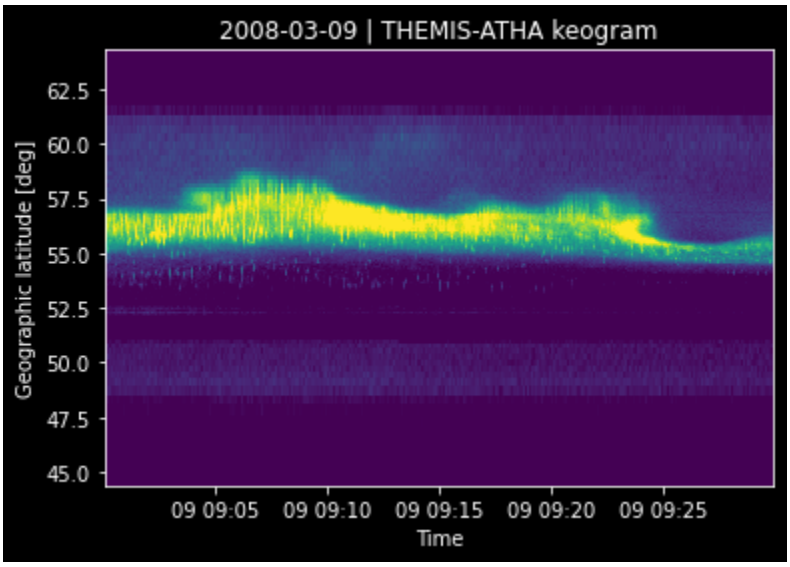
`asilib.plot_keogram` plots a keogram through the meridian, and using it is as simple as plotting a single image. If you don't specify a `map_alt`, the vertical axis will be pixel number. Note: `asilib.keogram` returns the keogram array that is plotted by `asilib.plot_keogram`.

[9]:
```
time_range = [datetime(2008, 3, 9, 9, 0, 0), datetime(2008, 3, 9, 9, 30, 0)]
asilib.plot_keogram(asi_array_code, location_code, time_range)
plt.xlabel('Time'); plt.ylabel('Pixel number');
```



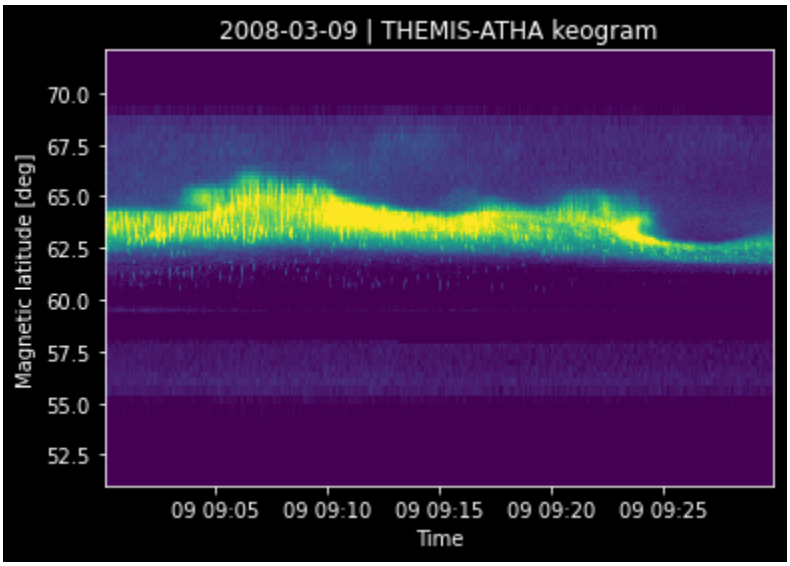
And if you specify a `map_alt` (in kilometers), the vertical axis will be geographic latitude.

[10]:
```
map_alt_km = 110
asilib.plot_keogram(asi_array_code, location_code, time_range, map_alt=map_alt_km)
plt.xlabel('Time'); plt.ylabel('Geographic latitude [deg]');
```



And lastly, if you specify a `map_alt` and set `aacgm=True`, the keogram's vertical axis will be magnetic latitude estimated using the aacgmv2 Python package that implements the Altitude-adjusted corrected geomagnetic coordinates defined in Shepherd 2014.

[11]:
```
map_alt_km = 110
asilib.plot_keogram(asi_array_code, location_code, time_range, map_alt=map_alt_km,␣
↪aacgm=True)
plt.xlabel('Time'); plt.ylabel('Magnetic latitude [deg]');
```



Simple enough? You can choose any other altitude from the `FULL_MAP_ALTITUDE` key in the skymap calibration data (described in the following section). If you pick a wrong altitude, `asilib` will give you a helpful error.

[12]:
```
try:
    asilib.plot_keogram(asi_array_code, location_code, time_range, map_alt=100)
except AssertionError as err:
    print('AssetionError:', err)
```

```
AssetionError: 100 is not in the skymap altitudes: [ 90. 110. 150.]
```

These aurora-emission altitudes are in the THEMIS ASI skymap files; They are different for the REGO ASIs.

Skymap calibration data

You may wonder how the image's pixel values were mapped to geographic latitude—this is where the skymap files come in. `asilib` also makes this easy and you only need to give it the ASI array code, location code, and the time (so the correct skymap file is loaded).

[13]:
```
skymap = asilib.load_skymap(asi_array_code, location_code, time)
skymap.keys()
```

[13]:
```
dict_keys(['GENERATION_INFO', 'SITE_UID', 'IMAGER_UID', 'PROJECT_UID', 'IMAGER_UNIX_TIME
↪', 'SITE_MAP_LATITUDE', 'SITE_MAP_LONGITUDE', 'SITE_MAP_ALTITUDE', 'FULL_ROW', 'FULL_
↪COLUMN', 'FULL_IGNORE', 'FULL_SUBTRACT', 'FULL_MULTIPLY', 'FULL_ELEVATION', 'FULL_
↪AZIMUTH', 'FULL_MAP_ALTITUDE', 'FULL_MAP_LATITUDE', 'FULL_MAP_LONGITUDE', 'FULL_BIN',
↪'BIN_ROW', 'BIN_COLUMN', 'BIN_ELEVATION', 'BIN_AZIMUTH', 'BIN_MAP_ALTITUDE', 'BIN_MAP_
↪LATITUDE', 'BIN_MAP_LONGITUDE', 'SKYMAP_PATH'])
```

The skymap data that `asilib` uses for keograms is the `FULL_MAP_LATITUDE`. Also, to confirm that you loaded the correct skymap file, the `skymap` dictionary contains a `SKYMAP_PATH` key that points to the local skymap file.

[14]:
```
skymap['SKYMAP_PATH']
```

[14]:
```
WindowsPath('C:/Users/mshumko/asilib-data/themis/skymap/atha/themis_skymap_atha_20070301-
↪20090522_vXX.sav')
```

The possible skymap mapping altitudes (in meters) that you can use in `asilib.plot_keogram` and `asilib.plot_map`.

[15]:
```
skymap['FULL_MAP_ALTITUDE']
```

[15]:
```
array([ 90000., 110000., 150000.], dtype=float32)
```

Mapping multiple all-sky images

You can plot one image from multiple ASI locations using a for-loop. In the following example, we will replicate Fig. 2b from:

Donovan, E., Liu, W., Liang, J., Spanswick, E., Voronkov, I., Connors, M., … & Rae, I. J. (2008). Simultaneous THEMIS in situ and auroral observations of a small substorm. Geophysical Research Letters, 35(17).

[16]:
```
# ASI parameters
time = datetime(2007, 3, 13, 5, 8, 45)
asi_array_code = 'THEMIS'
location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
map_alt = 110
min_elevation = 2

fig, bx = plt.subplots(figsize=(8, 5))
bx = asilib.make_map(lon_bounds=(-160, -52), lat_bounds=(40, 82), ax=bx)
```

(continues on next page)

(continued from previous page)

```
for location_code in location_codes:
    asilib.plot_map(asi_array_code, location_code, time, map_alt, ax=bx,
        min_elevation=min_elevation)

bx.set_title('Donovan et al. 2008 | First breakup of an auroral arc');
```

```
Made directory at C:\Users\mshumko\asilib-data\themis\skymap\fsim
Downloading themis_skymap_fsim_20070126_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20070201-20081229_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20090123-20090503_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20090912_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20110202_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20120226_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20130103_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20150915_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20170906_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20190227_vXX.sav: |####################| 100%
Downloading themis_skymap_fsim_20190303_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20191031_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20200313_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20201111_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20210307_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20211205_v02.sav: |####################| 100%
Downloading themis_skymap_fsim_20220304_v02.sav: |####################| 100%
Made directory at C:\Users\mshumko\asilib-data\themis\skymap\tpas
Downloading themis_skymap_tpas_20070214_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20070217_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20080116-20080118_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20070803-20090514_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20090914_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20100909_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20110207_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20120226_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20130105_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20151206_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20170926_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20190204_vXX.sav: |####################| 100%
Downloading themis_skymap_tpas_20200302_v02.sav: |####################| 100%
Downloading themis_skymap_tpas_20200918_v02.sav: |####################| 100%
Downloading themis_skymap_tpas_20210304_v02.sav: |####################| 100%
Downloading themis_skymap_tpas_20220304_v02.sav: |####################| 100%
```

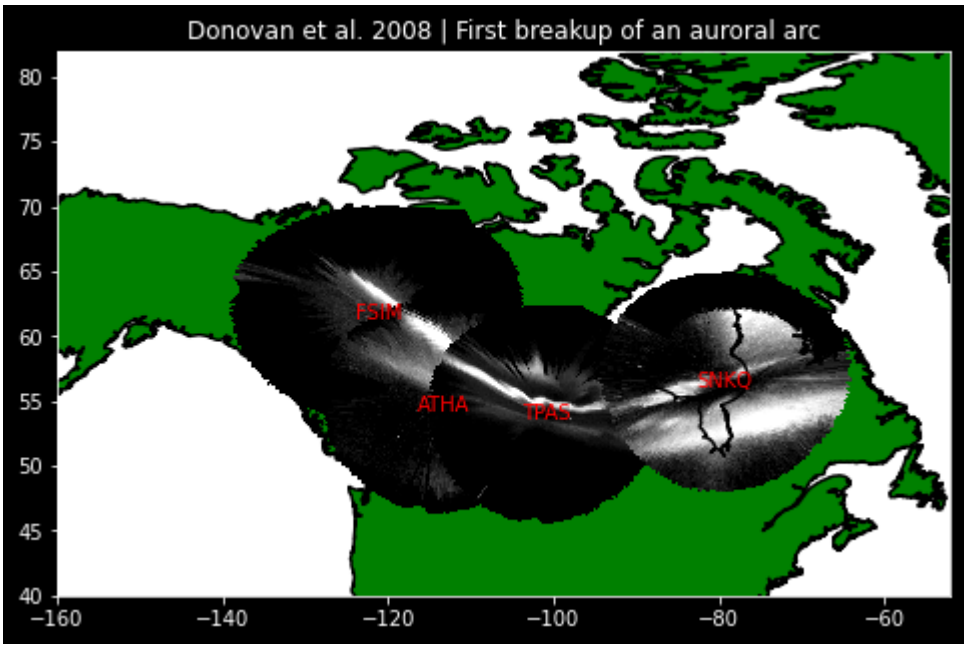
```
c:\Users\mshumko\.conda\envs\asilib_test\lib\site-packages\scipy\io\idl.py:281:␣
↪UserWarning: Not able to verify number of bytes from header
  warnings.warn("Not able to verify number of bytes from header")
```

```
Made directory at C:\Users\mshumko\asilib-data\themis\skymap\snkq
Downloading themis_skymap_snkq_20070306_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20070307_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20070313_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20071219_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20070919-20090521_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20090913_vXX.sav: |####################| 100%
```

(continues on next page)

(continued from previous page)

```
Downloading themis_skymap_snkq_20100910_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20110207_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20130105_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20150516_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20170926_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20180919_vXX.sav: |####################| 100%
Downloading themis_skymap_snkq_20190204_v02.sav: |####################| 100%
Downloading themis_skymap_snkq_20200301_v02.sav: |####################| 100%
Downloading themis_skymap_snkq_20200920_v02.sav: |####################| 100%
Downloading themis_skymap_snkq_20210304_v02.sav: |####################| 100%
Downloading themis_skymap_snkq_20210901_v02.sav: |####################| 100%
Downloading themis_skymap_snkq_20220304_v02.sav: |####################| 100%
```



Working with multiple images

The `asilib.load_image` function can also load (and automatically download) image data given a time duration specified by `time_range`. In this section, we will plot a montage of fisheye lens images that show the equatorward movement of a STEVE aurora studied in

Gallardo-Lacourt, B., Nishimura, Y., Donovan, E., Gillies, D. M., Perry, G. W., Archer, W. E., et al. (2018). A statistical analysis of STEVE. Journal of Geophysical Research: Space Physics, 123, 9893– 9905. https://doi.org/10.1029/2018JA025368

[17]:
```
asi_array_code = 'REGO'
location_code = 'LUCK'
time_range = [datetime(2017, 9, 27, 7, 15), datetime(2017, 9, 27, 8, 15)]

image_times, images = asilib.load_image(asi_array_code, location_code, time_range=time_
↪range, redownload=False)
```

[18]:
```
image_times.shape
```

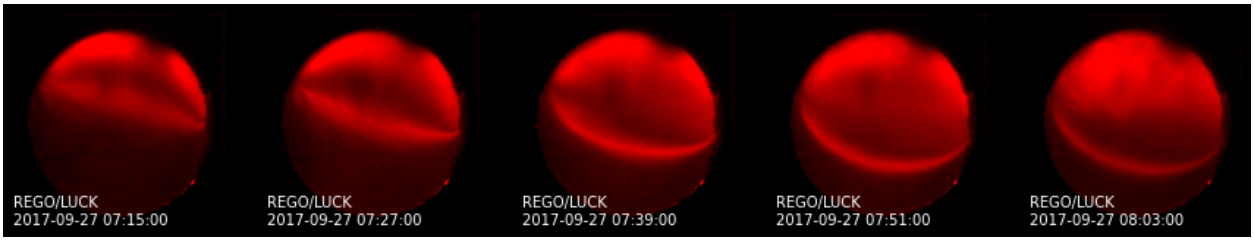
[18]:
```
(1200,)
```

[19]:
```
images.shape  # First axis is time
```

[19]:
```
(1200, 512, 512)
```

Now lets pick 5 images from that time interval and show the movement of STEVE.

[20]:
```
n_plots = 5

delta_time_s = int((time_range[1]-time_range[0]).total_seconds()/n_plots)
montage_times = [time_range[0]+timedelta(seconds=i*delta_time_s) for i in range(n_plots)]

fig, cx = plt.subplots(1, n_plots, figsize=(15, 8))

for montage_time, cx_i in zip(montage_times, cx):
    asilib.plot_fisheye(asi_array_code, location_code, montage_time, ax=cx_i)
    cx_i.axis('off')
```



Make a movie

Let's now make a simple fisheye lens movie of a substorm using `asilib.animate_fisheye`.

[21]:
```
asi_array_code = 'THEMIS'
location_code = 'FSMI'
time_range = [datetime(2015, 3, 26, 6, 7, 0), datetime(2015, 3, 26, 6, 30, 0)]

# loglevel is to suppress the verbose ffmpeg output.
asilib.animate_fisheye(asi_array_code, location_code, time_range, overwrite=True, ffmpeg_
↪output_params={'loglevel':'quiet'})
plt.close()  # To show a clean output in this tutorial---it is often unnecessary.

# When you run this, you should see the video below in your asilib-data/movies directory.
Video('https://github.com/mshumko/aurora-asi-lib/blob/
↪e6147ff1a3309c602c1aa48711ebc8a90a7863e1/docs/_static/'
    '20150326_060700_062957_themis_fsmi.mp4?raw=true')
```

```
Created a C:\Users\mshumko\asilib-data\animations\images\20150326_060700_themis_fsmi_
↪fisheye directory
```

[21]:
```
<IPython.core.display.Video object>
```

Animating images into the ionosphere is also straightforward.

[22]:
```
asi_array_code = 'THEMIS'
location_code = 'FSMI'
time_range = [datetime(2015, 3, 26, 6, 7, 0), datetime(2015, 3, 26, 6, 30, 0)]

# We need the skymap only to center the map on the projected image.
skymap = asilib.load_skymap(asi_array_code, location_code, time_range[0])
lat_bounds = (skymap['SITE_MAP_LATITUDE']-7, skymap['SITE_MAP_LATITUDE']+7)
lon_bounds = (skymap['SITE_MAP_LONGITUDE']-10, skymap['SITE_MAP_LONGITUDE']+10)

ax = asilib.make_map(lon_bounds=lon_bounds, lat_bounds=lat_bounds)
plt.subplots_adjust(top=0.99, bottom=0.05, left=0.05, right=0.99)
asilib.animate_map(asi_array_code, location_code, time_range, 110, overwrite=True, ax=ax,
    ffmpeg_output_params={'loglevel':'quiet'})

plt.close()  # To show a clean output in this tutorial---it is often unnecessary.

# When you run this, you should see the video below in your asilib-data/movies directory.
Video('https://github.com/mshumko/aurora-asi-lib/blob/main/docs/_static/20150326_060700_
↪062957_themis_fsmi_map.mp4?raw=true')
```

```
Created a C:\Users\mshumko\asilib-data\animations\images\20150326_060700_themis_fsmi_map␣
↪directory
```

[22]:
```
<IPython.core.display.Video object>
```

In the next section, we make a more sophisticated movie using `asilib.animate_fisheye_generator`.

Satellite conjunction

As a last example, here we show how `asilib` can be used to make a conjunction movie. Be forewarned, this example is long. We will use ASI data from the THEMIS-RANK imager.

[23]:
```
asi_array_code = 'THEMIS'
location_code = 'RANK'
time_range = (datetime(2017, 9, 15, 2, 32, 0), datetime(2017, 9, 15, 2, 35, 0))
```

Now we define an orbit path of a low Earth orbiting satellite. In this context, `lla` stands for the `(latitude, longitude, altitude)` coordinates. That we place near the THEMIS-RANK imager, hence we use load_skymap to extract the imager's location in Canada.

[24]:
```
# Load the skymap calibration data. This is necessary to create the satellite track␣
↪overhead.
skymap_dict = asilib.load_skymap(asi_array_code, location_code, time_range[0])

# Create the fake satellite track coordinates: latitude, longitude, altitude (LLA).
# This is a north-south satellite track oriented to the east of the THEMIS/RANK
# imager.
n = int((time_range[1] - time_range[0]).total_seconds() / 3)  # 3 second cadence.
lats = np.linspace(skymap_dict["SITE_MAP_LATITUDE"] + 5, skymap_dict["SITE_MAP_LATITUDE
↪"] - 5, n)
lons = (skymap_dict["SITE_MAP_LONGITUDE"] - 0.5) * np.ones(n)
alts = 110 * np.ones(n)
lla = np.array([lats, lons, alts]).T
```

```
Made directory at C:\Users\mshumko\asilib-data\themis\skymap\rank
Downloading themis_skymap_rank_20061101-20071203_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20071212_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20071218_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20071207-20090428_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20090911_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20100909_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20110202_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20120225_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20130107_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20150825_vXX.sav: |####################| 100%
Downloading themis_skymap_rank_20170915_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20171003_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20190206_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20190827_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20200227_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20200919_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20210309_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20210829_v02.sav: |####################| 100%
Downloading themis_skymap_rank_20220304_v02.sav: |####################| 100%
```

With the satellite track, we first need to map its location to the imager's elevation and azimuth (azel) using `asilib.lla2azel`. Note: if you need to find the satellite's footprint, you can use `asilib.lla2footprint` to map along the magnetic field line before you pass the footprint LLA into `asilib.lla2azel`. However, `asilib.lla2footprint` requires the IRBEM library.

[25]:
```
sat_azel, sat_azel_pixels = asilib.lla2azel(asi_array_code, location_code, time_range[0],
↪ lla)
```

Here we create the conjunction movie. We can't easily split up the steps into notebook cells, but I've add STEPS 1-4 for the main steps:

- STEP 1: create the subplots. ax[0] will plot the ASI images, while ax[1] will plot the ASI intensity time series along the satellite path. We initialize the `asilib.animate_fisheye_generator` function which allows us to manipulate the individual images before they are stitched together into a movie.
- STEP 2: We retrieve the imager time stamps and images using `movie_generator` (technically it is a `coroutine`). We need this to calculate the mean ASI intensity along the spacecraft track.
- STEP 3: We use `asilib.equal_area` to calculate the pixels inside a 20x20 km area at 110 km around the satellite's footprint (inferred from LLA). `asilib.equal_area` returns a mask: 1s inside the area, and NaNs outside it. We use it with `image_data.images` and `np.nanmean` to calculate the mean ASI intensity in all frames simultaneously.
- STEP 4: Loop over every image in `time_range`. `asilib.animate_fisheye_generator` will plot the current frame in `ax[0]` on top of which we superpose the full satellite track and its current location. We also use `plt.contour` to plot the 20x20 km area at 110 km in yellow. Lastly, in `ax[1]` we plot the mean ASI time series in the area box around the satelite's footprint, as well as a vertical line to guide your eye to the corresponding image in `ax[0]`.

[26]:
```
# STEP 1
fig, ax = plt.subplots(
    2, 1, figsize=(7, 10), gridspec_kw={'height_ratios': [4, 1]}, constrained_layout=True
)
movie_generator = asilib.animate_fisheye_generator(
```

(continues on next page)

(continued from previous page)

```
    asi_array_code, location_code, time_range, azel_contours=True, overwrite=True,␣
↪ax=ax[0],
    ffmpeg_output_params={'loglevel':'quiet'}
)

# STEP 2
# Use the generator to get the images and time stamps to estimate mean the ASI
# brightness along the satellite path and in a (10x10 km) box.
image_data = movie_generator.send('data')

# STEP 3
# Calculate what pixels are in a box_km around the satellite, and convolve it
# with the images to pick out the ASI intensity in that box.
area_box_mask = asilib.equal_area(
    asi_array_code, location_code, time_range[0], lla, box_km=(20, 20)
)
asi_brightness = np.nanmean(image_data.images * area_box_mask, axis=(1, 2))
area_box_mask[np.isnan(area_box_mask)] = 0  # To play nice with plt.contour()

# STEP 4
for i, (time, image, _, im) in enumerate(movie_generator):
    # Note that because we are drawing moving data: ASI image in ax[0] and
    # the ASI time series + a vertical bar at the image time in ax[1], we need
    # to redraw everything at every iteration.

    # Clear ax[1] (ax[0] cleared by asilib.animate_fisheye_generator())
    ax[1].clear()
    # Plot the entire satellite track
    ax[0].plot(sat_azel_pixels[:, 0], sat_azel_pixels[:, 1], 'red')
    ax[0].contour(area_box_mask[i, :, :], levels=[0.99], colors=['yellow'])
    # Plot the current satellite position.
    ax[0].scatter(sat_azel_pixels[i, 0], sat_azel_pixels[i, 1], c='red', marker='o',␣
↪s=50)

    # Plot the time series of the mean ASI intensity along the satellite path
    ax[1].plot(image_data.time, asi_brightness)
    ax[1].axvline(time, c='b')  # At the current image time.

    # Annotate the location_code and satellite info in the top-left corner.
    location_code_str = (
        f'{asi_array_code}/{location_code} '
        f'LLA=({skymap_dict["SITE_MAP_LATITUDE"]:.2f}, '
        f'{skymap_dict["SITE_MAP_LONGITUDE"]:.2f}, {skymap_dict["SITE_MAP_ALTITUDE"]:.2f}
↪)'
    )
    satellite_str = f'Satellite LLA=({lla[i, 0]:.2f}, {lla[i, 1]:.2f}, {lla[i, 2]:.2f})'
    ax[0].text(
        0,
        1,
        location_code_str + '\n' + satellite_str,
        va='top',
        color='red',
```

(continues on next page)

(continued from previous page)

```
    )
    ax[1].set(xlabel='Time', ylabel='Mean ASI intensity [counts]')

print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "movies"}')
plt.close()
```

```
Created a C:\Users\mshumko\asilib-data\animations\images\20170915_023200_themis_rank_
↪fisheye directory
Movie saved in C:\Users\mshumko\asilib-data\movies
```

[27]:
```
# When you run this, you should see the video below in your asilib-data/movies directory.
Video('https://github.com/mshumko/aurora-asi-lib/blob/main/docs/_static/20170915_023200_
↪023457_themis_rank.mp4?raw=true')
```

[27]:
```
<IPython.core.display.Video object>
```

1.4 Imager API Reference

asilib saves all of the ASI image files, skymap calibration files, and movies to

```
asilib.config['ASI_DATA_DIR']
```

By default this directory is set to *~/asilib-data/*, but you can configure the paths using the prompt opened by

```
python3 -m asilib config
```

Note: The longitude units are converted from [0, 360] to [-180, 180] degrees in the skymap calibration files.

1.4.1 ASI arrays

Time History of Events and Macroscale Interactions during Substorms (THEMIS)

asilib.asi.**themis**(*location_code*, *time=None*, *time_range=None*, *alt=110*, *custom_alt=False*, *redownload=False*, *missing_ok=True*, *load_images=True*, *imager=<class 'asilib.imager.Imager'>*)

Create an Imager instance with the THEMIS ASI images and skymaps.

Parameters

- **location_code** (*str*) – The ASI's location code (four letters).
- **time** (*str or datetime.datetime*) – A time to look for the ASI data at. Either time or time_range must be specified (not both or neither).
- **time_range** (*list of str or datetime.datetime*) – A length 2 list of string-formatted times or datetimes to bracket the ASI data time interval.
- **alt** (*int*) – The reference skymap altitude, in kilometers.
- **redownload** (*bool*) – If True, will download the data from the internet, regardless of wether or not the data exists locally (useful if the data becomes corrupted).

- **custom_alt** (*bool*) – If True, asilib will calculate (lat, lon) skymaps assuming a spherical Earth. Otherwise, it will use the official skymaps (Courtesy of University of Calgary).

  Note: The spherical model of Earth's surface is less accurate than the oblate spheroid geometrical representation. Therefore, there will be a small difference between these and the official skymaps.

- **missing_ok** (*bool*) – Wether to allow missing data files inside time_range (after searching for them locally and online).
- **load_images** (*bool*) – Create an Imager object without images. This is useful if you need to calculate conjunctions and don't need to download or load unnecessary data.
- **imager** (asilib.Imager) – Controls what Imager instance to return, asilib.Imager by default. This parameter is useful if you need to subclass asilib.Imager.

Returns
A THEMIS ASI instance with the time stamps, images, skymaps, and metadata.

Return type
Imager()

asilib.asi.**themis_info**()

Returns a pd.DataFrame with the THEMIS ASI names and locations.

Returns
A table of THEMIS imager names and locations.

Return type
pd.DataFrame

asilib.asi.**themis_skymap**(*location_code*, *time*, *redownload*)

Load a THEMIS ASI skymap file.

Parameters

- **location_code** (*str*) – The four character location name.
- **time** (*str or datetime.datetime*) – A ISO-fomatted time string or datetime object. Must be in UT time.
- **redownload** (*bool*) – Redownload all skymaps.

Red-line Emission Geospace Observatory (REGO)

asilib.asi.**rego**(*location_code*, *time=None*, *time_range=None*, *alt=110*, *custom_alt=False*, *redownload=False*, *missing_ok=True*, *load_images=True*, *imager=<class 'asilib.imager.Imager'>*)

Create an Imager instance with the REGO ASI images and skymaps.

Redline Emission Geospace Observatory (REGO) data is courtesy of Space Environment Canada (space-environment.ca). Use of the data must adhere to the rules of the road for that dataset. Please see below for the required data acknowledgement. Any questions about the REGO instrumentation or data should be directed to the University of Calgary, Emma Spanswick (elspansw@ucalgary.ca) and/or Eric Donovan (edonovan@ucalgary.ca).

"The Redline Emission Geospace Observatory (REGO) is a joint Canada Foundation for Innovation and Canadian Space Agency project developed by the University of Calgary. REGO is operated and maintained by Space Environment Canada with the support of the Canadian Space Agency (CSA) [23SUGOSEC]."

Parameters

- **location_code** (*str*) – The ASI's location code (four letters).
- **time** (*str or datetime.datetime*) – A time to look for the ASI data at. Either time or time_range must be specified (not both or neither).
- **time_range** (*list of str or datetime.datetime*) – A length 2 list of string-formatted times or datetimes to bracket the ASI data time interval.
- **alt** (*int*) – The reference skymap altitude, in kilometers.
- **custom_alt** (*bool*) – If True, asilib will calculate (lat, lon) skymaps assuming a spherical Earth. Otherwise, it will use the official skymaps (Courtesy of University of Calgary).

  Note: The spherical model of Earth's surface is less accurate than the oblate spheroid geometrical representation. Therefore, there will be a small difference between these and the official skymaps.

- **redownload** (*bool*) – If True, will download the data from the internet, regardless of wether or not the data exists locally (useful if the data becomes corrupted).
- **missing_ok** (*bool*) – Wether to allow missing data files inside time_range (after searching for them locally and online).
- **load_images** (*bool*) – Create an Imager object without images. This is useful if you need to calculate conjunctions and don't need to download or load unnecessary data.
- **imager** (asilib.Imager) – Controls what Imager instance to return, asilib.Imager by default. This parameter is useful if you need to subclass asilib.Imager.

Returns

The THEMIS Imager instance.

Return type

Imager()

Examples

```
>>> # Plot a single image.
>>> from datetime import datetime
>>> import matplotlib.pyplot as plt
>>> import asilib.asi
>>> import asilib.map
>>> location_code = 'RANK'
>>> time = datetime(2017, 9, 15, 2, 34, 0)
>>> alt_km = 110
>>> fig = plt.figure(figsize=(10, 6))
>>> ax = fig.add_subplot(121)
>>> bx = asilib.map.create_map(fig_ax=(fig, 122), lon_bounds = (-102, -82), lat_
↪bounds = (58, 68))
>>> asi = asilib.asi.rego(location_code, time=time, alt=alt_km)
>>> asi.plot_fisheye(ax=ax)
>>> asi.plot_map(ax=bx)
>>> plt.tight_layout()
>>> plt.show()
```

Returns
A REGO ASI instance with the time stamps, images, skymaps, and metadata.

Return type
Imager()

asilib.asi.**rego_info**()

Returns a pd.DataFrame with the REGO ASI names and locations.

Returns
A table of REGO imager names and locations.

Return type
pd.DataFrame

asilib.asi.**rego_skymap**(*location_code*, *time*, *redownload=False*)

Load a REGO ASI skymap file.

Parameters

- **location_code** (*str*) – The four character location name.
- **time** (*str or datetime.datetime*) – A ISO-fomatted time string or datetime object. Must be in UT time.
- **redownload** (*bool*) – Redownload the file.

Returns
The skymap.

Return type
dict

Transition Region Explorer (TREx)

asilib.asi.**trex_rgb**(*location_code*, *time=None*, *time_range=None*, *alt=110*, *custom_alt=False*, *redownload=False*, *missing_ok=True*, *load_images=True*, *colors='rgb'*, *burst=False*, *imager=<class 'asilib.imager.Imager'>*)

Create an Imager instance using the TREX-RGB ASI images and skymaps.

Transition Region Explorer (TREx) RGB data is courtesy of Space Environment Canada (space-environment.ca). Use of the data must adhere to the rules of the road for that dataset. Please see below for the required data acknowledgement. Any questions about the TREx instrumentation or data should be directed to the University of Calgary, Emma Spanswick (elspansw@ucalgary.ca) and/or Eric Donovan (edonovan@ucalgary.ca).

"The Transition Region Explorer RGB (TREx RGB) is a joint Canada Foundation for Innovation and Canadian Space Agency project developed by the University of Calgary. TREx-RGB is operated and maintained by Space Environment Canada with the support of the Canadian Space Agency (CSA) [23SUGOSEC]."

For more information see: https://www.ucalgary.ca/aurora/projects/trex.

Parameters

- **location_code** (*str*) – The ASI's location code (four letters).
- **time** (*str or datetime.datetime*) – A time to look for the ASI data at. Either time or time_range must be specified (not both or neither).
- **time_range** (*list of str or datetime.datetime*) – A length 2 list of string-formatted times or datetimes to bracket the ASI data time interval.

- **alt** (*int*) – The reference skymap altitude, in kilometers.
- **custom_alt** (*bool*) – If True, asilib will calculate (lat, lon) skymaps assuming a spherical Earth. Otherwise, it will use the official skymaps (Courtesy of University of Calgary).

  Note: The spherical model of Earth's surface is less accurate than the oblate spheroid geometrical representation. Therefore, there will be a small difference between these and the official skymaps.

- **redownload** (*bool*) – If True, will download the data from the internet, regardless of wether or not the data exists locally (useful if the data becomes corrupted).
- **missing_ok** (*bool*) – Wether to allow missing data files inside time_range (after searching for them locally and online).
- **load_images** (*bool*) – Create an Imager object without images. This is useful if you need to calculate conjunctions and don't need to download or load unnecessary data.
- **colors** (*str*) – Load all three color channels if "rgb", or individual color channels specified by "r", "g", "b" (or any combination of them).
- **burst** (*bool*) – Sometimes Trex-rgb uses a burst mode with higher resolution.
- **imager** (*Imager()*) – Controls what Imager instance to return, asilib.Imager by default. This parameter is useful if you need to subclass asilib.Imager.

Returns
The trex Imager instance.

Return type
Imager()

Examples

```
>>> from datetime import datetime
>>>
>>> import matplotlib.pyplot as plt
>>> import asilib.map
>>> import asilib
>>> from asilib.asi import trex_rgb
>>>
>>> time = datetime(2021, 11, 4, 7, 3, 51)
>>> location_codes = ['FSMI', 'LUCK', 'RABB', 'PINA', 'GILL']
>>> asi_list = []
>>> ax = asilib.map.create_simple_map()
>>> for location_code in location_codes:
>>>     asi_list.append(trex_rgb(location_code, time=time, colors='rgb'))
>>>
>>> asis = asilib.Imagers(asi_list)
>>> asis.plot_map(ax=ax)
>>> ax.set(title=time)
>>> plt.tight_layout()
>>> plt.show()
```

asilib.asi.**trex_rgb_info**()

Returns a pd.DataFrame with the TREx RGB ASI names and locations.

Returns
A table of TREx-RGB imager names and locations.

Return type
pd.DataFrame

asilib.asi.**trex_rgb_skymap**(*location_code*, *time*, *redownload=False*)

Load a TREx RGB skymap file.

Parameters

- **location_code** (*str*) – The four character location name.
- **time** (*str or datetime.datetime*) – A ISO-formatted time string or datetime object. Must be in UT time.
- **redownload** (*bool*) – Redownload the file.

Returns
The skymap.

Return type
dict

asilib.asi.**trex_nir**(*location_code*, *time=None*, *time_range=None*, *alt=110*, *custom_alt=False*, *redownload=False*, *missing_ok=True*, *load_images=True*, *imager=<class 'asilib.imager.Imager'>*)

Create an Imager instance using the TREX-NIR ASI images and skymaps.

Transition Region Explorer (TREx) NIR data is courtesy of Space Environment Canada (space-environment.ca). Use of the data must adhere to the rules of the road for that dataset. Please see below for the required data acknowledgement. Any questions about the TREx instrumentation or data should be directed to the University of Calgary, Emma Spanswick (elspansw@ucalgary.ca) and/or Eric Donovan (edonovan@ucalgary.ca).

"The Transition Region Explorer NIR (TREx NIR) is a joint Canada Foundation for Innovation and Canadian Space Agency project developed by the University of Calgary. TREx-NIR is operated and maintained by Space Environment Canada with the support of the Canadian Space Agency (CSA) [23SUGOSEC]."

For more information see: https://www.ucalgary.ca/aurora/projects/trex.

Parameters

- **location_code** (*str*) – The ASI's location code (four letters).
- **time** (*str or datetime.datetime*) – A time to look for the ASI data at. Either time or time_range must be specified (not both or neither).
- **time_range** (*list of str or datetime.datetime*) – A length 2 list of string-formatted times or datetimes to bracket the ASI data time interval.
- **alt** (*int*) – The reference skymap altitude, in kilometers.
- **custom_alt** (*bool*) – If True, asilib will calculate (lat, lon) skymaps assuming a spherical Earth. Otherwise, it will use the official skymaps (Courtesy of University of Calgary).

Note: The spherical model of Earth's surface is less accurate than the oblate spheroid geometrical representation. Therefore, there will be a small difference between these and the official skymaps.

- **redownload** (*bool*) – If True, will download the data from the internet, regardless of wether or not the data exists locally (useful if the data becomes corrupted).
- **missing_ok** (*bool*) – Wether to allow missing data files inside time_range (after searching for them locally and online).
- **load_images** (*bool*) – Create an Imager object without images. This is useful if you need to calculate conjunctions and don't need to download or load unnecessary data.
- **imager** (asilib.Imager) – Controls what Imager instance to return, asilib.Imager by default. This parameter is useful if you need to subclass asilib.Imager.

Returns

The THEMIS Imager instance.

Return type

Imager()

Examples

```
>>> import asilib
>>> import asilib.map
>>> import asilib.asi
>>> import matplotlib.pyplot as plt
>>>
>>> fig = plt.figure(figsize=(10, 6))
>>> ax = fig.add_subplot(121)
>>> bx = asilib.map.create_map(fig_ax=(fig, 122), lon_bounds=(-102, -86), lat_
↪bounds=(51, 61))
>>> asi = asilib.asi.trex_nir('gill', time='2020-03-21T06:00')
>>> asi.plot_fisheye(ax=ax)
>>> asi.plot_map(ax=bx)
>>> plt.tight_layout()
>>> plt.show()
```

```
>>> import asilib
>>> import asilib.map
>>> import asilib.asi
>>> import matplotlib.pyplot as plt
>>>
>>> time_range = ('2020-03-21T05:00', '2020-03-21T07:00')
>>> fig, ax = plt.subplots(2, sharex=True)
>>> asi = asilib.asi.trex_nir('gill', time_range=time_range)
>>> asi.plot_keogram(ax=ax[0])
>>> asi.plot_keogram(ax=ax[1], aacgm=True)
>>> ax[0].set_title(f'TREX_NIR GILL keogram | {time_range}')
>>> ax[0].set_ylabel('Geo Lat')
>>> ax[1].set_ylabel('Mag Lat')
>>> ax[1].set_xlabel('Time')
>>> plt.show()
```

---

asilib.asi.**trex_nir_skymap**(*location_code*, *time*, *redownload=False*)

Load a TREx NIR ASI skymap file.

Parameters

- **location_code** (*str*) – The four character location name.
- **time** (*str or datetime.datetime*) – A ISO-formatted time string or datetime object. Must be in UT time.
- **redownload** (*bool*) – Redownload the file.

Returns
The skymap.

Return type
dict

asilib.asi.**trex_nir_info**()

Returns a pd.DataFrame with the TREx NIR ASI names and locations.

Returns
A table of TREx-RGB imager names and locations.

Return type
pd.DataFrame

1.4.2 Class Interface

class asilib.**Imager**(*file_info*, *meta*, *skymap*, *plot_settings={}*)

Bases: object

The central asilib class to plot, animate, and analyze ASI data.

Normally asilib.Imager() should not be directly called by users, but by the ASI wrapper functions. This interface is thoroughly documented in the *Adding A New ASI* documentation page.

Parameters

- **file_info** (*dict*) – Specifies image file paths, start end end times for each file, the loader function, and if the user needs one or multiple images.
- **meta** (*dict*) – Specifies ASI metadata that describes the ASI name, location, cadence, and pixel resolution.
- **skymap** (*dict*) – Specifies what each pixel maps to in (azimuth, elevation) coordinates as well as (latitude, longitude) coordinates at a prescribed auroral mission altitude.
- **plot_settings** (*dict*) – An optional dictionary customizing the plot colormap, color scale (logarithmic vs linear), and color bounds (vmin, vmax arguments in matplotlib.imshow()).

plot_fisheye(*ax=None*, *label=True*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *azel_contours=False*, *azel_contour_color='yellow'*, *cardinal_directions='NE'*, *origin=(0.8, 0.1)*)

Plots one fisheye image, oriented with North on the top, and East on the left of the image.

Parameters

- **ax** (*plt.Axes*) – The subplot to plot the image on. If None this method will create one.
- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. By default will use a black-white colormap. For more information See matplotlib colormaps.

- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.
- **azel_contours** (*bool*) – Superpose azimuth and elevation contours on or off.
- **azel_contour_color** (*str*) – The color of the azimuth and elevation contours.
- **cardinal_directions** (*str*) – Plot one or more cardinal directions specified with a string containing the first letter of one or more cardinal directions. Case insensitive. For example, to plot the North and East directions, set cardinal_directions='NE'.
- **origin** (*tuple*) – The origin of the cardinal direction arrows.

Returns

- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*matplotlib.collections.QuadMesh*) – The plt.imshow image object. Common use for im is to add a colorbar. The image is oriented in the map orientation (north is up, south is down, west is right, and east is left). Set azel_contours=True to confirm.

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

```
>>> # A bright auroral arc that was analyzed by Imajo et al., 2021 "Active
>>> # auroral arc powered by accelerated electrons from very high altitudes"
>>> from datetime import datetime
>>> import matplotlib.pyplot as plt
>>> import asilib.asi
>>>
>>> asi = asilib.asi.themis('RANK', time=datetime(2017, 9, 15, 2, 34, 0))
>>> ax, im = asi.plot_fisheye(cardinal_directions='NE', origin=(0.95, 0.05))
>>> plt.colorbar(im)
>>> ax.axis('off')
>>> plt.show()
```

---

animate_fisheye(***kwargs*)

A wrapper for the `Imager.animate_fisheye_gen()` method that animates a series of fisheye images. Any kwargs are passed directly into `Imager.animate_fisheye_gen()`.

Parameters

- **ax** (*plt.Axes*) – The optional subplot that will be drawn on.

- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. By default will use a black-white colormap. For more information See matplotlib colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.
- **azel_contours** (*bool*) – Superpose azimuth and elevation contours on or off.
- **azel_contour_color** (*str*) – The color of the azimuth and elevation contours.
- **cardinal_directions** (*str*) – Plot one or more cardinal directions specified with a string containing the first letter of one or more cardinal directions. Case insensitive. For example, to plot the North and East directions, set cardinal_directions='NE'.
- **origin** (*tuple*) – The origin of the cardinal direction arrows.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_params** (*dict*) – The additional/overwritten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **overwrite** (*bool*) – If true, the output will be overwritten automatically. If false it will prompt the user to answer y/n.

Return type
None

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

```
>>> from datetime import datetime
>>> import asilib.asi
>>>
>>> time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
>>> asi = asilib.asi.themis('FSMI', time_range=time_range)
>>> asi.animate_fisheye(cardinal_directions='NE', origin=(0.95, 0.05),␣
↪overwrite=True)
>>> print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" /␣
↪asi.animation_name}')
```

animate_fisheye_gen(*ax=None*, *label=True*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *azel_contours=False*, *azel_contour_color='yellow'*, *cardinal_directions='NE'*, *origin=(0.8, 0.1)*, *movie_container='mp4'*, *animation_save_dir=None*, *ffmpeg_params={}*, *overwrite=False*)

Animate a series of fisheye images and superpose your data on each image.

A generator behaves like an iterator in that it plots one fisheye image at a time and yields (similar to returns) the image. You can modify, or add content to the image (such as a spacecraft position). Then, once the iteration is complete, this method stitches the images into an animation. See the examples below and in the examples page for use cases. The `animate_fisheye()` method takes care of the iteration.

Parameters

- **ax** (*plt.Axes*) – The optional subplot that will be drawn on.
- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. By default will use a black-white colormap. For more information See matplotlib colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.
- **azel_contours** (*bool*) – Superpose azimuth and elevation contours on or off.
- **azel_contour_color** (*str*) – The color of the azimuth and elevation contours.
- **cardinal_directions** (*str*) – Plot one or more cardinal directions specified with a string containing the first letter of one or more cardinal directions. Case insensitive. For example, to plot the North and East directions, set cardinal_directions='NE'.
- **origin** (*tuple*) – The origin of the cardinal direction arrows.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_params** (*dict*) – The additional/overwitten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **overwrite** (*bool*) – Overwrite the animation. If False, ffmpeg will prompt you to answer y/n if the animation already exists.

Yields

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.ndarray*) – A 2d image array of the image corresponding to image_time
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.

- **im** (*matplotlib.collections.QuadMesh*) – The plt.imshow image object. Common use for im is to add a colorbar. The image is oriented in the map orientation (north is up, south is down, west is right, and east is left). Set azel_contours=True to confirm.

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

```
>>> from datetime import datetime
>>> import asilib.asi
>>> import asilib
>>>
>>> time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
>>> asi = asilib.asi.themis('FSMI', time_range=time_range)
>>> gen = asi.animate_fisheye_gen(cardinal_directions='NE', origin=(0.95, 0.05),
↪ overwrite=True)
>>> for image_time, image, ax, im in gen:
...         # Add your code that modifies each image here.
...         pass
...
>>> print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" /␣
↪asi.animation_name}')
```

plot_map(*lon_bounds=(-160, -50)*, *lat_bounds=(40, 82)*, *ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *min_elevation=10*, *asi_label=True*, *pcolormesh_kwargs={}*)

Projects an ASI image onto a map at an altitude that is defined in the skymap calibration file.

Parameters

- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.
- **ax** (*plt.Axes, tuple*) – The subplot to put the map on. If cartopy is installed, `ax` must be a two element tuple specifying the `plt.Figure` object and subplot position passed directly as `args` into fig.add_subplot().
- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.
- **color_map** (*str*) – The matplotlib colormap to use. See matplotlib colormaps.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.

- **min_elevation** (*float*) – Masks the pixels below min_elevation degrees.
- **asi_label** (*bool*) – Annotates the map with the ASI code in the center of the mapped image.
- **color_bounds** (*List[float] or None*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile)
- **color_norm** – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic.
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh.

Returns

- *plt.Axes* – The subplot object to modify the axis, labels, etc.
- *matplotlib.collections.QuadMesh* – The plt.pcolormesh image object. Common use for p is to add a colorbar.

Examples

```
>>> # Project an image of STEVE onto a map.
>>> from datetime import datetime
>>> import matplotlib.pyplot as plt
>>> import asilib.asi
>>>
>>> asi = asilib.asi.themis('ATHA', time=datetime(2010, 4, 5, 6, 7, 0))
>>> asi.plot_map(lon_bounds=(-127, -100), lat_bounds=(45, 65))
>>> plt.tight_layout()
>>> plt.show()
```

---

animate_map(***kwargs*)

A wrapper for the `animate_map_gen()` method that animates a series of mapped images. Any kwargs are passed directly into `animate_map_gen()`.

Parameters

- **ax** (*plt.Axes*) – The optional subplot that will be drawn on.
- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. By default will use a black-white colormap. For more information See matplotlib colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)

- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.
- **azel_contours** (*bool*) – Superpose azimuth and elevation contours on or off.
- **azel_contour_color** (*str*) – The color of the azimuth and elevation contours.
- **cardinal_directions** (*str*) – Plot one or more cardinal directions specified with a string containing the first letter of one or more cardinal directions. Case insensitive. For example, to plot the North and East directions, set cardinal_directions='NE'.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_params** (*dict*) – The additional/overwritten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **overwrite** (*bool*) – If true, the output will be overwritten automatically. If false it will prompt the user to answer y/n.

Example

```
>>> from datetime import datetime
>>> import asilib.asi
>>> import asilib
>>>
>>> location = 'FSMI'
>>> time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
>>> asi = asilib.asi.themis(location, time_range=time_range)
>>> asi.animate_map(overwrite=True)
>>> print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" /␣
↪asi.animation_name}')
```

animate_map_gen(*lon_bounds=(-160, -50)*, *lat_bounds=(40, 82)*, *ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *min_elevation=10*, *pcolormesh_kwargs={}*, *asi_label=True*, *movie_container='mp4'*, *animation_save_dir=None*, *ffmpeg_params={}*, *overwrite=False*)

Animate a series of mapped images and superpose your data on each image.

A generator behaves like an iterator in that it plots one fisheye image at a time and yields (similar to returns) the image. You can modify, or add content to the image (such as a spacecraft position). Then, once the iteration is complete, this method stitches the images into an animation. See the examples below and in the examples page for use cases. The `animate_fisheye()` method takes care of the iteration.

Parameters

- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.
- **ax** (*plt.Axes, tuple*) – The subplot to put the map on. If cartopy is installed, `ax` must be a two element tuple specifying the `plt.Figure` object and subplot position passed directly as `args` into fig.add_subplot().

- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.
- **color_map** (*str*) – The matplotlib colormap to use. See matplotlib colormaps for supported colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.
- **min_elevation** (*float*) – Masks the pixels below min_elevation degrees.
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh.
- **asi_label** (*bool*) – Annotates the map with the ASI code in the center of the mapped image.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_params** (*dict*) – The additional/overwritten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **overwrite** (*bool*) – Overwrite the animation. If False, ffmpeg will prompt you to answer y/n if the animation already exists.

Yields

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.ndarray*) – A 2d image array of the image corresponding to image_time
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*matplotlib.collections.QuadMesh*) – The plt.imshow image object. Common use for im is to add a colorbar. The image is oriented in the map orientation (north is up, south is down, west is right, and east is left). Set azel_contours=True to confirm.

Example

```
>>> from datetime import datetime
>>> import matplotlib.pyplot as plt
>>> import asilib
>>> import asilib.asi
>>>
>>> location = 'FSMI'
```

(continues on next page)

(continued from previous page)

```
>>> time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
>>> asi = asilib.asi.themis(location, time_range=time_range)
>>> ax = asilib.map.create_map(lon_bounds=(-120, -100), lat_bounds=(55, 65))
>>> plt.tight_layout()
>>>
>>> gen = asi.animate_map_gen(overwrite=True, ax=ax)
>>>
>>> for image_time, image, ax, im in gen:
>>>     # Add your code that modifies each image here...
>>>     # To demonstrate, lets annotate each frame with the timestamp.
>>>     # We will need to delete the prior text object, otherwise the current␣
↪one
>>>     # will overplot on the prior one---clean up after yourself!
>>>     if 'text_obj' in locals():
>>>             ax.texts.remove(text_obj)
>>>     text_obj = ax.text(0, 0.9, f'THEMIS-{location} at {image_time:%F %T}',
>>>             transform=ax.transAxes, color='white', fontsize=15)
>>>
>>> print(f'Animation saved in {asilib.config["ASI_DATA_DIR"] / "animations" /␣
↪asi.animation_name}')
```

---

keogram(*path=None*, *aacgm=False*, *minimum_elevation=0*)

Create a keogram along the meridian or a custom path.

Parameters

- **path** (*array*) – Make a keogram along a custom path. The path is a lat/lon array of shape (n, 2). Longitude must be between [-180, 180].
- **aacgm** (*bool*) – Map the keogram latitudes to Altitude Adjusted Corrected Geogmagnetic Coordinates (aacgmv2) derived by Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501-7521, doi:10.1002/2014JA020264.
- **minimum_elevation** (*float*) – The minimum elevation of pixels to use in the keogram.

Returns

- *np.array* – Keogram timestamps
- *np.array* – Keogram latitudes: geographic if the aacgm kwarg is False and magnetic is aacgm if True.
- *np.array* – Keogram array with rows corresponding to times and columns with latitudes.

Example

```
>>> # A keogram in geographic and magnetic latitude coordinates. See
>>> # Imager.plot_keogram() on how to plot a keogram.
>>> # Event from https://doi.org/10.1029/2021GL094696
>>> import numpy as np
>>> import asilib.asi
>>>
>>> time_range=['2008-01-16T10', '2008-01-16T12']
```

(continues on next page)

(continued from previous page)

```
>>>
>>> asi = asilib.asi.themis('GILL', time_range=time_range)
>>> time, geo_lat, geo_keogram = asi.keogram()  # geographic latitude
>>> time, mag_lat, mag_keogram = asi.keogram(aacgm=True)  # magnetic latitude
>>> time
array([datetime.datetime(2008, 1, 16, 10, 0, 0, 20162),
    datetime.datetime(2008, 1, 16, 10, 0, 3, 9658),
    datetime.datetime(2008, 1, 16, 10, 0, 6, 29345), ...,
    datetime.datetime(2008, 1, 16, 11, 59, 51, 50496),
    datetime.datetime(2008, 1, 16, 11, 59, 54, 10602),
    datetime.datetime(2008, 1, 16, 11, 59, 57, 60543)], dtype=object)
>>> geo_lat[:10]
array([47.900368, 48.506763, 49.057587, 49.556927, 50.009083, 50.418365,
    50.788963, 51.124836, 51.42965 , 51.706768], dtype=float32)
>>> mag_lat[:10]
array([57.97198543, 58.55886486, 59.09144098, 59.57379565, 60.01019679,
    60.40490277, 60.76203547, 61.0854798 , 61.37882613, 61.64536043])
>>> np.all(mag_keogram == geo_keogram)  # aacgm=True only changes the latitudes.
True
```

---

plot_keogram(*ax=None*, *path=None*, *aacgm=False*, *title=True*, *minimum_elevation=0*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *pcolormesh_kwargs={}*)

Plot a keogram along the meridian or a custom path.

Parameters

- **ax** (*plt.Axes*) – The subplot to plot the keogram on.
- **path** (*np.array*) – Make a keogram along a custom path. The path is a lat/lon array of shape (n, 2). Longitude must be between [-180, 180].
- **aacgm** (*bool*) – Map the keogram latitudes to Altitude Adjusted Corrected Geogmagnetic Coordinates (aacgmv2) derived by Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501-7521, doi:10.1002/2014JA020264.
- **title** (*bool*) – Add a plot title with the date, ASI array, and ASI location.
- **minimum_elevation** (*float*) – The minimum elevation of pixels to use in the keogram.
- **color_map** (*str*) – The matplotlib colormap to use. See matplotlib colormaps for supported colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic. The norm is not applied to RGB images (see matplotlib.pyplot.imshow)
- **color_brighten** (*bool*) – If True, scales the RGB intensities from min(image)-max(image) to 0-1 range. This results in brighter colors. This is only applied to RGB images.

- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh.

Returns

- *plt.Axes* – The subplot object to modify the axis, labels, etc.
- *matplotlib.collections.QuadMesh* – The plt.pcolormesh image object, useful to add a colorbar, for example.

Example

```
>>> # Plot a keogram in geographic and magnetic latitude coordinates.
>>> # Event from https://doi.org/10.1029/2021GL094696
>>> import matplotlib.pyplot as plt
>>> import asilib.asi
>>>
>>> time_range=['2008-01-16T10', '2008-01-16T12']
>>> fig, ax = plt.subplots(2, sharex=True, figsize=(10, 6))
>>>
>>> asi = asilib.asi.themis('GILL', time_range=time_range)
>>> _, p = asi.plot_keogram(ax=ax[0], color_map='turbo')
>>> asi.plot_keogram(ax=ax[1], color_map='turbo', aacgm=True, title=False)
>>>
>>> ax[0].set_ylabel('Geographic Lat [deg]')
>>> ax[1].set_ylabel('Magnetic Lat [deg]')
>>> fig.subplots_adjust(right=0.8)
>>> cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
>>> fig.colorbar(p, cax=cbar_ax)
>>> plt.show()
```

`iter_files()`

Iterate one ASI file (or large chunks of a file) at a time. The output data is clipped by time_range.

Yields

- *np.array* – ASI timestamps in datetine.datetime() or numpy.datetime64() format.
- *np.array* – ASI images.

Example

Loop over ~5 minutes of data, starting in the middle of one file.

```
>>> import asilib.asi
>>> time_range=['2008-01-16T10:00:30', '2008-01-16T10:05']
>>> asi = asilib.asi.themis('GILL', time_range=time_range)
>>> for file_times, file_images in asi.iter_files():
...     print(file_times[0], file_times[-1], file_images.shape)
...
2008-01-16 10:00:30.019526 2008-01-16 10:00:57.049007 (10, 256, 256)
2008-01-16 10:01:00.058996 2008-01-16 10:01:57.049620 (20, 256, 256)
2008-01-16 10:02:00.059597 2008-01-16 10:02:57.029981 (20, 256, 256)
2008-01-16 10:03:00.050822 2008-01-16 10:03:57.020254 (20, 256, 256)
2008-01-16 10:04:00.030448 2008-01-16 10:04:57.046170 (20, 256, 256)
```

property data

Load ASI data.

Returns

A named tuple containing (times, images). Members can be accessed using either index notation, or dot notation.

Return type

namedtuple

class asilib.**Imagers**(*imagers*, *iter_tol=2*)

Bases: object

Plot and animate multiple *Imager()* objects.

Warning: This class is in development and not all methods are implemented/matured.

Parameters

- **imagers** (*Tuple*) – The Imager objects to plot and animate.
- **iter_tol** (*float*) – The allowable time tolerance, in units of time_tol*imager_cadence, for imagers to be considered synchronized. Adjusting this kwarg is useful if the imager has missing data and you need to animate a mosaic.

plot_fisheye(*ax*, ***kwargs*)

Plots one fisheye image in each subplot, oriented with North on the top, and East on the left of the image.

Parameters

- **ax** (*Tuple[plt.Axes]*) – Subplots corresponding to each fisheye lens image.
- **kwargs** (*dict*) – Keyword arguments directly passed into each *plot_fisheye()* method.

Example

```
>>> from datetime import datetime
>>>
>>> import matplotlib.pyplot as plt
>>> import asilib
>>> import asilib.asi
>>>
>>> time = datetime(2007, 3, 13, 5, 8, 45)
>>> location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
>>>
>>> fig, ax = plt.subplots(1, len(location_codes), figsize=(12, 3.5))
>>>
>>> _imagers = []
>>>
>>> for location_code in location_codes:
>>>     _imagers.append(asilib.asi.themis(location_code, time=time))
>>>
>>> for ax_i in ax:
>>>     ax_i.axis('off')
```

(continues on next page)

(continued from previous page)

```
>>>
>>> asis = asilib.Imagers(_imagers)
>>> asis.plot_fisheye(ax=ax)
>>>
>>> plt.suptitle('Donovan et al. 2008 | First breakup of an auroral arc')
>>> plt.tight_layout()
>>> plt.show()
```

---

plot_map(*overlap=False*, ***kwargs*)

Projects multiple ASI images onto a map at an altitude that is defined in the skymap calibration file.

Parameters

- **overlap** (*bool*) – If True, pixels that overlap between imager FOV's are overplotted such that only the final imager's pixels are shown.
- **kwargs** (*dict*) – Keyword arguments directly passed into each *plot_map()* method.

Example

```
>>> from datetime import datetime
>>>
>>> import matplotlib.pyplot as plt
>>>
>>> import asilib
>>> import asilib.map
>>> import asilib.asi
>>>
>>> time = datetime(2007, 3, 13, 5, 8, 45)
>>> location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
>>> map_alt = 110
>>> min_elevation = 2
>>>
>>> ax = asilib.map.create_map(lon_bounds=(-140, -60), lat_bounds=(40, 82))
>>>
>>> _imagers = []
>>>
>>> for location_code in location_codes:
>>>     _imagers.append(asilib.asi.themis(location_code, time=time, alt=map_
↪alt))
>>>
>>> asis = asilib.Imagers(_imagers)
>>> asis.plot_map(ax=ax, overlap=False, min_elevation=min_elevation)
>>>
>>> ax.set_title('Donovan et al. 2008 | First breakup of an auroral arc')
>>> plt.show()
```

---

animate_map(***kwargs*)

Animate an ASI mosaic. It is a wrapper for the *animate_map_gen()* method.

See *animate_map_gen()* documentation for the complete list of kwargs.

Example

```
>>> import asilib
>>> import asilib.asi
>>>
>>> time_range = ('2021-11-04T06:55', '2021-11-04T07:05')
>>> asis = asilib.Imagers(
>>>     [asilib.asi.trex_rgb(location_code, time_range=time_range)
>>>     for location_code in ['LUCK', 'PINA', 'GILL', 'RABB']]
>>>     )
>>> asis.animate_map(lon_bounds=(-115, -85), lat_bounds=(43, 63),␣
↪overwrite=True)
```

animate_map_gen(*overlap=False*, *lon_bounds=(-160, -50)*, *lat_bounds=(40, 82)*, *ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *color_map=None*, *color_bounds=None*, *color_norm=None*, *color_brighten=True*, *min_elevation=10*, *pcolormesh_kwargs={}*, *asi_label=True*, *movie_container='mp4'*, *animation_save_dir=None*, *ffmpeg_params={}*, *overwrite=False*)

Animate an ASI mosaic.

Parameters

- **overlap** (*bool*) – If True, pixels that overlap between imager FOV's are overplotted such that only the final imager's pixels are shown.
- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.
- **ax** (*plt.Axes, tuple*) – The subplot to put the map on. If cartopy is installed, `ax` must be a two element tuple specifying the `plt.Figure` object and subplot position passed directly as `args` into fig.add_subplot().
- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.
- **color_map** (*str*) – The matplotlib colormap to use. See matplotlib colormaps for supported colormaps.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. The default is: low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Set the 'lin' (linear) or 'log' (logarithmic) color normalization. If color_norm=None, the color normalization will be taken from the ASI array (if specified), and if not specified it will default to logarithmic.
- **min_elevation** (*float*) – Masks the pixels below min_elevation degrees.
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh.
- **asi_label** (*bool*) – Annotates the map with the ASI code in the center of the mapped image.

- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_params** (*dict*) – The additional/overwritten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **overwrite** (*bool*) – Overwrite the animation. If False, ffmpeg will prompt you to answer y/n if the animation already exists.

Yields

- *datetime.datetime* – The guide time used to keep the images synchronized.
- *List[datetime.datetime]* – Nearest imager time stamps to the guide time. If the difference between the imager time and the guide time is greater than time_tol*imager_cadence, or the imager is off, the imager is considered unsynchronized and the returned time is datetime.min.
- *List[np.ndarray]* – The images corresponding to the times returned above. If the that imager is unsynchronized, the corresponding image value is None.
- *plt.Axes* – The subplot object.

Example

```
>>> # Animate a TREX-RGB mosaic and print the individual time stamps
>>> # to confirm that the imagers are synchronized.
>>> import asilib
>>> import asilib.asi
>>>
>>> time_range = ('2021-11-04T06:55', '2021-11-04T07:05')
>>> asis = asilib.Imagers(
>>>     [asilib.asi.trex_rgb(location_code, time_range=time_range)
>>>     for location_code in ['LUCK', 'PINA', 'GILL', 'RABB']]
>>>     )
>>> gen = asis.animate_map_gen(
>>>     lon_bounds=(-115, -85), lat_bounds=(43, 63), overwrite=True
>>>     )
>>> for guide_time, asi_times, asi_images, ax in gen:
>>>     if '_text_obj' in locals():
>>>         _text_obj.remove()
>>>     info_str = f'Guide: {guide_time: %Y:%m:%d %H:%M:%S}\n'
>>>     # The below for loop is possible because the imagers and
>>>     # asi_times can be indexed together.
>>>     for _imager, _imager_time in zip(asis.imagers, asi_times):
>>>         info_str += f'{_imager.meta["location"]}: {_imager_time: %Y:%m:%d
↪%H:%M:%S}\n'
>>>     info_str = info_str[:-1]  # Remove the training newline
>>>
>>>     _text_obj = ax.text(
>>>         0.01, 0.99, info_str, va='top', transform=ax.transAxes,
>>>         bbox=dict(facecolor='grey', edgecolor='black'))
```

get_points(*min_elevation=10*)

Get pixel intensities in each (lat, lon) grid point.

Parameters

min_elevation (*float*) – Only return pixel intensities above min_elevation.

Returns

- *np.ndarray* – An (n, 2) array with each row corresponding to a (lat, lon) point.
- *np.ndarray* – Pixel intensities with shape (n) for white-light images, and (n, 3) for RGB images.

Examples

```
>>> from datetime import datetime
>>>
>>> import asilib
>>> import asilib.asi
>>>
>>> time = datetime(2007, 3, 13, 5, 8, 45)
>>> location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
>>> map_alt = 110
>>> min_elevation = 2
>>>
>>> _imagers = [asilib.asi.themis(location_code, time=time, alt=map_alt)
>>>             for location_code in location_codes]
>>> asis = asilib.Imagers(_imagers)
>>> lat_lon_points, intensities = asis.get_points(min_elevation=min_elevation)
```

```
>>> # A comprehensive example showing how Imagers.get_points() can closely␣
↪reproduce
>>> # Imagers.plot_map()
>>> from datetime import datetime
>>>
>>> import matplotlib.pyplot as plt
>>> import matplotlib.colors
>>>
>>> import asilib
>>> import asilib.map
>>> import asilib.asi
>>>
>>> time = datetime(2007, 3, 13, 5, 8, 45)
>>> location_codes = ['FSIM', 'ATHA', 'TPAS', 'SNKQ']
>>> map_alt = 110
>>> min_elevation = 2
>>>
>>> _imagers = [asilib.asi.themis(location_code, time=time, alt=map_alt)
>>>             for location_code in location_codes]
>>> asis = asilib.Imagers(_imagers)
>>> lat_lon_points, intensities = asis.get_points(min_elevation=min_elevation)
>>>
>>> fig = plt.figure(figsize=(12,5))
>>> ax = asilib.map.create_simple_map(
```

(continues on next page)

(continued from previous page)

```
>>>     lon_bounds=(-140, -60), lat_bounds=(40, 82), fig_ax=(fig, 121)
>>>     )
>>> bx = asilib.map.create_simple_map(
>>>     lon_bounds=(-140, -60), lat_bounds=(40, 82), fig_ax=(fig, 122)
>>>     )
>>> asis.plot_map(ax=ax, overlap=False, min_elevation=min_elevation)
>>> bx.scatter(lat_lon_points[:, 1], lat_lon_points[:, 0], c=intensities,
>>>         norm=matplotlib.colors.LogNorm())
>>> ax.text(0.01, 0.99, f'(A) Mosaic using Imagers.plot_map()', transform=ax.
↪transAxes,
>>>         va='top', fontweight='bold', color='red')
>>> bx.text(0.01, 0.99, f'(B) Mosaic from Imagers.get_points() scatter',␣
↪transform=bx.transAxes,
>>>         va='top', fontweight='bold', color='red')
>>> fig.suptitle('Donovan et al. 2008 | First breakup of an auroral arc')
>>> plt.tight_layout()
>>> plt.show()
```

---

class asilib.**Conjunction**(*imager*, *satellite*)

Bases: `object`

find(*min_el=20*, *time_gap_s=60*)

Finds the start and end times of conjunctions defined by a minimum elevation.

Parameters

min_el (*float*) – The minimum elevation of the conjunction.

intensity(*box=None*, *box_op=None*)

Calculate the auroral intensity near the satellite's footprint at the ASI time stamps.

Parameters

- **box** (*Tuple[float, float]*) – A tuple of two floats that specifies the rectangular area, at the auroral emission altitude, in which to calculate the auroral intensity. The intensities in the box are averaged by default, and can be changed using the `box_op` kwarg. If `box_size=None`, the auroral intensity will come from the pixel nearest to the footprint. In this case the box_op kwarg is irrelevant.
- **box_op** (*Callable*) – The function to apply to the auroral intensity in the box surrounding the footprint. Since it must operate on the mask array that *equal_area()* produces, the `box_op` function must work with (i.e., ignore) `np.nan` values. If None, the function is np.nanmean() summed over the x-y pixels.

Returns

- *np.ndarray* – The auroral intensity near the footprint, calculated either from the nearest pixel (if `box_size=None`) or the mean intensity in a rectangular area defined by `box_size`.
- .. *note::* – If box_size=None the nearest pixel is calculated using *map_azel()*, otherwise if `box_size=(10, 10)` and `box_op` is the default, *equal_area()* is used to calculate the mean intensity in a 10x10 km box. The two different implementations should yield similar results, but discrepancies may arise if the skymap (az, el) and (lat, lon) mapping arrays are mismatched. This is the case for some of the THEMIS ASIs right after they were deployed.

interp_sat()

Interpolate the satellite timestamps and LLA to imager timestamps.

lla_footprint(*alt*, *b_model='OPQ77'*, *maginput=None*, *hemisphere=0*)

Map the spacecraft's position to `alt` along the magnetic field line. The mapping is implemented in `IRBEM` and by default it maps to the same hemisphere.

Parameters

- **map_alt** (*float*) – The altitude to map to, in km, in the same hemisphere.
- **b_model** (*str*) – The magnetic field model to use, by default the model is Olson-Pfitzer 1974. This parameter is passed directly into IRBEM.MagFields as the 'kext' parameter.
- **maginput** (*dict*) – If you use a different b_model that requires time-dependent parameters, supply the appropriate values to the maginput dictionary. It is directly passed into IRBEM.MagFields so refer to IRBEM on the proper format.
- **hemisphere** (*int*) – The hemisphere to map to. This kwarg is passed to IRBEM and can be one of these four values: 0 = same magnetic hemisphere as starting point +1 = northern magnetic hemisphere -1 = southern magnetic hemisphere +2 = opposite magnetic hemisphere as starting point

Returns

magnetic_footprint – A numpy.array with size (n_times, 3) with lat, lon, alt columns representing the magnetic footprint coordinates.

Return type

np.ndarray

Raises

ImportError – If IRBEM can't be imported.

map_azel(*min_el=0*)

Maps a satellite's location to the ASI's azimuth and elevation (azel) coordinates and image pixel index.

Parameters

min_el (*float*) – The minimum elevation in degrees for which return valid values for. The satellite's azel values and pixel indices are NaN below min_el.

Returns

- *np.ndarray* – An array with shape (nPosition, 2) of the satellite's azimuth and elevation coordinates.
- *np.ndarray* – An array with shape (nPosition, 2) of the x- and y-axis pixel indices for the ASI image.
- .. *note::* – The azel pixel columns are ordered for plotting with an image: plt.plot(azel_pixels[:, 0], azel_pixels[:, 1]). However, the column order must be flipped for indexing. For example: image[azel_pixels[:, 1], azel_pixels[:, 0]]

equal_area(*box=(5, 5)*)

Find all pixels around the footprint within a rectangular area defined by box.

Parameters

box (*Tuple[float, float]*) – Bounds the emission area box dimensions in longitude and latitude. Units are kilometers.

Returns

An array with dimensions (n_time, n_x_pixels, n_y_pixels) with dimensions n_x_pixels and n_y_pixels dimensions the size of each image. Values inside the area are 1 and outside are np.nan.

Return type
np.ndarray

See also:

Conjunction.equal_area_gen
A memory-friendly way to calculate equal areas.

equal_area_gen(*box=(5, 5)*)

Generator to find all pixels around the footprint within a rectangular area defined by box.

Parameters
box (*Tuple[float, float]*) – Bounds the emission box dimensions in longitude and latitude. Units are kilometers.

Returns
An array with (n_time, n_x_pixels, n_y_pixels) dimensions with dimensions n_x_pixels and n_y_pixels dimensions the size of each image. Values inside the area are 1 and outside are np.nan.

Return type
np.ndarray

1.4.3 Geographic Maps

Plot geographic maps using cartopy or the simple built-in function. Before you project an ASI image onto a map, you will need to create a map using the following functions.

The simplest way to create a map is via *create_map()* that by default creates a map above North America. *create_map()* is a wrapper that automatically chooses what library to plot the map: cartopy if it is installed, or asilib's *create_simple_map()* function to create a simple map if cartopy is not installed. All of these functions output the subplot object with the map.

You can override this automatic behavior by calling the underlying functions directly: *create_simple_map()* or *create_cartopy_map()*.

The two functions are called similarly with the exception of the `ax` kwarg if you need to specify what subplot to create the map on:

- for *create_simple_map()* you must pass in a `ax` subplot object, and
- for *create_cartopy_map()*, you need to pass in a `ax` tuple containing two elements. The first element is the `plt.Figure` object, and the second element is a 3 digit number (or a tuple) specifying where to place that subplot. For example,

```
>>> fig = plt.Figure()
>>> ax = asilib.map.create_cartopy_map(ax=(fig, 111))
```

asilib.map.**create_map**(*lon_bounds=(-160, -50)*, *lat_bounds=(40, 82)*, *fig_ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*)

Create a geographic map using cartopy (if installed) or asilib's own map library.

Parameters

- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.

- **fig_ax** (*Tuple[matplotlib.figure.Figure, int]*) – A two element tuple. First element is a `matplotlib.figure.Figure` object and second element is the subplot index or matplotlib.gridspec.SubplotSpec object. The second element is passed directly as `args` fig.add_subplot().

  For example:

  ```
  fig = plt.Figure()
  ax = asilib.map.create_cartopy_map(fig_ax=(fig, 111))
  ```

- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.

Returns
The subplot object containing the map.

Return type
plt.Axes

Examples

These examples will differ if you have cartopy installed or not. To force a consistent map, replace `asilib.map.create_map` with `asilib.map.create_simple_map` or `asilib.map.create_cartopy_map`.

Examples

```
>>> # Create a map above Scandinavia in a single subplot
>>> import asilib.map
>>> import matplotlib.pyplot as plt
>>> ax = asilib.map.create_map(lon_bounds=[0, 38], lat_bounds=[50, 75])
>>> ax.set_title('Generated via asilib.map.create_map()')
>>> plt.show()
```

```
>>> # The above examples made a map on one subplot. But what if you have multiple␣
↪subplots?
>>> import asilib.map
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6, 10))
>>> bx = asilib.map.create_map(lon_bounds=[0, 38], lat_bounds=[50, 75], fig_ax=(fig,
↪ 211))
>>> cx = fig.add_subplot(2, 1, 2)
>>> cx.plot(np.arange(10), np.arange(10))
>>> fig.suptitle('Two subplots with equal sizes')
>>> plt.show()
```

```
>>> # Another multi-subplot example with different height ratios. The syntax is the␣
↪same as in plt.subplot() (See the args section in https://matplotlib.org/stable/
↪api/_as_gen/matplotlib.pyplot.subplot.html).
>>> import asilib.map
>>> import numpy as np
```

(continues on next page)

(continued from previous page)

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6, 10))
>>> dx = (3, 1, (1, 2))
>>> dx = asilib.map.create_map(lon_bounds=[0, 38], lat_bounds=[50, 75], fig_ax=(fig,
↪ dx))
>>> ex = fig.add_subplot(3, 1, 3)
>>> ex.plot(np.arange(10), np.arange(10))
>>> fig.suptitle('Two subplots with unequal sizes')
>>> plt.show()
```

```
>>> # And make a map using gridspec
>>> import asilib.map
>>> import matplotlib.pyplot as plt
>>> import matplotlib.gridspec
>>> fig = plt.figure()
>>> gs = matplotlib.gridspec.GridSpec(1, 1, fig)
>>> dx = asilib.map.create_map(lon_bounds=[0, 38], lat_bounds=[50, 75], fig_ax=(fig,
↪ gs))
>>> dx.set_title('Map made using gridspec')
>>> plt.show()
```

See also:

create_simple_map()
: Create a simple map using asilib's own map library.

create_cartopy_map()
: Create a map using the cartopy library.

asilib.map.**create_simple_map**(*lon_bounds=(-140, -60)*, *lat_bounds=(40, 82)*, *fig_ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *file='ne_10m_land'*)

Create a simple map without cartopy.

Parameters

- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.
- **fig_ax** (*Tuple[matplotlib.figure.Figure, int]*) – A two element tuple. First element is a `matplotlib.figure.Figure` object and second element is the subplot index or matplotlib.gridspec.SubplotSpec object. The second element is passed directly as `args` fig.add_subplot().

  For example:

  ```
  fig = plt.Figure()
  ax = asilib.map.create_simple_map(fig_ax=(fig, 111))
  ```

- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.

- **file** (*str or pathlib.Path*) – The path to the shapefile zip archive. If str, it will try to load the shapefile in asilib/data/{file}. You can download other shapefiles from https://www.naturalearthdata.com/downloads/10m-physical-vectors/.

Examples

See *create_map()* and replace `create_map` with `create_simple_map`.

Returns
The subplot object containing the map.

Return type
plt.Axes

asilib.map.**create_cartopy_map**(*lon_bounds=(-160, -50)*, *lat_bounds=(40, 82)*, *fig_ax=None*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*)

A helper function to create two map styles: a simple black and white map, and a more sophisticated map with green land.

Parameters

- **lon_bounds** (*tuple or list*) – A tuple of length 2 specifying the map's longitude bounds.
- **lat_bounds** (*tuple or list*) – A tuple of length 2 specifying the map's latitude bounds.
- **fig_ax** (*Tuple[matplotlib.figure.Figure, int]*) – A two element tuple. First element is a `matplotlib.figure.Figure` object and second element is the subplot index or matplotlib.gridspec.SubplotSpec object. The second element is passed directly as `args` fig.add_subplot().

  For example:

  ```
  fig = plt.Figure()
  ax = asilib.map.create_cartopy_map(fig_ax=(fig, 111))
  ```

- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.

Returns
ax – The subplot object with a cartopy map.

Return type
plt.Axes

Examples

See *create_map()* and replace `create_map` with `create_simple_map`.

1.5 Legacy API Reference

Warning: This API interface is deprecated and will be removed on or after December 2023. See the *Imager API Reference* for the current (and equivalent) asilib implementation.

asilib saves all of the ASI image files, skymap calibration files, and movies to

```
asilib.config['ASI_DATA_DIR']
```

By default this directory is set to *~/asilib-data/*, but you can configure the paths using the prompt opened by

```
python3 -m asilib config
```

The below functions can be imported and called using either of the following ways:

- ```
  import asilib
  asilib.load_image(...)
  ```

- ```
  from asilib.io.load import load_image
  load_image(...)
  ```

The former option is possible because these functions are all imported by default. However, this may change in a future release, so absolute import (shown in the latter example) is preferred.

Note: The longitude units are converted from [0, 360] to [-180, 180] degrees in the skymap calibration files.

1.5.1 Function Summary

asilib.io.download.download_image	Download a CDF image file when `time` is given, or multiple files when `time_range` is given.
asilib.io.download.download_skymap	Download all of the THEMIS or REGO skymap IDL .sav files.
asilib.io.load.load_skymap	Loads the appropriate THEMIS or REGO skymap file (closest and before `time`) into memory.
asilib.io.load.load_image	Load into memory an ASI image and time stamp when `time` is given, or images with time stamps when `time_range` is given.
asilib.io.load.load_image_generator	Yields multiple ASI image files one by one and crops the time stamps by `time_range`.
asilib.plot.plot_keogram.plot_keogram	Makes a keogram along the central meridian.
asilib.plot.plot_fisheye.plot_fisheye	Plots one fisheye image, oriented with North on the top, and East on the right of the image.
asilib.plot.plot_map.plot_map	Projects the ASI images to a map at an altitude defined in the skymap calibration file.
asilib.plot.animate_fisheye.animate_fisheye	Animate a series of THEMIS or REGO fisheye images.
asilib.plot.animate_fisheye.animate_fisheye_generator	A generator function that loads the ASI data and then yields individual ASI images, image by image.
asilib.plot.animate_map.animate_map	Projects a series of THEMIS or REGO images on a map at map_alt altitude in kilometers and animates them.
asilib.plot.animate_map.animate_map_generator	Projects a series of THEMIS or REGO images on a map at map_alt altitude in kilometers and animates them.
asilib.analysis.keogram.keogram	Makes a keogram pd.DataFrame along the central meridian.
asilib.analysis.map.lla2azel	Maps, a satellite's latitude, longitude, and altitude (LLA) coordinates to the ASI's azimuth and elevation (azel) coordinates and image pixel index.
asilib.analysis.map.lla2footprint	Map the spacecraft's position to `map_alt` along the magnetic field line.
asilib.analysis.equal_area.equal_area	Calculate a `box_km` area at the aurora emission altitude.

1.5.2 Download

asilib.io.download.**download_image**(*asi_array_code*, *location_code*, *time=None*, *time_range=None*, *redownload=False*, *ignore_missing_data=True*)

Download a CDF image file when `time` is given, or multiple files when `time_range` is given.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, `time` must be in the ISO 8601 standard.
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.

- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete.
- **ignore_missing_data** (*bool*) – Flag to ignore the `FileNotFoundError` that is raised when ASI data is unavailable for that date-hour. Only useful when `time_range` is passed.

Returns
download_paths – A list of pathlib.Path objects that contain the downloaded file path(s).

Return type
list

Example

from datetime import datetime

import asilib

asi_array_code = 'THEMIS'
location_code = 'LUCK'
time = datetime(2017, 4, 13, 5)
download_path = asilib.download_image(asi_array_code, location_code, time)

`asilib.io.download.`**download_skymap**(*asi_array_code*, *location_code*, *redownload=False*)

Download all of the THEMIS or REGO skymap IDL .sav files.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete.

Returns
download_paths – A list of pathlib.Path objects that contain the skymap file path(s).

Return type
list

Example

import asilib

asi_array_code = 'THEMIS'
location_code = 'LUCK'
asilib.download_skymap(asi_array_code, location_code)

1.5.3 Load

The following functions are very useful if you want to work with the raw image and skymap data without dealing without explicitly downloading them.

asilib.io.load.**load_image**(*asi_array_code*, *location_code*, *time=None*, *time_range=None*, *redownload=False*, *time_thresh_s=3*, *ignore_missing_data=True*)

Load into memory an ASI image and time stamp when `time` is given, or images with time stamps when `time_range` is given.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, `time` must be in the ISO 8601 standard.
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete.
- **time_thresh_s** (*float*) – The maximum allowable time difference between `time` and an ASI time stamp. This is relevant only when `time` is specified.
- **ignore_missing_data** (*bool*) – Flag to ignore the `FileNotFoundError` that is raised when ASI data is unavailable for that date-hour. Only useful when `time_range` is passed.

Returns

- **times** (*datetime, or List[datetime]*) – The image timestamp if `time` is passed, or a list of timestamps if `time_range` is passed. When `time_range` is passed, the timestamps can include start time if a timestamp exactly matches, but will exclude the timestamp that exactly matches the end time.
- **images** (*np.ndarray*) – Either an (nPixelRows x nPixelCols) or (nTime x nPixelRows x nPixelCols) array containing the ASI images.

Example

```
# Load a single image
asi_array_code = 'THEMIS'
location_code = 'ATHA'
time = datetime(2008, 3, 9, 9, 18, 0)
image_time, image = asilib.load_image(asi_array_code, location_code, time=time, redownload=False)

# Load multiple images
asi_array_code = 'REGO'
location_code = 'LUCK'
time_range = [datetime(2017, 9, 27, 7, 15), datetime(2017, 9, 27, 8, 15)]
image_times, images = asilib.load_image(asi_array_code, location_code, time_range=time_range,
redownload=False)
```

asilib.io.load.**load_image_generator**(*asi_array_code*, *location_code*, *time_range*, *redownload=False*, *ignore_missing_data=True*)

Yields multiple ASI image files one by one and crops the time stamps by `time_range`.

This generator is useful for loading lots of data—useful for keograms. The returned time stamps span a range from time_range[0], up to, but excluding a time stamp exactly matching time_range[1].

See asilib._load_images() for an example on how to use this function.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete.
- **ignore_missing_data** (*bool*) – Flag to ignore the `FileNotFoundError` that is raised when ASI data is unavailable for that date-hour. Only useful when `time_range` is passed.

Yields

- **times** (*datetime*) – The image timestamps contained in time_range, including the start time and excluding the end time (if time_range[1] exactly matches a ASI time stamp).
- **images** (*np.ndarray*) – An (nTime x nPixelRows x nPixelCols) array containing the ASI images for times contained in time_range.

asilib.io.load.**load_skymap**(*asi_array_code*, *location_code*, *time*, *redownload=False*)

Loads the appropriate THEMIS or REGO skymap file (closest and before `time`) into memory.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, `time` must be in the ISO 8601 standard.
- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete.

Returns
The skymap data with longitudes mapped from 0->360 to to -180->180 degrees.

Return type
dict

Example

import asilib

rego_skymap = asilib.load_skymap('REGO', 'GILL', '2018-10-01')

1.5.4 Plot

asilib.plot.plot_keogram.**plot_keogram**(*asi_array_code*, *location_code*, *time_range*, *map_alt=None*, *path=None*, *aacgm=False*, *ax=None*, *color_bounds=None*, *color_norm='lin'*, *title=True*, *pcolormesh_kwargs={}*)

Makes a keogram along the central meridian.

Warning: Use `plot_keogram()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **map_alt** (*int*) – The mapping altitude, in kilometers, used to index the mapped latitude in the skymap calibration data. If None, will plot pixel index for the y-axis.
- **path** (*array*) – Make a keogram along a custom path. Path shape must be (n, 2) and contain the lat/lon coordinates that are mapped to map_alt. If the map_alt kwarg is unspecified, this function will raise a ValueError.
- **aacgm** (*bool*) – Map the keogram latitudes to Altitude Adjusted Corrected Geogmagnetic Coordinates (aacgmv2) derived by Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501-7521, doi:10.1002/2014JA020264.
- **ax** (*plt.Axes*) – The subplot to plot the image on. If None, this function will create one.
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile)
- **color_norm** (*str*) – Sets the linear ('lin') or logarithmic ('log') color normalization.
- **title** (*bool*) – Toggles a default plot title with the format "date ASI_array_code-location_code keogram".
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh. One use of this parameter is to change the colormap. For example, pcolormesh_kwargs = {'cmap':'tu'}

Returns

- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*plt.AxesImage*) – The plt.pcolormesh image object. Common use for im is to add a colorbar.

Raises

AssertionError – If len(time_range) != 2. Also if map_alt does not equal the mapped altitudes in the skymap mapped values.

Example

```
import matplotlib.pyplot as plt

import asilib

asi_array_code='REGO'
location_code='LUCK'
time_range=['2017-09-27T07', '2017-09-27T09']

fig, ax = plt.subplots(figsize=(8, 6))
ax, im = asilib.plot_keogram(asi_array_code, location_code, time_range,
    ax=ax, map_alt=230, color_bounds=(300, 800), pcolormesh_kwargs={'cmap':'turbo'})

plt.colorbar(im)
plt.tight_layout()
plt.show()
```

---

asilib.plot.plot_fisheye.**plot_fisheye**(*asi_array_code*, *location_code*, *time*, *redownload=False*, *time_thresh_s=3*, *ax=None*, *label=True*, *color_map='auto'*, *color_bounds=None*, *color_norm='log'*, *azel_contours=False*)

Plots one fisheye image, oriented with North on the top, and East on the right of the image.

Warning: Use `plot_fisheye()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, `time` must be in the ISO 8601 standard.
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **redownload** (*bool*) – If True, download the file even if it already exists. Useful if a prior data download was incomplete and corrupted.
- **time_thresh_s** (*float*) – The maximum allowable time difference between `time` and an ASI time stamp.
- **ax** (*plt.Axes*) – The subplot to plot the image on. If None, this function will create one.
- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. If 'auto', will default to a black-red colormap for REGO and black-white colormap for THEMIS. For more information See https://matplotlib.org/3.3.3/tutorials/colors/colormaps.html
- **color_bounds** (*List[float]*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile). This range works well for most cases.
- **color_norm** (*str*) – Sets the 'lin' linear or 'log' logarithmic color normalization.

- **azel_contours** (*bool*) – Switch azimuth and elevation contours on or off.

Returns

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.array*) – The 2d ASI image corresponding to image_time.
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*plt.AxesImage*) – The plt.imshow image object. Common use for im is to add a colorbar. The image is oriented in the map orientation (north is up, south is down, west is right, and east is left), contrary to the camera orientation where the east/west directions are flipped. Set azel_contours=True to confirm.

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

```
from datetime import datetime

import matplotlib.pyplot as plt

import asilib

# A bright auroral arc that was analyzed by Imajo et al., 2021 "Active
# auroral arc powered by accelerated electrons from very high altitudes"
time = datetime(2017, 9, 15, 2, 34, 0)
image_time, ax, im = asilib.plot_fisheye('THEMIS', 'RANK', time,
    color_norm='log', redownload=False)

plt.colorbar(im)
ax.axis('off')
plt.show()
```

This module contains functions to project the ASI images to a map.

asilib.plot.plot_map.**plot_map**(*asi_array_code*, *location_code*, *time*, *map_alt*, *time_thresh_s=3*, *ax=None*, *color_map='auto'*, *min_elevation=10*, *norm=True*, *asi_label=True*, *color_bounds=None*, *color_norm='log'*, *pcolormesh_kwargs={}*, *map_shapefile='ne_10m_land'*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *lon_bounds=(-140, -60)*, *lat_bounds=(40, 82)*)

Projects the ASI images to a map at an altitude defined in the skymap calibration file.

> **Warning:** Use `plot_map()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. THEMIS or REGO.
- **location_code** (*str*) – The ASI station code, i.e. ATHA
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, time must be in the ISO 8601 standard.
- **map_alt** (*float*) – The altitude in kilometers to project to. Must be an altitude value in the skymap calibration.
- **time_thresh_s** (*float*) – The maximum allowable time difference between time and an ASI time stamp. This is relevant only when time is specified.
- **ax** (*plt.Axes*) – The subplot to plot the image on. If None, this function will create one.
- **color_map** (*str*) – The matplotlib colormap to use. If 'auto', will default to a black-red colormap for REGO and black-white colormap for THEMIS. For more information See https://matplotlib.org/3.3.3/tutorials/colors/colormaps.html
- **min_elevation** (*float*) – Masks the pixels below min_elevation degrees.
- **norm** (*bool*) – If True, normalizes the image array to 0-1. This is useful when mapping images from multiple imagers.
- **asi_label** (*bool*) – Annotates the map with the ASI code in the center of the image.
- **color_bounds** (*List[float] or None*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile)
- **color_norm** (*str*) – Sets the 'lin' linear or 'log' logarithmic color normalization.
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh. One use of this parameter is to change the colormap. For example, pcolormesh_kwargs = {'cmap':'tu}
- **map_shapefile** (*str or pathlib.Path*) – The path to the shapefile zip archive. If str, it will try to load the shapefile in asilib/data/{file}.
- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.
- **ax** – The subplot to put the map on.
- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.

Returns

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.array*) – The 2d ASI image corresponding to image_time.
- **skyamp** (*dict*) – The skymap calibration for that ASI.
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **p** (*plt.AxesImage*) – The plt.pcolormesh image object. Common use for p is to add a colorbar.

Example

from datetime import datetime

import matplotlib.pyplot as plt

import asilib

asi_array_code = 'THEMIS'
location_code = 'ATHA'
time = datetime(2008, 3, 9, 9, 18, 0)
map_alt_km = 110
asilib.plot_map(asi_array_code, location_code, time, map_alt_km);
plt.show()

---

asilib.plot.plot_map.**make_map**(*file='ne_10m_land'*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *ax=None*, *lon_bounds=(-140, -60)*, *lat_bounds=(40, 82)*)

Makes a map using the mercator projection with a shapefile read in by the pyshp package.

A good place to download shapefiles is https://www.naturalearthdata.com/downloads/10m-physical-vectors/.

Warning: Use *create_map()* instead. This function will be removed in or after December 2023.

Parameters

- **file** (*str or pathlib.Path*) – The path to the shapefile zip archive. If str, it will try to load the shapefile in asilib/data/{file}.
- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.
- **ax** (*plt.Axes*) – The subplot to put the map on.
- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.

Returns
The subplot object containing the map.

Return type
plt.Axes

Example

import asilib

ax = asilib.make_map(lon_bounds=(-127, -100), lat_bounds=(45, 65))

---

asilib.plot.animate_fisheye.**animate_fisheye**(*asi_array_code*, *location_code*, *time_range*, ***kwargs*)

Animate a series of THEMIS or REGO fisheye images.

This function basically runs `animate_fisheye_generator()` in a for loop. The two function's arguments and keyword arguments are identical, so see `animate_fisheye_generator()` docs for the full argument list.

Note: To make movies, you'll need to install `ffmpeg` in your operating system.

Warning: Use `animate_fisheye()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.

Return type
None

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

from datetime import datetime

import asilib

time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
asilib.animate_fisheye('THEMIS', 'FSMI', time_range)
print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "movies"}')

---

asilib.plot.animate_fisheye.**animate_fisheye_generator**(*asi_array_code*, *location_code*, *time_range*, *overwrite=False*, *label=True*, *color_map='auto'*, *color_bounds=None*, *color_norm='log'*, *azel_contours=False*, *ax=None*, *movie_container='mp4'*, *ffmpeg_output_params={}*)

A generator function that loads the ASI data and then yields individual ASI images, image by image. This allows the user to add content to each image, such as the spacecraft position, and that will convert it to a movie. If you just want to make an ASI fisheye movie, use the wrapper for this function, called animate_fisheye().

Once this generator is initiated with the name *gen*, for example, but **before** the for loop, you can get the ASI images and times by calling *gen.send('data')*. This will yield a collections.namedtuple with *time* and *images* attributes.

Warning: Use `animate_fisheye_gen()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **overwrite** (*bool*) – If true, the output animation will be overwritten, otherwise it will prompt the user to answer y/n.
- **label** (*bool*) – Flag to add the "asi_array_code/location_code/image_time" text to the plot.
- **color_map** (*str*) – The matplotlib colormap to use. If 'auto', will default to a black-red colormap for REGO and black-white colormap for THEMIS. For more information See https://matplotlib.org/3.3.3/tutorials/colors/colormaps.html
- **color_bounds** (*List[float] or None*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile)
- **ax** (*plt.Axes*) – The optional subplot that will be drawn on.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_output_params** (*dict*) – The additional/overwitten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **color_norm** (*str*) – Sets the 'lin' linear or 'log' logarithmic color normalization.
- **azel_contours** (*bool*) – Switch azimuth and elevation contours on or off.

Yields

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.ndarray*) – A 2d image array of the image corresponding to image_time
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*plt.AxesImage*) – The plt.imshow image object. Common use for im is to add a colorbar. The image is oriented in the map orientation (north is up, south is down, west is right, and east is left). Set azel_contours=True to confirm.

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".

Example

from datetime import datetime

```
import asilib

time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
movie_generator = asilib.animate_fisheye_generator('THEMIS', 'FSMI', time_range)

for image_time, image, im, ax in movie_generator:
    # The code that modifies each image here.
    pass

print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "movies"}')
```

---

asilib.plot.animate_map.**animate_map**(*asi_array_code*, *location_code*, *time_range*, *map_alt*, ***kwargs*)

Projects a series of THEMIS or REGO images on a map at map_alt altitude in kilometers and animates them.

This function basically runs `animate_map_generator()` in a for loop. The two function's arguments and keyword arguments are identical, so see `animate_map_generator()` docs for the full argument list.

Note: To make animations, you'll need to install `ffmpeg` in your operating system.

Warning: Use `animate_map()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.

Return type
None

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".
- **AssertionError** – If the ASI data exists for that time period, but without time stamps inside time_range.

Example

```
from datetime import datetime

import asilib

map_alt=110 # km
```

time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
asilib.animate_map('THEMIS', 'FSMI', time_range, map_alt=map_alt)
print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "animations"}')

---

asilib.plot.animate_map.**animate_map_generator**(*asi_array_code*, *location_code*, *time_range*, *map_alt*, *min_elevation=10*, *overwrite=False*, *color_map='auto'*, *color_bounds=None*, *color_norm='log'*, *ax=None*, *map_shapefile='ne_10m_land'*, *coast_color='k'*, *land_color='g'*, *ocean_color='w'*, *lon_bounds=(-140, -60)*, *lat_bounds=(40, 82)*, *label=True*, *movie_container='mp4'*, *ffmpeg_output_params={}*, *pcolormesh_kwargs={}*)

Projects a series of THEMIS or REGO images on a map at map_alt altitude in kilometers and animates them. This generator function is useful if you need to superpose other data onto a map in the movie.

Once this generator is initiated with the name *gen*, for example, but **before** the for loop, you can get the ASI images and times by calling *gen.send('data')*. This will yield a collections.namedtuple with *time* and *images* attributes.

Warning: Use `animate_map_gen()` instead. This function will be removed in or after December 2023.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **map_alt** (*float*) – The altitude in kilometers to project to. Must be an altitude value in the skymap calibration.
- **min_elevation** (*float*) – Masks the pixels below min_elevation degrees.
- **overwrite** (*bool*) – If True, the animation will be overwritten. Otherwise it will prompt the user to answer y/n.
- **color_map** (*str*) – The matplotlib colormap to use. If 'auto', will default to a black-red colormap for REGO and black-white colormap for THEMIS. For more information See https://matplotlib.org/3.3.3/tutorials/colors/colormaps.html
- **color_bounds** (*List[float] or None*) – The lower and upper values of the color scale. If None, will automatically set it to low=1st_quartile and high=min(3rd_quartile, 10*1st_quartile)
- **ax** (*plt.Axes*) – The optional subplot that will be drawn on.
- **map_shapefile** (*str or pathlib.Path*) – The path to the shapefile zip archive. If str, it will try to load the shapefile in asilib/data/{file}.
- **coast_color** (*str*) – The coast color. If None will not draw it.
- **land_color** (*str*) – The land color. If None will not draw it.
- **ocean_color** (*str*) – The ocean color. If None will not draw it.

- **lon_bounds** (*tuple*) – The map's longitude bounds.
- **lat_bounds** (*tuple*) – The map's latitude bounds.
- **label** (*bool*) – Annotates the map with the ASI code in the center of the image.
- **movie_container** (*str*) – The movie container: mp4 has better compression but avi was determined to be the official container for preserving digital video by the National Archives and Records Administration.
- **ffmpeg_output_params** (*dict*) – The additional/overwritten ffmpeg output parameters. The default parameters are: framerate=10, crf=25, vcodec=libx264, pix_fmt=yuv420p, preset=slower.
- **color_norm** (*str*) – Sets the 'lin' linear or 'log' logarithmic color normalization.
- **pcolormesh_kwargs** (*dict*) – A dictionary of keyword arguments (kwargs) to pass directly into plt.pcolormesh. One use of this parameter is to change the colormap. For example, pcolormesh_kwargs = {'cmap':'tu}

Yields

- **image_time** (*datetime.datetime*) – The time of the current image.
- **image** (*np.ndarray*) – A 2d image array of the image corresponding to image_time
- **ax** (*plt.Axes*) – The subplot object to modify the axis, labels, etc.
- **im** (*plt.AxesImage*) – The plt.pcolormesh object.

Raises

- **NotImplementedError** – If the colormap is unspecified ('auto' by default) and the auto colormap is undefined for an ASI array.
- **ValueError** – If the color_norm kwarg is not "log" or "lin".
- **AssertionError** – If the ASI data exists for that time period, but without time stamps inside time_range.

Example

```
from datetime import datetime

import asilib

map_alt=110
time_range = (datetime(2015, 3, 26, 6, 7), datetime(2015, 3, 26, 6, 12))
map_generator = asilib.animate_map_generator('THEMIS', 'FSMI', time_range, map_alt=map_alt,
    lon_bounds=(-125, -100), lat_bounds=(55, 70))

for (image_time1, image, ax, p) in map_generator:
    # The code that modifies each image here.
    pass

print(f'Movie saved in {asilib.config["ASI_DATA_DIR"] / "animations"}')
```

1.5.5 Analysis

asilib.analysis.keogram.**keogram**(*asi_array_code*, *location_code*, *time_range*, *map_alt=None*, *path=None*, *aacgm=False*)

Makes a keogram pd.DataFrame along the central meridian.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time_range** (*list of datetime.datetimes or stings*) – Defined the duration of data to download. Must be of length 2.
- **map_alt** (*int*) – The mapping altitude, in kilometers, used to index the mapped latitude in the skymap data. If None, will plot pixel index for the y-axis.
- **path** (*array*) – Make a keogram along a custom path. Path shape must be (n, 2) and contain the lat/lon coordinates that are mapped to map_alt. If the map_alt kwarg is unspecified, this function will raise a ValueError.
- **aacgm** (*bool*) – Map the keogram latitudes to Altitude Adjusted Corrected Geogmagnetic Coordinates (aacgmv2) derived by Shepherd, S. G. (2014), Altitude-adjusted corrected geomagnetic coordinates: Definition and functional approximations, Journal of Geophysical Research: Space Physics, 119, 7501-7521, doi:10.1002/2014JA020264.

Returns

keo – The 2d keogram with the time index. The columns are the geographic latitude if map_alt != None, otherwise it is the image pixel values (0-265) or (0-512).

Return type

pd.DataFrame

Raises

- **AssertionError** – If map_alt does not equal the mapped altitudes in the skymap mapped values.
- **ValueError** – If no images are in `time_range`.
- **ValueError** – If a custom path is provided but not map_alt.

asilib.analysis.map.**lla2azel**(*asi_array_code*, *location_code*, *time*, *sat_lla*)

Maps, a satellite's latitude, longitude, and altitude (LLA) coordinates to the ASI's azimuth and elevation (azel) coordinates and image pixel index.

This function is useful to plot a satellite's location in the ASI image using the pixel indices.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. `THEMIS` or `REGO`.
- **location_code** (*str*) – The ASI station code, i.e. `ATHA`
- **time** (*datetime.datetime or str*) – The date and time to find the relevant skymap file. If str, `time` must be in the ISO 8601 standard.

- **sat_lla** (*np.ndarray or pd.DataFrame*) – The satellite's latitude, longitude, and altitude coordinates in a 2d array with shape (nPosition, 3) where each row is the number of satellite positions to map, and the columns correspond to latitude, longitude, and altitude, respectively. The altitude is in kilometer units.

Returns

- **sat_azel** (*np.ndarray*) – An array with shape (nPosition, 2) of the satellite's azimuth and elevation coordinates.
- **asi_pixels** (*np.ndarray*) – An array with shape (nPosition, 2) of the x- and y-axis pixel indices for the ASI image.

Raises

AssertionError – If the sat_lla argument does not have exactly 3 columns (1st dimension).

Example

from datetime import datetime

import numpy as np
from asilib import lla2azel

# THEMIS/ATHA's LLA coordinates are (54.72, -113.301, 676 (meters)).
# The LLA is a North-South pass right above ATHA..
n = 50
lats = np.linspace(60, 50, n)
lons = -113.64*np.ones(n)
alts = 500**np.ones(n)
lla = np.array([lats, lons, alts]).T

time = datetime(2015, 10, 1) # To load the proper skymap file.

azel, pixels = lla2azel('REGO', 'ATHA', time, lla)

---

asilib.analysis.map.**lla2footprint**(*space_time*, *map_alt*, *b_model='OPQ77'*, *maginput=None*, *hemisphere=0*)

Map the spacecraft's position to `map_alt` along the magnetic field line. The mapping is implemeneted in `IRBEM` and by default it maps to the same hemisphere.

Parameters

- **space_time** (*np.ndarray*) – A 2d array with shape (n_times, 4) with the columns containing the time, latitude, longitude, and altitude coordinates in that order.
- **map_alt** (*float*) – The altitude to map to, in km, in the same hemisphere.
- **b_model** (*str*) – The magnetic field model to use, by default the model is Olson-Pfitzer 1974. This parameter is passed directly into IRBEM.MagFields as the 'kext' parameter.
- **maginput** (*dict*) – If you use a differnet b_model that requires time-dependent parameters, supply the appropriate values to the maginput dictionary. It is directly passed into

IRBEM.MagFields so refer to IRBEM on the proper format.

- **hemisphere** (*int*) – The hemisphere to map to. This kwarg is passed to IRBEM and can be one of these four values: 0 = same magnetic hemisphere as starting point +1 = northern magnetic hemisphere -1 = southern magnetic hemisphere +2 = opposite magnetic hemisphere as starting point

Returns
magnetic_footprint – A numpy.array with size (n_times, 3) with lat, lon, alt columns representing the magnetic footprint coordinates.

Return type
np.ndarray

Raises
ImportError – If IRBEM can't be imported.

asilib.analysis.equal_area.**equal_area**(*asi_array_code*, *location_code*, *time*, *lla*, *box_km=(5, 5)*, *alt_thresh_km=3*)

Calculate a box_km area at the aurora emission altitude.

Parameters

- **asi_array_code** (*str*) – The imager array name, i.e. THEMIS or REGO.
- **location_code** (*str*) – The ASI station code, i.e. ATHA
- **time** (*datetime.datetime or str*) – The date and time to download of the data. If str, time must be in the ISO 8601 standard.
- **lla** (*np.ndarray*) – An array with (n_time, 3) dimensions with the columns representing the latitude, longitude, and altitude (LLA) coordinates.
- **box_size_km** (*iterable*) – Bounds the emission box dimensions in longitude and latitude. Units are kilometers.

Returns
pixel_mask – An array with (n_time, n_x_pixels, n_y_pixels) dimensions with dimensions n_x_pixels and n_y_pixels dimensions the size of each image. Values inside the area are 1 and outside are np.nan.

Return type
np.ndarray

1.6 Contribute

We welcome community support in the form of feature request, bug reports, or directly contributing to asilib.

1.6.1 Bug Reports and Feature Requests

If you find a bug or you have a feature request, feel free to let us know via GitHub's issues. Please select the appropriate issue template to help the developers quickly understand your problem or request, and determine what changes to implement in asilib. Lastly, before suggesting a feature, please read the *Scope* section below to decide if the feature is aligned with the asilib goals.

1.6.2 Installation

To standardize the development packages, run the following commands to reproduce the development environment locally:

```
git clone git@github.com:mshumko/asilib.git
cd asilib
python3 -m pip install -r requirements.txt
```

To develop the docs, you must install Sphinx to your operating system. For linux the command is

```
apt-get install python3-sphinx
```

1.6.3 Adding A New ASI

You can add a new ASI to *asilib* by writing a *wrapper* function that creates and returns an *asilib.Imager* instance. As you read the following interface descriptions, you're welcome to see an example in the *asilib/asi/fake_asi.py* module that contains a *fake_asi()* wrapper function.

The *asilib.Imager* interface consists of four dictionaries:

- *file_info*,
- *skymap*,
- *meta*, and
- *plot_settings* (optional)

file_info dictionary

The *file_info* dictionary provides information on when and how to load ASI images. See the two code snippets below for the required key-value pairs for loading one or multiple images.

One Image

```
file_info = {
    # The time to load the image
    'time': datetime.datetime(),
    # Specify the path the relevant image file. List length is 1.
    'path': List[pathlib.Path],
    # The time of the first image in each file in `path`. List length is 1.
    'start_time': List[datetime.datetime()],
    # The time of the last image in each file in `path`. List length is 1.
    'end_time': List[datetime.datetime()],
    # The function that takes an image path and returns an `np.array()` of
    # `datetime.datetime()` time stamps and an `np.array()` with images. The
```

(continues on next page)

(continued from previous page)

```
    # first dimension of both arrays must correspond to the number of time
    # stamps (1 if there is only one image per file).
    'loader': callable,
}
```

The function specified by the *loader* key is called by *asilib.Imager* when it needs to call the images. This type of function is often called a callback function.

Multiple Images

```
file_info = {
    # The start and end times to load the images. The Tuple length is 2.
    'time_range': Tuple[datetime.datetime()],
    # The paths to all relevant image file. List length is N.
    'path': List[pathlib.Path],
    # The time of the first image in each file in `path`. List length is N.
    'start_time': List[datetime.datetime()],
    # The time of the last image in each file in `path`. List length is N.
    'end_time': List[datetime.datetime()],
    # The function that takes an image path and returns time stamps represented
    # as `datetime.datetime()` and images represented as a `np.array()`.
    'loader': callable,
}
```

The reason that *asilib* needs both the *time_range*, as well as *start_time* and *end_time* is that in general, the *time_range* will not correspond to *start_time[0]* and *end_time[-1]*.

Skymap Dictionary

The *skymap* dictionary provides information on how to orient and map images onto a geographic map. See the code snippet below for the required key-value pairs.

```
skymap = {
        'lat':np.array(...),  # Latitude of pixel vertices.
        'lon':np.array(...),  # Longitude of pixel vertices. In the (-180->180) degree␣
↪range.
        'alt':float,  # The mapping altitude in km.
        'el':np.array(...),   # The elevation of each pixel.
        'az':np.array(...),   # The azimuth of each pixel.
        'path':pathlib.Path(...),  # The path to the skymap file.
    }
```

Meta Dictionary

The *meta* dictionary provides information about the ASI. See the code snippet below for the required key-value pairs.

```
meta = {
    'array': str,  # The ASI array name
    'location': str,  # The ASI location name.
    'lat': float,  # Latitude in units of degrees.
    'lon': float, # Longitude in units of degrees. In the (-180->180) degree range.
```

(continues on next page)

(continued from previous page)

```
    'alt': float,  # Imager altitude in units of km.
    'cadence': float,  # Imager cadence in units of seconds.
    'resolution': (int, int),  # Imager pixel resolution.
}
```

Plot Settings

An optional dictionary that customizes the *asilib.Imager*'s plot settings.

```
plot_settings = {
    # REGO colormap goes from black to red.
    'color_map': matplotlib.colors.LinearSegmentedColormap.from_list('black_to_red', ['k
↪', 'r']),
    'color_norm': 'log',
    # A function that takes in an image and returns the (vmin, vmax) values passed into␣
↪matplotlib.
    'color_bounds': callable
    }
```

1.6.4 Tests

At a bare minimum, your asi loader function needs to include an example in its docstring. Furthermore, this example should also be wrapped up in a test.

See the matplotlib docs on how to create and test functions that create images.

1.6.5 Examples

TODO: Add guidance

1.6.6 Scope

TODO: Add

PYTHON MODULE INDEX

a

INDEX

M

P

R

T