aurora

Release 0.0.1

2021, Karl Kappler, Jared Peacock, Tim Ronan, Andy Frassetto, Li

Jun 04, 2022

USER DOCUMENTATION

AURORA

Aurora is a Python library for processing natural source electromagnetic data. It uses MTH5 formatted Magnetotelluric data and a configuration file as inputs and generates transfer functions that can be formatted as EMTF XML or other output formats.

Documentation for the Aurora project can be found at http://simpeg.xyz/aurora/

We recommend installing Python (>= 3.6) using Anaconda or Miniconda

CHAPTER

ONE

CONDA FORGE

The simplest way to install *aurora* is using conda-forge:

```
conda install -c conda-forge aurora
```

CHAPTER

TWO

PYPI

You can also use pypi to install *aurora*:

```
pip install aurora
```

CHAPTER

THREE

FROM SOURCE

If you would like to install *aurora* from GitHub, you can clone it:

```
git clone https://github.com/simpeg/aurora.git
```

and install it:

```
cd aurora
pip install -e .
```

CHAPTER

FOUR

ISSUES

If you run into installation challenges, please post an issue on GitHub.

CHAPTER

FIVE

BUILD AN MTH5 AND OPERATE THE AURORA PIPELINE

This notebook pulls MT miniSEED data from the IRIS Dataselect web service and produces MTH5 out of it. It outlines the process of making an MTH5 file, generating a processing config, and running the Aurora processor.

It assumes that aurora, mth5, and mt_metadata have all been installed.

In this "new" version, the workflow has changed somewhat.

1. The process_mth5 call works with a dataset dataframe, rather than a single run_id
2. The config object is now based on the mt_metadata.base Base class
3. Remote reference processing is supported (at least in theory)

5.1 Flow of this notebook

Section 1: Here we do imports and construct a table of the data that we will access to build the mth5. Note that there is no explanation here as to the table source – a future update can show how to create such a table from IRIS data_availability tools

Seciton 2: the metadata and the data are accessed, and the mth5 is created and stored.

Section 3:

[1]:
```
# Required imports for the program.
from pathlib import Path
import pandas as pd
from mth5.clients.make_mth5 import MakeMTH5
from mth5 import mth5, timeseries
from mt_metadata.utils.mttime import get_now_utc, MTime
from aurora.config import BANDS_DEFAULT_FILE
from aurora.config.config_creator import ConfigCreator
from aurora.pipelines.process_mth5 import process_mth5
from aurora.tf_kernel.dataset import DatasetDefinition
from aurora.tf_kernel.helpers import extract_run_summary_from_mth5
```

```
2022-06-03 17:47:11,454 [line 135] mth5.setup_logger - INFO: Logging file can be found /
↪home/kkappler/software/irismt/mth5/logs/mth5_debug.log
```

Build an MTH5 file from information extracted by IRIS

[2]:
```
# Set path so MTH5 file builds to current working directory.
default_path = Path().cwd()
default_path
```

[2]:
```
PosixPath('/home/kkappler/software/irismt/aurora/docs/notebooks')
```

[3]:
```
# Initialize the Make MTH5 code.
m = MakeMTH5(mth5_version='0.1.0')
m.client = "IRIS"
```

5.2 Specify the data to access from IRIS

Note that here we explicitly prescribe the data, but this dataframe could be built from IRIS data availability tools in a programatic way

[4]:
```
# Generate data frame of FDSN Network, Station, Location, Channel, Startime, Endtime␣
↪codes of interest

CAS04LQE = ['8P', 'CAS04', '', 'LQE', '2020-06-02T19:00:00', '2020-07-13T19:00:00']
CAS04LQN = ['8P', 'CAS04', '', 'LQN', '2020-06-02T19:00:00', '2020-07-13T19:00:00']
CAS04BFE = ['8P', 'CAS04', '', 'LFE', '2020-06-02T19:00:00', '2020-07-13T19:00:00']
CAS04BFN = ['8P', 'CAS04', '', 'LFN', '2020-06-02T19:00:00', '2020-07-13T19:00:00']
CAS04BFZ = ['8P', 'CAS04', '', 'LFZ', '2020-06-02T19:00:00', '2020-07-13T19:00:00']

request_list = [CAS04LQE, CAS04LQN, CAS04BFE, CAS04BFN, CAS04BFZ]

# Turn list into dataframe
request_df =  pd.DataFrame(request_list, columns=m.column_names)
```

[5]:
```
# Inspect the dataframe
print(request_df)
```

```
  network station location channel                start                  end
0      8P   CAS04              LQE  2020-06-02T19:00:00  2020-07-13T19:00:00
1      8P   CAS04              LQN  2020-06-02T19:00:00  2020-07-13T19:00:00
2      8P   CAS04              LFE  2020-06-02T19:00:00  2020-07-13T19:00:00
3      8P   CAS04              LFN  2020-06-02T19:00:00  2020-07-13T19:00:00
4      8P   CAS04              LFZ  2020-06-02T19:00:00  2020-07-13T19:00:00
```

[6]:
```
# Request the inventory information from IRIS
inventory = m.get_inventory_from_df(request_df, data=False)
```

[7]:
```
# Inspect the inventory
inventory
```

[7]:
```
(Inventory created at 2022-06-04T00:47:20.278242Z
        Created by: ObsPy 1.2.2
                    https://www.obspy.org
        Sending institution: MTH5
        Contains:
                Networks (1):
                        8P
                Stations (1):
                        8P.CAS04 (Corral Hollow, CA, USA)
```

(continues on next page)

(continued from previous page)

```
                Channels (5):
                        8P.CAS04..LFZ, 8P.CAS04..LFN, 8P.CAS04..LFE, 8P.CAS04..LQN,
                        8P.CAS04..LQE,
 0 Trace(s) in Stream:
)
```

Builds an MTH5 file from the user defined database. Note: Intact keeps the MTH5 open after it is done building

With the mth5 object set, we are ready to actually request the data from the fdsn client (IRIS) and save it to an MTH5 file. This process builds an MTH5 file and can take some time depending on how much data is requested.

Note: interact keeps the MTH5 open after it is done building

[8]:
```
interact = True
mth5_object = m.make_mth5_from_fdsnclient(request_df, interact=interact)
```

```
2022-06-03 17:47:31,516 [line 591] mth5.mth5.MTH5.open_mth5 - WARNING: 8P_CAS04.h5 will␣
↪be overwritten in 'w' mode
2022-06-03 17:47:31,939 [line 656] mth5.mth5.MTH5._initialize_file - INFO: Initialized␣
↪MTH5 0.1.0 file /home/kkappler/software/irismt/aurora/docs/notebooks/8P_CAS04.h5 in␣
↪mode w
2022-06-03 17:48:28,522 [line 121] mt_metadata.base.metadata.station.add_run - WARNING:␣
↪Run a is being overwritten with current information
```

```
2022-06-03T17:48:28 [line 136] obspy_stages.create_filter_from_stage - INFO: Converting␣
↪PoleZerosResponseStage electric_si_units to a CoefficientFilter
2022-06-03T17:48:28 [line 136] obspy_stages.create_filter_from_stage - INFO: Converting␣
↪PoleZerosResponseStage electric_dipole_92.000 to a CoefficientFilter
2022-06-03T17:48:28 [line 136] obspy_stages.create_filter_from_stage - INFO: Converting␣
↪PoleZerosResponseStage electric_si_units to a CoefficientFilter
2022-06-03T17:48:28 [line 136] obspy_stages.create_filter_from_stage - INFO: Converting␣
↪PoleZerosResponseStage electric_dipole_92.000 to a CoefficientFilter
More or less runs have been requested by the user than are defined in the metadata. Runs␣
↪will be defined but only the requested run extents contain time series data based on␣
↪the users request.
```

```
2022-06-03 17:48:31,009 [line 224] mth5.timeseries.run_ts.RunTS.validate_metadata -␣
↪WARNING: start time of dataset 2020-06-02T19:00:00+00:00 does not match metadata start␣
↪2020-06-02T18:41:43+00:00 updating metatdata value to 2020-06-02T19:00:00+00:00
2022-06-03 17:48:51,497 [line 462] mth5.timeseries.run_ts.RunTS.from_obspy_stream -␣
↪WARNING: could not find ey
2022-06-03 17:49:04,164 [line 462] mth5.timeseries.run_ts.RunTS.from_obspy_stream -␣
↪WARNING: could not find ex
2022-06-03 17:49:04,272 [line 462] mth5.timeseries.run_ts.RunTS.from_obspy_stream -␣
↪WARNING: could not find ey
2022-06-03 17:49:04,770 [line 235] mth5.timeseries.run_ts.RunTS.validate_metadata -␣
↪WARNING: end time of dataset 2020-07-13T19:00:00+00:00 does not match metadata end␣
↪2020-07-13T21:46:12+00:00 updating metatdata value to 2020-07-13T19:00:00+00:00
```

5.3 Examine and Update the MTH5 object

With the open MTH5 Object, we can start to examine what is in it. For example, we retrieve the filename and file_version. You can additionally do things such as getting the station information and edit it by setting a new value, in this case the declination model.

[9]:
```
mth5_object
```

[9]:
```
/:
====================
    |- Group: Survey
    ----------------
        |- Group: Filters
        -----------------
            |- Group: coefficient
            ---------------------
                |- Group: electric_analog_to_digital
                ------------------------------------
                |- Group: electric_dipole_92.000
                --------------------------------
                |- Group: electric_si_units
                ---------------------------
                |- Group: magnetic_analog_to_digital
                ------------------------------------
            |- Group: fap
            -------------
            |- Group: fir
            -------------
            |- Group: time_delay
            --------------------
                |- Group: electric_time_offset
                ------------------------------
                |- Group: hx_time_offset
                ------------------------
                |- Group: hy_time_offset
                ------------------------
                |- Group: hz_time_offset
                ------------------------
            |- Group: zpk
            -------------
                |- Group: electric_butterworth_high_pass
                ----------------------------------------
                    --> Dataset: poles
                    ...
                    --> Dataset: zeros
                    ...
                |- Group: electric_butterworth_low_pass
                ---------------------------------------
                    --> Dataset: poles
                    ...
                    --> Dataset: zeros
                    ...
                |- Group: magnetic_butterworth_low_pass
```

(continues on next page)

(continued from previous page)

```
            ---------------------------------------
                --> Dataset: poles
                ...
                --> Dataset: zeros
                ...
        |- Group: Reports
        -----------------
        |- Group: Standards
        -------------------
            --> Dataset: summary
            ...
        |- Group: Stations
        ------------------
            |- Group: CAS04
            ---------------
                |- Group: Transfer_Functions
                ----------------------------
                |- Group: a
                -----------
                    --> Dataset: ex
                    ...
                    --> Dataset: ey
                    ...
                    --> Dataset: hx
                    ...
                    --> Dataset: hy
                    ...
                    --> Dataset: hz
                    ...
                |- Group: b
                -----------
                    --> Dataset: ex
                    ...
                    --> Dataset: ey
                    ...
                    --> Dataset: hx
                    ...
                    --> Dataset: hy
                    ...
                    --> Dataset: hz
                    ...
                |- Group: c
                -----------
                    --> Dataset: ex
                    ...
                    --> Dataset: ey
                    ...
                    --> Dataset: hx
                    ...
                    --> Dataset: hy
                    ...
                    --> Dataset: hz
```

(continues on next page)

(continued from previous page)

```
                ...
            |- Group: d
            -----------
                --> Dataset: ex
                ...
                --> Dataset: ey
                ...
                --> Dataset: hx
                ...
                --> Dataset: hy
                ...
                --> Dataset: hz
                ...
        --> Dataset: channel_summary
        ...
        --> Dataset: tf_summary
        ...
```

5.4 Need documentation:

Is the cell below required? Or is it an example of some metadata specification?

[10]:
```
# Edit and update the MTH5 metadata
s = mth5_object.get_station("CAS04")
print(s.metadata.location.declination.model)
s.metadata.location.declination.model = 'IGRF'
print(s.metadata.location.declination.model)
s.write_metadata()    # writes to file mth5_filename
```

```
IGRF-13
IGRF
```

[11]:
```
# Collect information from the MTh5 Object and use it in the config files.
mth5_filename = mth5_object.filename
version = mth5_object.file_version
print(mth5_filename)
```

```
/home/kkappler/software/irismt/aurora/docs/notebooks/8P_CAS04.h5
```

[12]:
```
# Get the available stations and runs from the MTH5 object
mth5_object.channel_summary.summarize()
ch_summary = mth5_object.channel_summary.to_dataframe()
ch_summary
```

[12]:

	survey	station	run	latitude	longitude	elevation	component
0	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	ex
1	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	ey
2	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hx
3	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hy
4	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hz
5	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	ex

(continues on next page)

(continued from previous page)

```
6   CONUS South   CAS04   b  37.633351 -121.468382   329.3875        ey
7   CONUS South   CAS04   b  37.633351 -121.468382   329.3875        hx
8   CONUS South   CAS04   b  37.633351 -121.468382   329.3875        hy
9   CONUS South   CAS04   b  37.633351 -121.468382   329.3875        hz
10  CONUS South   CAS04   c  37.633351 -121.468382   329.3875        ex
11  CONUS South   CAS04   c  37.633351 -121.468382   329.3875        ey
12  CONUS South   CAS04   c  37.633351 -121.468382   329.3875        hx
13  CONUS South   CAS04   c  37.633351 -121.468382   329.3875        hy
14  CONUS South   CAS04   c  37.633351 -121.468382   329.3875        hz
15  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        ex
16  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        ey
17  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hx
18  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hy
19  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hz

                       start                       end  n_samples  \
0  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
1  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
2  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
3  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
4  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
5  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
6  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
7  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
8  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
9  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
10 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
11 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
12 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
13 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
14 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
15 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
16 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
17 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
18 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
19 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586

    sample_rate measurement_type  azimuth  tilt          units  \
0           1.0         electric     13.2   0.0  digital counts
1           1.0         electric    103.2   0.0  digital counts
2           1.0         magnetic     13.2   0.0  digital counts
3           1.0         magnetic    103.2   0.0  digital counts
4           1.0         magnetic      0.0  90.0  digital counts
5           1.0         electric     13.2   0.0  digital counts
6           1.0         electric    103.2   0.0  digital counts
7           1.0         magnetic     13.2   0.0  digital counts
8           1.0         magnetic    103.2   0.0  digital counts
9           1.0         magnetic      0.0  90.0  digital counts
10          1.0         electric     13.2   0.0  digital counts
11          1.0         electric      0.0   0.0          counts
12          1.0         magnetic     13.2   0.0  digital counts
13          1.0         magnetic    103.2   0.0  digital counts
```

(continues on next page)

(continued from previous page)

```
14          1.0         magnetic      0.0  90.0  digital counts
15          1.0         electric      0.0   0.0          counts
16          1.0         electric      0.0   0.0          counts
17          1.0         magnetic     13.2   0.0  digital counts
18          1.0         magnetic    103.2   0.0  digital counts
19          1.0         magnetic      0.0  90.0  digital counts

             hdf5_reference       run_hdf5_reference   station_hdf5_reference
0   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
1   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
2   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
3   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
4   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
5   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
6   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
7   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
8   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
9   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
10  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
11  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
12  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
13  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
14  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
15  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
16  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
17  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
18  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
19  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
```

CHAPTER

SIX

IF MTH5 FILE ALREADY EXISTS YOU CAN START HERE

If you dont want to execute the previous code to get data again

[4]:
```
#interact = False
if interact:
    pass
else:
    h5_path = default_path.joinpath("8P_CAS04.h5")
    mth5_object = mth5.MTH5(file_version="0.1.0")
    #mth5_object.open_mth5(config.stations.local.mth5_path, mode="a")
    mth5_object.open_mth5(h5_path, mode="a")
    ch_summary = mth5_object.channel_summary.to_dataframe()
```

6.1 Generate an Aurora Configuration file using MTH5 as an input

Up to this point, we have used mth5 and mt_metadata, but haven't yet used aurora. So we will use the MTH5 that we just created (and examined and updated) as input into Aurora.

First, we get a list of the available runs to process from the MTH5

[57]:
```
ch_summary
```

[57]:

	survey	station	run	latitude	longitude	elevation	component
0	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	ex
1	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	ey
2	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hx
3	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hy
4	CONUS South	CAS04	a	37.633351	-121.468382	329.3875	hz
5	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	ex
6	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	ey
7	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	hx
8	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	hy
9	CONUS South	CAS04	b	37.633351	-121.468382	329.3875	hz
10	CONUS South	CAS04	c	37.633351	-121.468382	329.3875	ex
11	CONUS South	CAS04	c	37.633351	-121.468382	329.3875	ey
12	CONUS South	CAS04	c	37.633351	-121.468382	329.3875	hx
13	CONUS South	CAS04	c	37.633351	-121.468382	329.3875	hy
14	CONUS South	CAS04	c	37.633351	-121.468382	329.3875	hz

(continues on next page)

(continued from previous page)

```
15  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        ex
16  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        ey
17  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hx
18  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hy
19  CONUS South   CAS04   d  37.633351 -121.468382   329.3875        hz

                       start                       end  n_samples  \
0  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
1  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
2  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
3  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
4  2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00      11267
5  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
6  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
7  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
8  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
9  2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00     847649
10 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
11 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
12 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
13 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
14 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00    1638043
15 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
16 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
17 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
18 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586
19 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00    1034586

    sample_rate measurement_type  azimuth  tilt          units  \
0           1.0         electric     13.2   0.0  digital counts
1           1.0         electric    103.2   0.0  digital counts
2           1.0         magnetic     13.2   0.0  digital counts
3           1.0         magnetic    103.2   0.0  digital counts
4           1.0         magnetic      0.0  90.0  digital counts
5           1.0         electric     13.2   0.0  digital counts
6           1.0         electric    103.2   0.0  digital counts
7           1.0         magnetic     13.2   0.0  digital counts
8           1.0         magnetic    103.2   0.0  digital counts
9           1.0         magnetic      0.0  90.0  digital counts
10          1.0         electric     13.2   0.0  digital counts
11          1.0         electric      0.0   0.0          counts
12          1.0         magnetic     13.2   0.0  digital counts
13          1.0         magnetic    103.2   0.0  digital counts
14          1.0         magnetic      0.0  90.0  digital counts
15          1.0         electric      0.0   0.0          counts
16          1.0         electric      0.0   0.0          counts
17          1.0         magnetic     13.2   0.0  digital counts
18          1.0         magnetic    103.2   0.0  digital counts
19          1.0         magnetic      0.0  90.0  digital counts

             hdf5_reference       run_hdf5_reference   station_hdf5_reference
0   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
```

(continues on next page)

(continued from previous page)

```
1   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
2   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
3   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
4   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
5   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
6   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
7   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
8   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
9   <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
10  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
11  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
12  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
13  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
14  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
15  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
16  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
17  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
18  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
19  <HDF5 object reference>  <HDF5 object reference>  <HDF5 object reference>
```

[17]:
```
available_runs = ch_summary.run.unique()
sr = ch_summary.sample_rate.unique()
if len(sr) != 1:
    print('Only one sample rate per run is available')
available_stations = ch_summary.station.unique()
```

[7]:
```
mth5_object.filename
```

[7]:
```
PosixPath('/home/kkappler/software/irismt/aurora/docs/notebooks/8P_CAS04.h5')
```

[ ]:
```
available_stations[0]
```

6.2 Now, we condense the channel summary into a run_summary.

This method takes an iterable of mth5_objs or h5_paths

6.3 If the mth5 file is closed, you can get the run summary this way

[15]:
```
from aurora.tf_kernel.helpers import extract_run_summaries_from_mth5s
h5_path = default_path.joinpath("8P_CAS04.h5")
run_summary = extract_run_summaries_from_mth5s([h5_path,])
```

```
2022-06-03 17:53:53,098 [line 731] mth5.mth5.MTH5.close_mth5 - INFO: Flushing and␣
↪closing /home/kkappler/software/irismt/aurora/docs/notebooks/8P_CAS04.h5
```

CHAPTER

SEVEN

OTHERWISE, WORK FROM THE MTH5_OBJECT

[16]:
```
run_summary = extract_run_summary_from_mth5(mth5_object)
```

[17]:
```
print(run_summary.columns)
```

```
Index(['station_id', 'run_id', 'start', 'end', 'sample_rate', 'input_channels',
       'output_channels', 'channel_scale_factors', 'mth5_path'],
      dtype='object')
```

[18]:
```
print(run_summary[['station_id', 'run_id', 'start', 'end', ]])
```

```
  station_id run_id                     start                       end
0      CAS04      a 2020-06-02 19:00:00+00:00 2020-06-02 22:07:46+00:00
1      CAS04      b 2020-06-02 22:24:55+00:00 2020-06-12 17:52:23+00:00
2      CAS04      c 2020-06-12 18:32:17+00:00 2020-07-01 17:32:59+00:00
3      CAS04      d 2020-07-01 19:36:55+00:00 2020-07-13 19:00:00+00:00
```

There are no reference stations, set all rows to False

[19]:
```
run_summary["remote"] = False
```

Make an aurora configuration file (and then save that json file.)

[20]:
```
    cc = ConfigCreator()#config_path=CONFIG_PATH)
    config = cc.create_run_processing_object(emtf_band_file=BANDS_DEFAULT_FILE,
                                             sample_rate=sr[0]
                                             )
    config.stations.from_dataset_dataframe(run_summary)
    for decimation in config.decimations:
        decimation.estimator.engine = "RME"
```

Take a look at the config:

[21]:
```
config
```

[21]:
```
{
    "processing": {
        "decimations": [
            {
                "decimation_level": {
                    "anti_alias_filter": "default",
```

(continues on next page)

(continued from previous page)


```
                    "bands": [
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 30,
                                "index_min": 25
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 24,
                                "index_min": 20
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 19,
                                "index_min": 16
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 15,
                                "index_min": 13
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 12,
                                "index_min": 10
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 0,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 9,
```


(continues on next page)

(continued from previous page)


```
                        "index_min": 8
                    }
                },
                {
                    "band": {
                        "decimation_level": 0,
                        "frequency_max": 0,
                        "frequency_min": 0,
                        "index_max": 7,
                        "index_min": 6
                    }
                },
                {
                    "band": {
                        "decimation_level": 0,
                        "frequency_max": 0,
                        "frequency_min": 0,
                        "index_max": 5,
                        "index_min": 5
                    }
                }
            ],
            "decimation.factor": 1.0,
            "decimation.level": 0,
            "decimation.method": "default",
            "decimation.sample_rate": 1.0,
            "estimator.engine": "RME",
            "estimator.estimate_per_channel": true,
            "extra_pre_fft_detrend_type": "linear",
            "input_channels": [
                "hx",
                "hy"
            ],
            "output_channels": [
                "hz",
                "ex",
                "ey"
            ],
            "prewhitening_type": "first difference",
            "reference_channels": [
                "hx",
                "hy"
            ],
            "regression.max_iterations": 10,
            "regression.max_redescending_iterations": 2,
            "regression.minimum_cycles": 10,
            "window.num_samples": 128,
            "window.overlap": 32,
            "window.type": "boxcar"
        }
    },
    {
```


(continues on next page)

(continued from previous page)

```
                "decimation_level": {
                    "anti_alias_filter": "default",
                    "bands": [
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 17,
                                "index_min": 14
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 13,
                                "index_min": 11
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 10,
                                "index_min": 9
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 8,
                                "index_min": 7
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
                                "frequency_min": 0,
                                "index_max": 6,
                                "index_min": 6
                            }
                        },
                        {
                            "band": {
                                "decimation_level": 1,
                                "frequency_max": 0,
```

(continues on next page)

(continued from previous page)

```
                        "frequency_min": 0,
                        "index_max": 5,
                        "index_min": 5
                    }
                }
            ],
            "decimation.factor": 4.0,
            "decimation.level": 1,
            "decimation.method": "default",
            "decimation.sample_rate": 0.25,
            "estimator.engine": "RME",
            "estimator.estimate_per_channel": true,
            "extra_pre_fft_detrend_type": "linear",
            "input_channels": [
                "hx",
                "hy"
            ],
            "output_channels": [
                "hz",
                "ex",
                "ey"
            ],
            "prewhitening_type": "first difference",
            "reference_channels": [
                "hx",
                "hy"
            ],
            "regression.max_iterations": 10,
            "regression.max_redescending_iterations": 2,
            "regression.minimum_cycles": 10,
            "window.num_samples": 128,
            "window.overlap": 32,
            "window.type": "boxcar"
        }
    },
    {
        "decimation_level": {
            "anti_alias_filter": "default",
            "bands": [
                {
                    "band": {
                        "decimation_level": 2,
                        "frequency_max": 0,
                        "frequency_min": 0,
                        "index_max": 17,
                        "index_min": 14
                    }
                },
                {
                    "band": {
                        "decimation_level": 2,
                        "frequency_max": 0,
```

(continues on next page)

(continued from previous page)


```
                    "frequency_min": 0,
                    "index_max": 13,
                    "index_min": 11
                }
            },
            {
                "band": {
                    "decimation_level": 2,
                    "frequency_max": 0,
                    "frequency_min": 0,
                    "index_max": 10,
                    "index_min": 9
                }
            },
            {
                "band": {
                    "decimation_level": 2,
                    "frequency_max": 0,
                    "frequency_min": 0,
                    "index_max": 8,
                    "index_min": 7
                }
            },
            {
                "band": {
                    "decimation_level": 2,
                    "frequency_max": 0,
                    "frequency_min": 0,
                    "index_max": 6,
                    "index_min": 6
                }
            },
            {
                "band": {
                    "decimation_level": 2,
                    "frequency_max": 0,
                    "frequency_min": 0,
                    "index_max": 5,
                    "index_min": 5
                }
            }
        ],
        "decimation.factor": 4.0,
        "decimation.level": 2,
        "decimation.method": "default",
        "decimation.sample_rate": 0.0625,
        "estimator.engine": "RME",
        "estimator.estimate_per_channel": true,
        "extra_pre_fft_detrend_type": "linear",
        "input_channels": [
            "hx",
            "hy"
```


(continues on next page)

(continued from previous page)

```
                ],
                "output_channels": [
                    "hz",
                    "ex",
                    "ey"
                ],
                "prewhitening_type": "first difference",
                "reference_channels": [
                    "hx",
                    "hy"
                ],
                "regression.max_iterations": 10,
                "regression.max_redescending_iterations": 2,
                "regression.minimum_cycles": 10,
                "window.num_samples": 128,
                "window.overlap": 32,
                "window.type": "boxcar"
            }
        },
        {
            "decimation_level": {
                "anti_alias_filter": "default",
                "bands": [
                    {
                        "band": {
                            "decimation_level": 3,
                            "frequency_max": 0,
                            "frequency_min": 0,
                            "index_max": 22,
                            "index_min": 18
                        }
                    },
                    {
                        "band": {
                            "decimation_level": 3,
                            "frequency_max": 0,
                            "frequency_min": 0,
                            "index_max": 17,
                            "index_min": 14
                        }
                    },
                    {
                        "band": {
                            "decimation_level": 3,
                            "frequency_max": 0,
                            "frequency_min": 0,
                            "index_max": 13,
                            "index_min": 10
                        }
                    },
                    {
                        "band": {
```

(continues on next page)

(continued from previous page)


```
                            "decimation_level": 3,
                            "frequency_max": 0,
                            "frequency_min": 0,
                            "index_max": 9,
                            "index_min": 7
                        }
                    },
                    {
                        "band": {
                            "decimation_level": 3,
                            "frequency_max": 0,
                            "frequency_min": 0,
                            "index_max": 6,
                            "index_min": 5
                        }
                    }
                ],
                "decimation.factor": 4.0,
                "decimation.level": 3,
                "decimation.method": "default",
                "decimation.sample_rate": 0.015625,
                "estimator.engine": "RME",
                "estimator.estimate_per_channel": true,
                "extra_pre_fft_detrend_type": "linear",
                "input_channels": [
                    "hx",
                    "hy"
                ],
                "output_channels": [
                    "hz",
                    "ex",
                    "ey"
                ],
                "prewhitening_type": "first difference",
                "reference_channels": [
                    "hx",
                    "hy"
                ],
                "regression.max_iterations": 10,
                "regression.max_redescending_iterations": 2,
                "regression.minimum_cycles": 10,
                "window.num_samples": 128,
                "window.overlap": 32,
                "window.type": "boxcar"
            }
        }
    ],
    "id": "None-None",
    "stations.local.id": "CAS04",
    "stations.local.mth5_path": "/home/kkappler/software/irismt/aurora/docs/
↪notebooks/8P_CAS04.h5",
    "stations.local.remote": false,
```


(continues on next page)

(continued from previous page)


```
        "stations.local.runs": [
            {
                "run": {
                    "id": "a",
                    "input_channels": [
                        {
                            "channel": {
                                "id": "hx",
                                "scale_factor": 1.0
                            }
                        },
                        {
                            "channel": {
                                "id": "hy",
                                "scale_factor": 1.0
                            }
                        }
                    ],
                    "output_channels": [
                        {
                            "channel": {
                                "id": "ex",
                                "scale_factor": 1.0
                            }
                        },
                        {
                            "channel": {
                                "id": "ey",
                                "scale_factor": 1.0
                            }
                        },
                        {
                            "channel": {
                                "id": "hz",
                                "scale_factor": 1.0
                            }
                        }
                    ],
                    "sample_rate": 1.0,
                    "time_periods": [
                        {
                            "time_period": {
                                "end": "2020-06-02T22:07:46+00:00",
                                "start": "2020-06-02T19:00:00+00:00"
                            }
                        }
                    ]
                }
            },
            {
                "run": {
                    "id": "b",
```


(continues on next page)

(continued from previous page)


```
                "input_channels": [
                    {
                        "channel": {
                            "id": "hx",
                            "scale_factor": 1.0
                        }
                    },
                    {
                        "channel": {
                            "id": "hy",
                            "scale_factor": 1.0
                        }
                    }
                ],
                "output_channels": [
                    {
                        "channel": {
                            "id": "ex",
                            "scale_factor": 1.0
                        }
                    },
                    {
                        "channel": {
                            "id": "ey",
                            "scale_factor": 1.0
                        }
                    },
                    {
                        "channel": {
                            "id": "hz",
                            "scale_factor": 1.0
                        }
                    }
                ],
                "sample_rate": 1.0,
                "time_periods": [
                    {
                        "time_period": {
                            "end": "2020-06-12T17:52:23+00:00",
                            "start": "2020-06-02T22:24:55+00:00"
                        }
                    }
                ]
            }
        },
        {
            "run": {
                "id": "c",
                "input_channels": [
                    {
                        "channel": {
                            "id": "hx",
```


(continues on next page)

(continued from previous page)

```
                        "scale_factor": 1.0
                    }
                },
                {
                    "channel": {
                        "id": "hy",
                        "scale_factor": 1.0
                    }
                }
            ],
            "output_channels": [
                {
                    "channel": {
                        "id": "ex",
                        "scale_factor": 1.0
                    }
                },
                {
                    "channel": {
                        "id": "ey",
                        "scale_factor": 1.0
                    }
                },
                {
                    "channel": {
                        "id": "hz",
                        "scale_factor": 1.0
                    }
                }
            ],
            "sample_rate": 1.0,
            "time_periods": [
                {
                    "time_period": {
                        "end": "2020-07-01T17:32:59+00:00",
                        "start": "2020-06-12T18:32:17+00:00"
                    }
                }
            ]
        }
    },
    {
        "run": {
            "id": "d",
            "input_channels": [
                {
                    "channel": {
                        "id": "hx",
                        "scale_factor": 1.0
                    }
                },
                {
```

(continues on next page)

(continued from previous page)


```
                            "channel": {
                                "id": "hy",
                                "scale_factor": 1.0
                            }
                        }
                    ],
                    "output_channels": [
                        {
                            "channel": {
                                "id": "ex",
                                "scale_factor": 1.0
                            }
                        },
                        {
                            "channel": {
                                "id": "ey",
                                "scale_factor": 1.0
                            }
                        },
                        {
                            "channel": {
                                "id": "hz",
                                "scale_factor": 1.0
                            }
                        }
                    ],
                    "sample_rate": 1.0,
                    "time_periods": [
                        {
                            "time_period": {
                                "end": "2020-07-13T19:00:00+00:00",
                                "start": "2020-07-01T19:36:55+00:00"
                            }
                        }
                    ]
                }
            }
        ],
        "stations.remote": []
    }
}
```


Run the Aurora Pipeline using the input MTh5 and Confiugration File

[22]:
```
dataset_definition = DatasetDefinition()
dataset_definition.df = run_summary
dataset_definition.df
```

[22]:

	station_id	run_id	start	end	\
0	CAS04	a	2020-06-02 19:00:00+00:00	2020-06-02 22:07:46+00:00	
1	CAS04	b	2020-06-02 22:24:55+00:00	2020-06-12 17:52:23+00:00	
2	CAS04	c	2020-06-12 18:32:17+00:00	2020-07-01 17:32:59+00:00	
3	CAS04	d	2020-07-01 19:36:55+00:00	2020-07-13 19:00:00+00:00	

(continues on next page)

(continued from previous page)

```
   sample_rate input_channels output_channels  \
0          1.0       [hx, hy]    [ex, ey, hz]
1          1.0       [hx, hy]    [ex, ey, hz]
2          1.0       [hx, hy]    [ex, ey, hz]
3          1.0       [hx, hy]    [ex, ey, hz]

                              channel_scale_factors  \
0  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...
1  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...
2  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...
3  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...

                                          mth5_path  remote
0  /home/kkappler/software/irismt/aurora/docs/not...   False
1  /home/kkappler/software/irismt/aurora/docs/not...   False
2  /home/kkappler/software/irismt/aurora/docs/not...   False
3  /home/kkappler/software/irismt/aurora/docs/not...   False
```

CHAPTER

EIGHT

SELECT THE RUNS YOU WANT TO PROCESS

use the *run_list* variable

[23]:
```
run_list = ['b',]
new_df = dataset_definition.restrict_runs_by_station("CAS04", run_list, overwrite=True)
print(new_df)
dataset_definition.df
```

```
   index station_id run_id                     start  \
0      1      CAS04      b 2020-06-02 22:24:55+00:00

                         end  sample_rate input_channels output_channels  \
0 2020-06-12 17:52:23+00:00          1.0       [hx, hy]    [ex, ey, hz]

                              channel_scale_factors  \
0  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...

                                         mth5_path  remote
0  /home/kkappler/software/irismt/aurora/docs/not...   False
```

[23]:
```
   index station_id run_id                     start  \
0      1      CAS04      b 2020-06-02 22:24:55+00:00

                         end  sample_rate input_channels output_channels  \
0 2020-06-12 17:52:23+00:00          1.0       [hx, hy]    [ex, ey, hz]

                              channel_scale_factors  \
0  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...

                                         mth5_path  remote
0  /home/kkappler/software/irismt/aurora/docs/not...   False
```

[24]:
```
show_plot = True
tf_cls = process_mth5(config,
                      dataset_definition,
                      units="MT",
                      show_plot=show_plot,
                      z_file_path=None,
                      return_collection=False
                  )
```

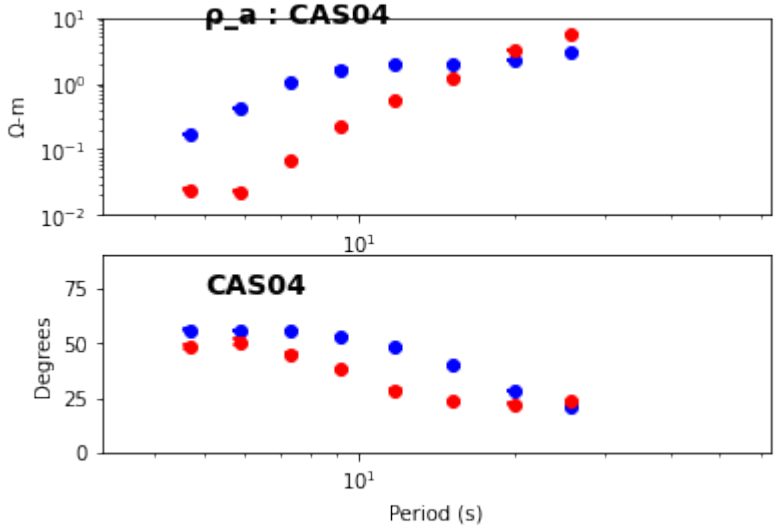
```
Processing config indicates 4 decimation levels
fix this so that it gets from config based on station_id, without caring if local or␣
↪remote
```

(continues on next page)

(continued from previous page)

```
/home/kkappler/software/irismt/aurora/aurora/pipelines/time_series_helpers.py:224:␣
↪RuntimeWarning: invalid value encountered in true_divide
  stft_obj[channel_id].data /= calibration_response
```

```
Processing band 25.728968s
Processing band 19.929573s
Processing band 15.164131s
Processing band 11.746086s
Processing band 9.195791s
Processing band 7.362526s
Processing band 5.856115s
Processing band 4.682492s
GET PLOTTER FROM MTpy
OK, now ser linewidth and markersize
```




```
/home/kkappler/anaconda2/envs/py37/lib/python3.7/site-packages/pandas/core/indexing.py:
↪1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

```
fix this so that it gets from config based on station_id, without caring if local or␣
↪remote
```

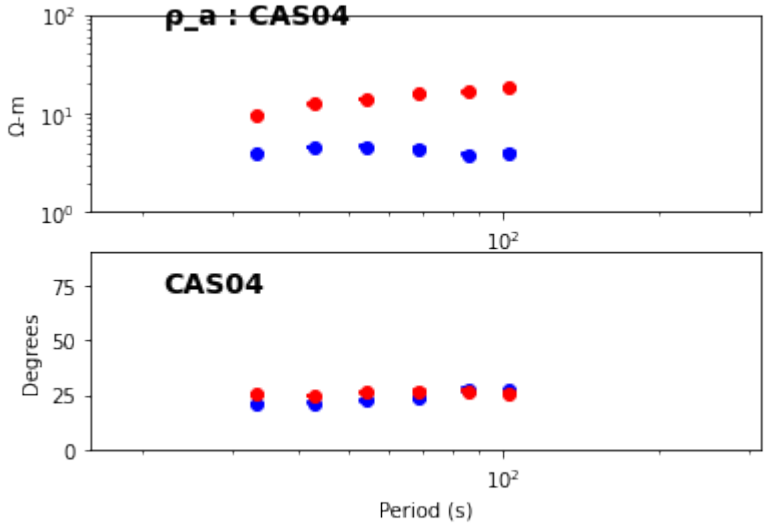
```
/home/kkappler/software/irismt/aurora/aurora/pipelines/time_series_helpers.py:224:␣
↪RuntimeWarning: invalid value encountered in true_divide
  stft_obj[channel_id].data /= calibration_response
```

```
Processing band 102.915872s
Processing band 85.631182s
Processing band 68.881694s
Processing band 54.195827s
Processing band 43.003958s
Processing band 33.310722s
```

(continues on next page)

(continued from previous page)

```
GET PLOTTER FROM MTpy
OK, now ser linewidth and markersize
```



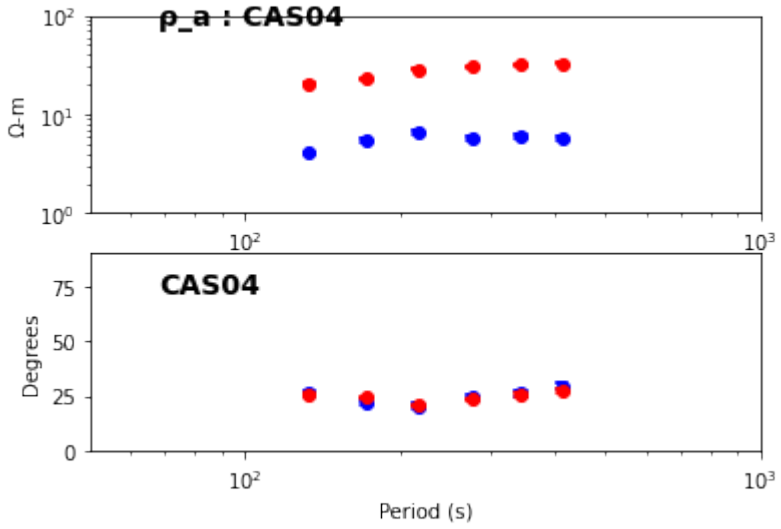

```
/home/kkappler/anaconda2/envs/py37/lib/python3.7/site-packages/pandas/core/indexing.py:
↪1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

```
fix this so that it gets from config based on station_id, without caring if local or␣
↪remote
```

```
/home/kkappler/software/irismt/aurora/aurora/pipelines/time_series_helpers.py:224:␣
↪RuntimeWarning: invalid value encountered in true_divide
  stft_obj[channel_id].data /= calibration_response
```

```
Processing band 411.663489s
Processing band 342.524727s
Processing band 275.526776s
Processing band 216.783308s
Processing band 172.015831s
Processing band 133.242890s
GET PLOTTER FROM MTpy
OK, now ser linewidth and markersize
```



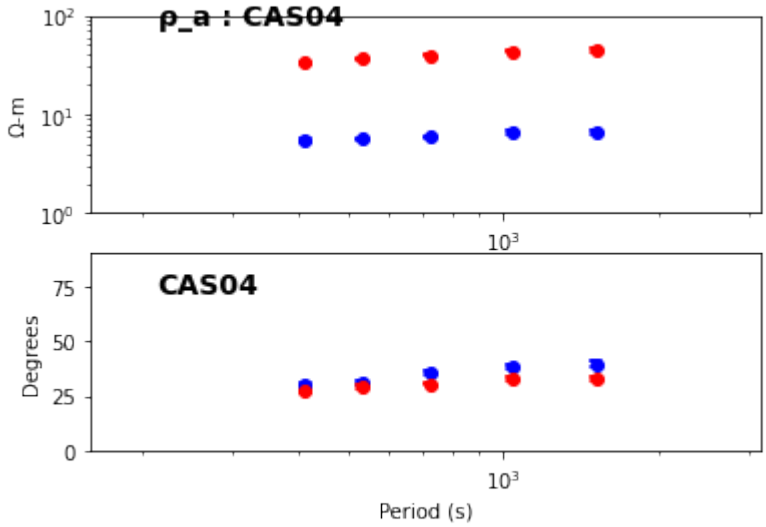

```
/home/kkappler/anaconda2/envs/py37/lib/python3.7/site-packages/pandas/core/indexing.py:
↪1732: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
↪guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_block(indexer, value, name)
```

```
fix this so that it gets from config based on station_id, without caring if local or␣
↪remote
```

```
/home/kkappler/software/irismt/aurora/aurora/pipelines/time_series_helpers.py:224:␣
↪RuntimeWarning: invalid value encountered in true_divide
  stft_obj[channel_id].data /= calibration_response
```

```
Processing band 1514.701336s
Processing band 1042.488956s
Processing band 723.371271s
Processing band 532.971560s
Processing band 412.837995s
GET PLOTTER FROM MTpy
OK, now ser linewidth and markersize
```




```
2022-06-03 17:55:09,287 [line 731] mth5.mth5.MTH5.close_mth5 - INFO: Flushing and␣
↪closing /home/kkappler/software/irismt/aurora/docs/notebooks/8P_CAS04.h5
```

```
['b']
```

[25]:
```
dataset_definition.df
```

```
2022-06-03 17:58:34,548 [line 113] mth5.groups.base.Run.__str__ - WARNING: MTH5 file is␣
↪closed and cannot be accessed.
2022-06-03 17:58:34,830 [line 113] mth5.groups.base.Run.__str__ - WARNING: MTH5 file is␣
↪closed and cannot be accessed.
```

[25]:
```
   index station_id run_id                     start  \
0      1      CAS04      b 2020-06-02 22:24:55+00:00

                        end  sample_rate input_channels output_channels  \
0 2020-06-12 17:52:23+00:00          1.0       [hx, hy]    [ex, ey, hz]

                              channel_scale_factors  \
0  {'ex': 1.0, 'ey': 1.0, 'hx': 1.0, 'hy': 1.0, '...

                                          mth5_path  remote  \
0  /home/kkappler/software/irismt/aurora/docs/not...   False

                                       mth5_obj  \
0  HDF5 file is closed and cannot be accessed.

                                           run  \
0  MTH5 file is closed and cannot be accessed.

                                    run_dataarray  stft
0  [[<xarray.DataArray ()>\narray(-7814.971096500...  None
```

[26]:
```
type(tf_cls)
```

[26]:
```
mt_metadata.transfer_functions.core.TF
```

Write the transfer functions generated by the Aurora pipeline

[27]:
```
tf_cls.write_tf_file(fn="emtfxml_test.xml", file_type="emtfxml")
```

```
2022-06-03 17:58:38,607 [line 197] mt_metadata.transfer_functions.io.readwrite.write_
↪file - INFO: Wrote emtfxml_test.xml
```

[27]:
```
EMTFXML(station='CAS04', latitude=37.63, longitude=-121.47, elevation=329.39)
```

[28]:
```
tf_cls.write_tf_file(fn="emtfxml_test.xml", file_type="edi")
```

```
2022-06-03 17:58:39,327 [line 197] mt_metadata.transfer_functions.io.readwrite.write_
↪file - INFO: Wrote emtfxml_test.xml
```

[28]:
```
EMTFXML(station='CAS04', latitude=37.63, longitude=-121.47, elevation=329.39)
```

[29]:
```
tf_cls.write_tf_file(fn="emtfxml_test.xml", file_type="zmm")
```

```
2022-06-03 17:58:39,876 [line 197] mt_metadata.transfer_functions.io.readwrite.write_
↪file - INFO: Wrote emtfxml_test.xml
```

[29]:
```
EMTFXML(station='CAS04', latitude=37.63, longitude=-121.47, elevation=329.39)
```

CHAPTER

NINE

API REFERENCE

9.1 Time Series

9.1.1 Filters

9.1.2 Apodization Window

@author: kkappler

Module to manage windowing prior to FFT. Intended to support most apodization windows available via scipy.signal.get_window()

Supported Window types = [‘boxcar’, ‘triang’, ‘blackman’, ‘hamming’, ‘hann’, ‘bartlett’, ‘flattop’, ‘parzen’, ‘bohman’, ‘blackmanharris’, ‘nuttall’, ‘barthann’, ‘kaiser’, ‘gaussian’, ‘general_gaussian’, ‘slepian’, ‘chebwin’]

have_additional_args = { ‘kaiser’ : ‘beta’, ‘gaussian’ : ‘std’, ‘general_gaussian’ : (‘power’, ‘width’), ‘slepian’ : ‘width’, ‘chebwin’ : ‘attenuation’,

}

The Taper Config has 2 possible forms: 1. Standard form for accessing scipy.signal: [“taper_family”, “num_samples_window”, “additional_args”] 2. User-defined : for defining custom tapers

Example 1 : Standard form “taper_family” = “hamming” “num_samples_window” = 128 “additional_args” = {}

Example 2 : Standard form “taper_family” = “kaiser” “num_samples_window” = 64 “additional_args” = {“beta”:8}

Examples 3 : User Defined 2. user-defined: [“array”] In this case num_samples_window is defined by the array. “array” = [1, 2, 3, 4, 5, 4, 3, 2, 1] If “array” is non-empty then assume the user-defined case.

It is a little bit unsatisfying that the args need to be ordered for scipy.signal.get_window(). Probably use OrderedDict() for any windows that have more than one additional args.

For example “taper_family” = ‘general_gaussian’ “additional_args” = OrderedDict(“power”:1.5, “sigma”:7)

class aurora.time_series.apodization_window.**ApodizationWindow**(***kwargs*)

Bases: object

Instantiate an apodization window object. Example usages: apod_window = ApodizationWindow() taper=ApodizationWindow(taper_family=’hanning’, num_samples_window=55)

Window factors S1, S2, CG, ENBW are modelled after Heinzel et al. p12-14 [1] Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows. G. Heinzel, A. Roudiger and R. Schilling, Max-Planck Institut fur Gravitationsphysik (Albert-Einstein-Institut) Teilinstitut Hannover February 15, 2002 See Also [2] Harris FJ. On the use of

windows for harmonic analysis with the discrete Fourier transform. Proceedings of the IEEE. 1978 Jan;66(1):51-83.

Nomenclature from Heinzel et al. ENBW: Effective Noise BandWidth, see Equation (22) NENBW Normalized Equivalent Noise BandWidth, see Equation (21)

Parameters

taper_family [string] Specify the taper type - boxcar, kaiser, hanning, etc

num_samples_window [int] The number of samples in the taper

taper [numpy array] The actual window coefficients themselves. This can be passed if a particular custom window is desired.

additional_args: dictionary These are any additional requirements scipy needs in order to generate the window.

Attributes

S1 sum of the window coefficients

S2 sum of squares of the window coefficients

apodization_factor

coherent_gain DC gain of the window normalized by window length

nenbw NENBW Normalized Equivalent Noise BandWidth, see Equation (21) in

num_samples_window

summary Returns

taper

Methods

enbw(fs)	Notes that unlike NENBW, CG, S1, S2, this is not a pure property of the window -- but instead this is a property of the window combined with the sample rate.
make()	this is just a wrapper call to scipy.signal Note: see scipy.signal.get_window for a description of what is expected in args[1:].
test_linear_spectral_density_factor()	This is just a test to verify some algebra Claim: The lsd_calibration factors A (1./coherent_gain)*np.sqrt((2*dt)/(nenbw*N)) and B np.sqrt(2/(sample_rate*self.S2)) are identical.

property S1
sum of the window coefficients

property S2
sum of squares of the window coefficients

property apodization_factor

property coherent_gain
DC gain of the window normalized by window length

enbw(*fs*)
Notes that unlike NENBW, CG, S1, S2, this is not a pure property of the window – but instead this is a property of the window combined with the sample rate. Parameters ————- fs : sampling frequency (1/dt)

Returns

make()
this is just a wrapper call to scipy.signal Note: see scipy.signal.get_window for a description of what is expected in args[1:]. http://docs.scipy.org/doc/scipy/reference/ generated/scipy.signal.get_window.html

note: this is just repackaging the args so that scipy.signal.get_window() accepts all cases.

property nenbw
NENBW Normalized Equivalent Noise BandWidth, see Equation (21) in Heinzel et al 2002

property num_samples_window

property summary

Returns

out_str: str String comprised of the taper_family, number_of_samples, and True/False if self.taper is not None

property taper

test_linear_spectral_density_factor()
This is just a test to verify some algebra Claim: The lsd_calibration factors A (1./coherent_gain)*np.sqrt((2*dt)/(nenbw*N)) and B np.sqrt(2/(sample_rate*self.S2)) are identical.

Note sqrt(2*dt)==sqrt(2*sample_rate) so we can cancel these terms and A=B IFF (1./coherent_gain) * np.sqrt(1/(nenbw*N)) == 1/np.sqrt(S2) which I show in githib aurora issue #3 via . (CG**2) * NENBW *N = S2

Returns

aurora.time_series.apodization_window.**main**()

aurora.time_series.apodization_window.**test_can_inititalize_apodization_window**()

Returns

9.1.3 Decorators

9.1.4 Frequency Band

9.1.5 Frequency Band Helpers

9.1.6 Frequency Domain Helpers

aurora.time_series.frequency_domain_helpers.**get_fft_harmonics**(*samples_per_window*, *sample_rate*, *one_sided=True*)
Works for odd and even number of points. Does not return Nyquist, does return DC component Could be midified with kwargs to support one_sided, two_sided, ignore_dc ignore_nyquist, and etc. Could actally take Frequency-Bands as an argument if we wanted as well.

Parameters

samples_per_window

sample_rate

Returns

9.1.7 Time Axis Helpers

aurora.time_series.time_axis_helpers.**decide_time_axis_method**(*sample_rate*)

aurora.time_series.time_axis_helpers.**do_some_tests**()

aurora.time_series.time_axis_helpers.**fast_arange**(*t0*, *n_samples*, *sample_rate*)

aurora.time_series.time_axis_helpers.**main**()

aurora.time_series.time_axis_helpers.**make_time_axis**(*t0*, *n_samples*, *sample_rate*)

aurora.time_series.time_axis_helpers.**slow_comprehension**(*t0*, *n_samples*, *sample_rate*)

aurora.time_series.time_axis_helpers.**test_generate_time_axis**(*t0*, *n_samples*, *sample_rate*)

Two obvious ways to generate an axis of timestanps here. One method is slow and more precise, the other is fast but drops some nanoseconds due to integer roundoff error.

To see this, consider the example of say 3Hz, we are 333333333ns between samples, which drops 1ns per second if we scale a nanoseconds=np.arange(N)

The issue here is that the nanoseconds granularity forces a roundoff error,

Probably will use logic like: if there_are_integer_ns_per_sample:

time_stamps = do_it_the_fast_way()

else: time_stamps = do_it_the_slow_way()

return time_stamps

Parameters

t0

n_samples

sample_rate

Returns

9.1.8 Window Helpers

Notes in google doc: https://docs.google.com/document/d/1CsRhSLXsRG8HQxM4lKNqVj-V9KA9iUQAvCOtouVzFs0/edit?usp=sharing

aurora.time_series.window_helpers.**apply_fft_to_windowed_array**(*windowed_array*)
This will operate row-wise as well Parameters ————- windowed_array

Returns

aurora.time_series.window_helpers.**available_number_of_windows_in_array**(*n_samples_array*, *n_samples_window*, *n_advance*)

Parameters

n_samples_array: int The length of the time series

n_samples_window: int The length of the window (in samples)

n_advance: int The number of samples the window advances at each step

Returns

available_number_of_strides: int The number of windows the time series will yield

aurora.time_series.window_helpers.**check_all_sliding_window_functions_are_equivalent**()
simple sanity check that runs each sliding window function on a small array and confirms the results are numerically identical. Note that striding window will return int types where others return float. Returns ——-

aurora.time_series.window_helpers.**do_some_tests**()

aurora.time_series.window_helpers.**main**()

aurora.time_series.window_helpers.**sliding_window_crude**(*data*, *num_samples_window*, *num_samples_advance*, *num_windows=None*)

Parameters

data: np.ndarray The time series data to be windowed

num_samples_window: int The length of the window (in samples)

num_samples_advance: int The number of samples the window advances at each step

num_windows: int The number of windows to "take". Must be less or equal to the number of available windows.

Returns

output_array: numpy.ndarray The windowed time series

aurora.time_series.window_helpers.**sliding_window_numba**(*data*, *num_samples_window*, *num_samples_advance*, *num_windows*)

Parameters

data: np.ndarray The time series data to be windowed

num_samples_window: int The length of the window (in samples)

num_samples_advance: int The number of samples the window advances at each step

num_windows: int The number of windows to "take".

Returns

output_array: numpy.ndarray The windowed time series

aurora.time_series.window_helpers.**striding_window**(*data*, *num_samples_window*, *num_samples_advance*, *num_windows=None*)

Applies a striding window to an array. We use 1D arrays here. Note that this method is extendable to N-dimensional arrays as was once shown at http://www.johnvinyard.com/blog/?p=268

Karl has an implementation of this code but chose to restict to 1D here. This is becuase of several warnings encountered, on the notes of stride_tricks.py, as well as for example here: https://stackoverflow.com/questions/4936620/using-strides-for-an-efficient-moving-average-filter

While we can possibly setup Aurora so that no copies of the strided window are made downstream, we cannot guarantee that another user may not add methods that require copies. For robustness we will use 1d implementation only for now.

Another clean example of this method can be found in the razorback codes from brgm.

result is 2d: result[i] is the i-th window

```
>>> sliding_window(np.arange(15), 4, 3, 2)
array([[0, 1, 2],
       [2, 3, 4],
       [4, 5, 6],
       [6, 7, 8]])
```

Parameters

data: np.ndarray The time series data to be windowed

num_samples_window: int The length of the window (in samples)

num_samples_advance: int The number of samples the window advances at each step

num_windows: int The number of windows to "take". Must be less or equal to the number of available windows.

Returns

strided_window: numpy.ndarray The windowed time series

aurora.time_series.window_helpers.**test_apply_taper**()

9.1.9 Windowed Time Series

9.1.10 Windowing Scheme

9.2 Transfer Function

9.2.1 Plot

9.2.2 Regression

9.2.3 Base

9.2.4 EMTF Z File Helpers

These methods can possibly be moved under mt_metadata, or mth5

They extract info needed to setup emtf_z files.

aurora.transfer_function.emtf_z_file_helpers.**clip_bands_from_z_file**(*z_path*, *n_bands_clip*, *output_z_path=None*, *n_sensors=5*)

This function takes a z_file and clips periods off the end of it. It can come in handy sometimes – specifically for manipulating matlab results of synthetic data.

Parameters

z_path: Path or str path to the z_file to read in and clip periods from

n_periods_clip: integer how many periods to clip from the end of the zfile

overwrite: bool whether to overwrite the zfile or rename it

n_sensors

Returns

aurora.transfer_function.emtf_z_file_helpers.**get_default_orientation_block**(*n_ch=5*)

Helper function used when working with matlab structs which do not have enough info to make headers

Parameters

n_ch: int number of channels at the station

Returns

orientation_strs: list List of text strings, one per channel

aurora.transfer_function.emtf_z_file_helpers.**make_orientation_block_of_z_file**(*run_obj*, *channel_list=None*)

Replicates emtz z-file metadata about orientation like this: 1 0.00 0.00 tes Hx 2 90.00 0.00 tes Hy 3 0.00 0.00 tes Hz 4 0.00 0.00 tes Ex 5 90.00 0.00 tes Ey

based on this fortran snippet: write(3, 115) k, orient(1, k), orient(2, k), stname(1: 3), chid(k)

format(i5, 1x, f8.2, 1x, f8.2, 1x, a3, 2x, a6) #Fortran Format

Parameters

run_obj: mth5.groups.master_station_run_channel.RunGroup Container with metadata about the channels

Returns

output_strings: list (of strings) Each element of the list corresponds to one channel, and is a block of text for the emtf z-file with the channel orientation, name and associated station

aurora.transfer_function.emtf_z_file_helpers.**merge_tf_collection_to_match_z_file**(*aux_data*, *tf_collection*)

Currently this is only used for the synthtetic test, but maybe useful for other tests. Given data from a z_file, and a tf_collection, the tf_collection may have several TF estimates at the same frequency from multiple decimation levels. This tries to make a single array as a function of period for all rho and phi

Parameters

aux_data: merge_tf_collection_to_match_z_file Object representing a z-file

tf_collection: aurora.transfer_function.transfer_function_collection

.TransferFunctionCollection Object representing the transfer function returnd from the aurora processing

Returns

result: dict of dicts Keyed by ["rho", "phi"], below each of these is an ["xy", "yx",] entry. The lowest level entries are numpy arrays.

9.2.5 Iter Control

follows Gary's IterControl.m in iris_mt_scratch/egbert_codes-20210121T193218Z-001/egbert_codes/matlabPrototype_10-13-20/TF/classes

class aurora.transfer_function.regression.iter_control.**IterControl**(*max_number_of_iterations=10*, *max_number_of_redescending_iterations=2*, ***kwargs*)

Bases: object

Attributes

continue_redescending

correction_factor TODO: This is an RME specific property.

Methods

converged(b, b0)

Parameters

property continue_redescending

converged(*b*, *b0*)

Parameters

b [complex-valued numpy array] the most recent regression estimate

b0 [complex-valued numpy array] The previous regression estimate

Returns

converged: bool True of the regression has terminated, False otherwise

Notes:

The variable maximum_change finds the maximum amplitude component of the vector

1-b/b0. Looking at the formula, one might want to cast this instead as

1 - abs(b/b0), however, that will be insensitive to phase changes in b,

which is complex valued. The way it is coded np.max(np.abs(1 - b / b0)) is

correct as it stands.

property correction_factor

TODO: This is an RME specific property. Suggest move r0, u0 and this method into an RME-config class.

See notes on usage in transfer_function.regression.helper_functions.rme_beta

Returns

correction_factor [float] correction factor used for scaling the residual error_variance

9.2.6 Transfer Function Collection

9.2.7 Transfer Function Header

follows Gary's TFHeader.m iris_mt_scratch/egbert_codes-20210121T193218Z-001/egbert_codes/matlabPrototype_10-13-20/TF/classes

class aurora.transfer_function.transfer_function_header.**TransferFunctionHeader**(***kwargs*)

Bases: object

class for storing metadata for a TF estimate

This class should inherit the metadata from the remote and reference stations. As of 2021-07-20 the class functions with only the station_id and channel_id values.

<See Issue #41> </See Issue #41>

Attributes

local_channels

local_station

local_station_id

num_input_channels

num_output_channels

reference_station_id

property local_channels

property local_station

property local_station_id

property num_input_channels

property num_output_channels

property reference_station_id

9.2.8 TTFZ

9.3 Pipelines

9.3.1 Process MTH5

9.3.2 Time Series Helpers

9.3.3 Transfer Function Helpers

9.4 Interval

Source https://github.com/internetarchive/analyze_ocr/blob/master/interval.py Liscence Not Specified

Provides the Interval and IntervalSet classes

The interval module provides the Interval and IntervalSet data types. Intervals describe continuous ranges that can be open, closed, half-open, or infinite. IntervalSets contain zero to many disjoint sets of Intervals.

Intervals don't have to pertain to numbers. They can contain any data that is comparable via the Python operators <, <=, ==, >=, and >. Here's an example of how strings can be used with Intervals:

```
>>> volume1 = Interval.between("A", "Foe")
>>> volume2 = Interval.between("Fog", "McAfee")
>>> volume3 = Interval.between("McDonalds", "Space")
>>> volume4 = Interval.between("Spade", "Zygote")
>>> encyclopedia = IntervalSet([volume1, volume2, volume3, volume4])
>>> mySet = IntervalSet([volume1, volume3, volume4])
>>> "Meteor" in encyclopedia
True
>>> "Goose" in encyclopedia
True
>>> "Goose" in mySet
False
>>> volume2 in (encyclopedia ^ mySet)
True
```

Here's an example of how times can be used with Intervals:

```
>>> officeHours = IntervalSet.between("08:00", "17:00")
>>> myLunch = IntervalSet.between("11:30", "12:30")
>>> myHours = IntervalSet.between("08:30", "19:30") - myLunch
>>> myHours.issubset(officeHours)
False
>>> "12:00" in myHours
False
>>> "15:30" in myHours
True
>>> inOffice = officeHours & myHours
>>> print inOffice
['08:30'..'11:30'),('12:30'..'17:00']
>>> overtime = myHours - officeHours
```

(continues on next page)

(continued from previous page)

```
>>> print overtime
('17:00'..'19:30']
```

class aurora.interval.**BaseIntervalSet**(*items=[]*)

Bases: object

Base class for IntervalSet and FrozenIntervalSet.

Methods

all()	Returns an interval set containing all values
between(a, b[, closed])	Returns an IntervalSet of all values between a and b.
bounds()	Returns an interval that encompasses the entire BaseIntervalSet
copy()	Returns a copy of the object
difference(other)	Returns the difference between the object and the given object
empty()	Returns an interval set containing no values.
greater_than(n)	Returns an IntervalSet containing values greater than the given value
greater_than_or_equal_to(n)	Returns an IntervalSet containing values greater than or equal to the given value
intersection(other)	Returns the intersection between the object and the given value
issubset(other)	Tells if the given object is a subset of the object
issuperset(other)	Tells whether the given object is a superset of the object
less_than(n)	Returns an IntervalSet containing values less than the given value
less_than_or_equal_to(n[, closed])	Returns an IntervalSet containing values less than or equal to the given value
lower_bound()	Returns the lower boundary of the BaseIntervalSet
lower_closed()	Returns a boolean telling whether the lower bound is closed or not
not_equal_to(n)	Returns an IntervalSet of all values not equal to n
symmetric_difference(other)	Returns the exclusive or of the given value with the object
union(other)	Returns the union of the given value with the object
upper_bound()	Returns the upper boundary of the BaseIntervalSet
upper_closed()	Returns a boolean telling whether the upper bound is closed or not

classmethod all()

Returns an interval set containing all values

```
>>> print IntervalSet.all()
(...)
```

classmethod between(*a*, *b*, *closed=True*)

Returns an IntervalSet of all values between a and b.

If closed is True, then the endpoints are included; otherwise, they aren't.

```
>>> print IntervalSet.between(0, 100)
[0..100]
>>> print IntervalSet.between(-1, 1)
[-1..1]
```

bounds()

Returns an interval that encompasses the entire BaseIntervalSet

```
>>> print IntervalSet([Interval.between(4, 6), 2, 12]).bounds()
[2..12]
>>> print IntervalSet().bounds()
<Empty>
>>> print IntervalSet.all().bounds()
(...)
```

copy()

Returns a copy of the object

```
>>> s = IntervalSet(
...   [7, 2, 3, 2, 6, 2, Interval.greater_than(3)])
>>> s2 = s.copy()
>>> s == s2
True
>>> s = FrozenIntervalSet(
...   [7, 2, 3, 2, 6, 2, Interval.greater_than(3)])
>>> s2 = s.copy()
>>> s == s2
True
```

difference(*other*)

Returns the difference between the object and the given object

Returns all values of self minus all matching values in other. It is identical to the - operator, only it accepts any iterable as the operand.

```
>>> negatives = IntervalSet.less_than(0)
>>> positives = IntervalSet.greater_than(0)
>>> naturals  = IntervalSet.greater_than_or_equal_to(0)
>>> evens     = IntervalSet([-8, -6, -4, -2, 0, 2, 4, 6, 8])
>>> zero      = IntervalSet([0])
>>> nonzero   = IntervalSet.not_equal_to(0)
>>> empty     = IntervalSet.empty()
>>> all       = IntervalSet.all()
>>> print evens.difference(nonzero)
0
>>> print empty.difference(naturals)
<Empty>
>>> print zero.difference(naturals)
<Empty>
>>> print positives.difference(zero)
(0...)
>>> print naturals.difference(negatives)
[0...)
```

(continues on next page)

(continued from previous page)

```
>>> print all.difference(zero)
(...0),(0...)
>>> all.difference(zero) == nonzero
True
>>> naturals.difference([0]) == positives
True
```

classmethod empty()

Returns an interval set containing no values.

```
>>> print IntervalSet.empty()
<Empty>
```

classmethod greater_than(*n*)

Returns an IntervalSet containing values greater than the given value

```
>>> print IntervalSet.greater_than(0)
(0...)
>>> print IntervalSet.greater_than(-23)
(-23...)
```

classmethod greater_than_or_equal_to(*n*)

Returns an IntervalSet containing values greater than or equal to the given value

```
>>> print IntervalSet.greater_than_or_equal_to(0)
[0...)
>>> print IntervalSet.greater_than_or_equal_to(-23)
[-23...)
```

intersection(*other*)

Returns the intersection between the object and the given value

This function returns the intersection of self and other. It is identical to the & operator, except this function accepts any iterable as an operand, and & accepts only another BaseIntervalSet.

```
>>> negatives = IntervalSet.less_than(0)
>>> positives = IntervalSet.greater_than(0)
>>> naturals  = IntervalSet.greater_than_or_equal_to(0)
>>> evens     = IntervalSet([-8, -6, -4, -2, 0, 2, 4, 6, 8])
>>> zero      = IntervalSet([0])
>>> nonzero   = IntervalSet.not_equal_to(0)
>>> empty     = IntervalSet.empty()
>>> print naturals.intersection(naturals)
[0...)
>>> print evens.intersection(zero)
0
>>> print negatives.intersection(zero)
<Empty>
>>> print nonzero.intersection(positives)
(0...)
>>> print empty.intersection(zero)
<Empty>
```

issubset(*other*)

Tells if the given object is a subset of the object

Returns true if self is a subset of other. other can be any iterable object.

```
>>> zero = IntervalSet([0])
>>> positives = IntervalSet.greater_than(0)
>>> naturals = IntervalSet.greater_than_or_equal_to(0)
>>> negatives = IntervalSet.less_than(0)
>>> r = IntervalSet.between(3, 6)
>>> r2 = IntervalSet.between(-8, -2)
>>> zero.issubset(positives)
False
>>> zero.issubset(naturals)
True
>>> positives.issubset(zero)
False
>>> r.issubset(zero)
False
>>> r.issubset(positives)
True
>>> positives.issubset(r)
False
>>> negatives.issubset(IntervalSet.all())
True
>>> r2.issubset(negatives)
True
>>> negatives.issubset(positives)
False
>>> zero.issubset([0, 1, 2, 3])
True
```

issuperset(*other*)

Tells whether the given object is a superset of the object

Returns true if self is a superset of other. other can be any iterable object.

```
>>> zero = IntervalSet([0])
>>> positives = IntervalSet.greater_than(0)
>>> naturals = IntervalSet.greater_than_or_equal_to(0)
>>> negatives = IntervalSet.less_than(0)
>>> r = IntervalSet.between(3, 6)
>>> r2 = IntervalSet.between(-8, -2)
>>> zero.issuperset(positives)
False
>>> zero.issuperset(naturals)
False
>>> positives.issuperset(zero)
False
>>> r.issuperset(zero)
False
>>> r.issuperset(positives)
False
>>> positives.issuperset(r)
True
```

(continues on next page)

(continued from previous page)

```
>>> negatives.issuperset(IntervalSet.all())
False
>>> r2.issuperset(negatives)
False
>>> negatives.issuperset(positives)
False
>>> negatives.issuperset([-2, -632])
True
```

classmethod less_than(*n*)

Returns an IntervalSet containing values less than the given value

```
>>> print IntervalSet.less_than(0)
(...0)
>>> print IntervalSet.less_than(-23)
(...-23)
```

classmethod less_than_or_equal_to(*n*, *closed=False*)

Returns an IntervalSet containing values less than or equal to the given value

```
>>> print IntervalSet.less_than_or_equal_to(0)
(...0]
>>> print IntervalSet.less_than_or_equal_to(-23)
(...-23]
```

lower_bound()

Returns the lower boundary of the BaseIntervalSet

```
>>> IntervalSet([Interval.between(4, 6), 2, 12]).lower_bound()
2
>>> IntervalSet().lower_bound()
Traceback (most recent call last):
    ...
IndexError: The BaseIntervalSet is empty
>>> IntervalSet.all().lower_bound()
-Inf
```

lower_closed()

Returns a boolean telling whether the lower bound is closed or not

```
>>> IntervalSet([Interval.between(4, 6), 2, 12]).lower_closed()
True
>>> IntervalSet().lower_closed()
Traceback (most recent call last):
    ...
IndexError: The BaseIntervalSet is empty
>>> IntervalSet.all().lower_closed()
False
```

classmethod not_equal_to(*n*)

Returns an IntervalSet of all values not equal to n

```
>>> print IntervalSet.not_equal_to(0)
(...0),(0...)
>>> print IntervalSet.not_equal_to(-23)
(...-23),(-23...)
```

symmetric_difference(*other*)

Returns the exclusive or of the given value with the object

This function returns the exclusive or of two IntervalSets. It is identical to the ^ operator, except it accepts any iterable object for the operand.

```
>>> negatives = IntervalSet.less_than(0)
>>> positives = IntervalSet.greater_than(0)
>>> naturals  = IntervalSet.greater_than_or_equal_to(0)
>>> evens     = IntervalSet([-8, -6, -4, -2, 0, 2, 4, 6, 8])
>>> zero      = IntervalSet([0])
>>> nonzero   = IntervalSet.not_equal_to(0)
>>> empty     = IntervalSet.empty()
>>> print nonzero.symmetric_difference(naturals)
(...0]
>>> print zero.symmetric_difference(negatives)
(...0]
>>> print positives.symmetric_difference(empty)
(0...)
>>> print evens.symmetric_difference(zero)
-8,-6,-4,-2,2,4,6,8
>>> print evens.symmetric_difference(range(0, 9, 2))
-8,-6,-4,-2
```

union(*other*)

Returns the union of the given value with the object

This function returns the union of a BaseIntervalSet and an iterable object. It is identical to the | operator, except that | only accepts a BaseIntervalSet operand and union accepts any iterable.

```
>>> negatives = IntervalSet.less_than(0)
>>> positives = IntervalSet.greater_than(0)
>>> naturals  = IntervalSet.greater_than_or_equal_to(0)
>>> evens     = IntervalSet([-8, -6, -4, -2, 0, 2, 4, 6, 8])
>>> zero      = IntervalSet([0])
>>> nonzero   = IntervalSet.not_equal_to(0)
>>> empty     = IntervalSet.empty()
>>> all       = IntervalSet.all()
>>> print evens.union(positives)
-8,-6,-4,-2,[0...)
>>> print negatives.union(zero)
(...0]
>>> print empty.union(negatives)
(...0)
>>> print empty.union(naturals)
[0...)
>>> print nonzero.union(evens)
(...)
>>> print negatives.union(range(5))
```

(continues on next page)

(continued from previous page)

```
(...0],1,2,3,4
```

upper_bound()

Returns the upper boundary of the BaseIntervalSet

```
>>> IntervalSet([Interval.between(4, 6), 2, 12]).upper_bound()
12
>>> IntervalSet().upper_bound()
Traceback (most recent call last):
    ...
IndexError: The BaseIntervalSet is empty
>>> IntervalSet.all().upper_bound()
Inf
```

upper_closed()

Returns a boolean telling whether the upper bound is closed or not

```
>>> IntervalSet([Interval.between(4, 6), 2, 12]).upper_closed()
True
>>> IntervalSet().upper_closed()
Traceback (most recent call last):
    ...
IndexError: The BaseIntervalSet is empty
>>> IntervalSet.all().upper_closed()
False
```

class aurora.interval.**FrozenIntervalSet**(*items=[]*)

Bases: *aurora.interval.BaseIntervalSet*

An immutable version of BaseIntervalSet

FrozenIntervalSet is like IntervalSet, only add and remove are not implemented, and hashes can be generated.

```
>>> fs = FrozenIntervalSet([3, 6, 2, 4])
>>> fs.add(12)
Traceback (most recent call last):
  ...
AttributeError: 'FrozenIntervalSet' object has no attribute 'add'
>>> fs.remove(4)
Traceback (most recent call last):
  ...
AttributeError: 'FrozenIntervalSet' object has no attribute 'remove'
>>> fs.clear()
Traceback (most recent call last):
  ...
AttributeError: 'FrozenIntervalSet' object has no attribute 'clear'
```

Because FrozenIntervalSets are immutable, they can be used as a dictionary key.

```
>>> d = {
...   FrozenIntervalSet([3, 66]) : 52,
...   FrozenIntervalSet.less_than(3) : 3}
```

Methods

all()	Returns an interval set containing all values
between(a, b[, closed])	Returns an IntervalSet of all values between a and b.
bounds()	Returns an interval that encompasses the entire BaseIntervalSet
copy()	Duplicates the object
difference(other)	Returns the difference between the object and the given object
empty()	Returns an interval set containing no values.
greater_than(n)	Returns an IntervalSet containing values greater than the given value
greater_than_or_equal_to(n)	Returns an IntervalSet containing values greater than or equal to the given value
intersection(other)	Returns the intersection between the object and the given value
issubset(other)	Tells if the given object is a subset of the object
issuperset(other)	Tells whether the given object is a superset of the object
less_than(n)	Returns an IntervalSet containing values less than the given value
less_than_or_equal_to(n[, closed])	Returns an IntervalSet containing values less than or equal to the given value
lower_bound()	Returns the lower boundary of the BaseIntervalSet
lower_closed()	Returns a boolean telling whether the lower bound is closed or not
not_equal_to(n)	Returns an IntervalSet of all values not equal to n
symmetric_difference(other)	Returns the exclusive or of the given value with the object
union(other)	Returns the union of the given value with the object
upper_bound()	Returns the upper boundary of the BaseIntervalSet
upper_closed()	Returns a boolean telling whether the upper bound is closed or not

copy()

Duplicates the object

For FrozenIntervalSet objects, since they're immutable, a reference, not a copy, of self is returned.

```
>>> s = FrozenIntervalSet(
...   [7, 2, 3, 2, 6, 2, Interval.greater_than(3)])
>>> s2 = s.copy()
>>> s == s2
True
>>> id(s) == id(s2)
True
```

class aurora.interval.**Interval**(*lower_bound=- Inf*, *upper_bound=Inf*, ***kwargs*)

Bases: object

Represents a continuous range of values

An Interval is composed of the lower bound, a closed lower bound flag, an upper bound, and a closed upper bound flag. The attributes are called lower_bound, lower_closed, upper_bound, and upper_closed, respectively.

For an infinite interval, the bound is set to inf or -inf. IntervalSets are composed of zero to many Intervals.

Methods

adjacent_to(other)	Tells whether an Interval is adjacent to the object without overlap
all()	Returns an interval encompassing all values
between(a, b[, closed])	Returns an interval between two values
comes_before(other)	Tells whether an interval lies before the object
duration()	@note: added 20140618 kkappler returns the duration of interval
equal_to(a)	Returns an point interval
greater_than(a)	Returns interval of all values greater than the given value
greater_than_or_equal_to(a)	Returns interval of all values greater than or equal to the given value
join(other)	Combines two continous Intervals
less_than(a)	Returns interval of all values less than the given value
less_than_or_equal_to(a)	Returns an interval containing the given values and everything less
none()	Returns an empty interval
overlaps(other)	Tells whether the given interval overlaps the object
pad(pad_width)	@note: added 20140618 kkappler returns a padded interval; Intent was to add a little so that open-intervals, or adjacent intervals would easily make an interval set

adjacent_to(*other*)

Tells whether an Interval is adjacent to the object without overlap

Returns True if self is adjacent to other, meaning that if they were joined, there would be no discontinuity. They cannot overlap.

```
>>> r1  = Interval.less_than(-100)
>>> r2  = Interval.less_than_or_equal_to(-100)
>>> r3  = Interval.less_than(100)
>>> r4  = Interval.less_than_or_equal_to(100)
>>> r5  = Interval.all()
>>> r6  = Interval.between(-100, 100, False)
>>> r7  = Interval(-100, 100, lower_closed=False)
>>> r8  = Interval.greater_than(-100)
>>> r9  = Interval.equal_to(-100)
>>> r10 = Interval(-100, 100, upper_closed=False)
>>> r11 = Interval.between(-100, 100)
>>> r12 = Interval.greater_than_or_equal_to(-100)
>>> r13 = Interval.greater_than(100)
>>> r14 = Interval.equal_to(100)
>>> r15 = Interval.greater_than_or_equal_to(100)
>>> r1.adjacent_to(r6)
False
>>> r6.adjacent_to(r11)
False
```

(continues on next page)

(continued from previous page)

```
>>> r7.adjacent_to(r9)
True
>>> r3.adjacent_to(r10)
False
>>> r5.adjacent_to(r14)
False
>>> r6.adjacent_to(r15)
True
>>> r1.adjacent_to(r8)
False
>>> r12.adjacent_to(r14)
False
>>> r6.adjacent_to(r13)
False
>>> r2.adjacent_to(r15)
False
>>> r1.adjacent_to(r4)
False
```

classmethod all()

Returns an interval encompassing all values

```
>>> print Interval.all()
(...)
```

classmethod between(*a*, *b*, *closed=True*)

Returns an interval between two values

Returns an interval between values a and b. If closed is True, then the endpoints are included. Otherwise, the endpoints are excluded.

```
>>> print Interval.between(2, 4)
[2..4]
>>> print Interval.between(2, 4, False)
(2..4)
```

comes_before(*other*)

Tells whether an interval lies before the object

self comes before other when sorted if its lower bound is less than other's smallest value. If the smallest value is the same, then the Interval with the smallest upper bound comes first. Otherwise, they are equal.

```
>>> Interval.equal_to(1).comes_before(Interval.equal_to(4))
True
>>> Interval.less_than_or_equal_to(1).comes_before(Interval.equal_to(4))
True
>>> Interval.less_than_or_equal_to(5).comes_before(
...   Interval.less_than(5))
False
>>> Interval.less_than(5).comes_before(
...   Interval.less_than_or_equal_to(5))
True
>>> Interval.all().comes_before(Interval.all())
False
```

duration()
@note: added 20140618 kkappler returns the duration of interval

classmethod equal_to(*a*)
Returns an point interval

Returns an interval containing only a.

```
>>> print Interval.equal_to(32)
32
```

classmethod greater_than(*a*)
Returns interval of all values greater than the given value

```
>>> print Interval.greater_than(32)
(32...)
```

classmethod greater_than_or_equal_to(*a*)
Returns interval of all values greater than or equal to the given value

```
>>> print Interval.greater_than_or_equal_to(32)
[32...)
```

join(*other*)
Combines two continous Intervals

Combines two continuous Intervals into one Interval. If the two Intervals are disjoint, then an exception is raised.

```
>>> r1  = Interval.less_than(-100)
>>> r2  = Interval.less_than_or_equal_to(-100)
>>> r3  = Interval.less_than(100)
>>> r4  = Interval.less_than_or_equal_to(100)
>>> r5  = Interval.all()
>>> r6  = Interval.between(-100, 100, False)
>>> r7  = Interval(-100, 100, lower_closed=False)
>>> r8  = Interval.greater_than(-100)
>>> r9  = Interval.equal_to(-100)
>>> r10 = Interval(-100, 100, upper_closed=False)
>>> r11 = Interval.between(-100, 100)
>>> r12 = Interval.greater_than_or_equal_to(-100)
>>> r13 = Interval.greater_than(100)
>>> r14 = Interval.equal_to(100)
>>> r15 = Interval.greater_than_or_equal_to(100)
>>> print r13.join(r15)
[100...)
>>> print r7.join(r6)
(-100..100]
>>> print r11.join(r2)
(...100]
>>> print r4.join(r15)
(...)
>>> print r8.join(r8)
(-100...)
>>> print r3.join(r7)
```

(continues on next page)

(continued from previous page)

```
(...100]
>>> print r5.join(r10)
(...)
>>> print r9.join(r1)
(...-100]
>>> print r12.join(r5)
(...)
>>> print r13.join(r1)
Traceback (most recent call last):
...
ArithmeticError: The Intervals are disjoint.
>>> print r14.join(r2)
Traceback (most recent call last):
...
ArithmeticError: The Intervals are disjoint.
```

classmethod less_than(*a*)

Returns interval of all values less than the given value

Returns an interval containing all values less than a. If closed is True, then all values less than or equal to a are returned.

```
>>> print Interval.less_than(32)
(...32)
```

classmethod less_than_or_equal_to(*a*)

Returns an interval containing the given values and everything less

```
>>> print Interval.less_than_or_equal_to(32)
(...32]
```

classmethod none()

Returns an empty interval

```
>>> print Interval.none()
<Empty>
```

overlaps(*other*)

Tells whether the given interval overlaps the object

Returns True if the one Interval overlaps another. If they are immediately adjacent, then this returns False. Use the adjacent_to function for testing for adjacent Intervals.

```
>>> r1  = Interval.less_than(-100)
>>> r2  = Interval.less_than_or_equal_to(-100)
>>> r3  = Interval.less_than(100)
>>> r4  = Interval.less_than_or_equal_to(100)
>>> r5  = Interval.all()
>>> r6  = Interval.between(-100, 100, False)
>>> r7  = Interval(-100, 100, lower_closed=False)
>>> r8  = Interval.greater_than(-100)
>>> r9  = Interval.equal_to(-100)
>>> r10 = Interval(-100, 100, upper_closed=False)
>>> r11 = Interval.between(-100, 100)
```

(continues on next page)

(continued from previous page)

```
>>> r12 = Interval.greater_than_or_equal_to(-100)
>>> r13 = Interval.greater_than(100)
>>> r14 = Interval.equal_to(100)
>>> r15 = Interval.greater_than_or_equal_to(100)
>>> r8.overlaps(r9)
False
>>> r12.overlaps(r6)
True
>>> r7.overlaps(r8)
True
>>> r8.overlaps(r4)
True
>>> r14.overlaps(r11)
True
>>> r10.overlaps(r13)
False
>>> r5.overlaps(r1)
True
>>> r5.overlaps(r2)
True
>>> r15.overlaps(r6)
False
>>> r3.overlaps(r1)
True
```

pad(*pad_width*)

@note: added 20140618 kkappler returns a padded interval; Intent was to add a little so that open-intervals, or adjacent intervals would easily make an interval set

class aurora.interval.**IntervalSet**(*items=[]*)

Bases: *aurora.interval.BaseIntervalSet*

The mutable version of BaseIntervalSet

IntervalSet is a class representing sets of continuous values, as opposed to a discrete set, which is already implemented by the set type in Python.

IntervalSets can be bounded, unbounded, and non-continuous. They were designed to accomodate any sort of mathematical set dealing with continuous values. This will usually mean numbers, but any Python type that has valid comparison operations can be used in an IntervalSet.

Because IntervalSets are mutable, it cannot be used as a dictionary key.

```
>>> {IntervalSet([3, 66]) : 52}
Traceback (most recent call last):
    ...
TypeError: unhashable instance
```

Methods

add(obj)	Adds an Interval or discrete value to the object
all()	Returns an interval set containing all values
between(a, b[, closed])	Returns an IntervalSet of all values between a and b.
bounds()	Returns an interval that encompasses the entire BaseIntervalSet
clear()	Removes all Intervals from the object
copy()	Returns a copy of the object
difference(other)	Returns the difference between the object and the given object
difference_update(other)	Removes any elements in the given value from the object
discard(obj)	Removes a value from the object
empty()	Returns an interval set containing no values.
greater_than(n)	Returns an IntervalSet containing values greater than the given value
greater_than_or_equal_to(n)	Returns an IntervalSet containing values greater than or equal to the given value
intersection(other)	Returns the intersection between the object and the given value
intersection_update(other)	Removes values not found in the parameter
issubset(other)	Tells if the given object is a subset of the object
issuperset(other)	Tells whether the given object is a superset of the object
less_than(n)	Returns an IntervalSet containing values less than the given value
less_than_or_equal_to(n[, closed])	Returns an IntervalSet containing values less than or equal to the given value
lower_bound()	Returns the lower boundary of the BaseIntervalSet
lower_closed()	Returns a boolean telling whether the lower bound is closed or not
not_equal_to(n)	Returns an IntervalSet of all values not equal to n
pop()	Returns and discards an Interval or value from the IntervalSet
remove(obj)	Removes a value from the object
symmetric_difference(other)	Returns the exclusive or of the given value with the object
symmetric_difference_update(other)	Updates the object as though doing an xor with the parameter
union(other)	Returns the union of the given value with the object
update(other)	Adds elements from the given value to the object
upper_bound()	Returns the upper boundary of the BaseIntervalSet
upper_closed()	Returns a boolean telling whether the upper bound is closed or not

add(*obj*)

Adds an Interval or discrete value to the object

```
>>> r = IntervalSet()
>>> r.add(4)
```

(continues on next page)

(continued from previous page)

```
>>> print r
4
>>> r.add(Interval(23, 39, lower_closed=False))
>>> print r
4,(23..39]
>>> r.add(Interval.less_than(25))
>>> print r
(...39]
```

clear()

Removes all Intervals from the object

```
>>> s = IntervalSet([2, 7, Interval.greater_than(8), 2, 6, 34])
>>> print s
2,6,7,(8...)
>>> s.clear()
>>> print s
<Empty>
```

difference_update(*other*)

Removes any elements in the given value from the object

This function removes the elements in other from self. other can be any iterable object.

```
>>> r = IntervalSet.all()
>>> r.difference_update([4])
>>> print r
(...4),(4...)
>>> r.difference_update(
...   IntervalSet([Interval(23, 39, lower_closed=False)]))
>>> print r
(...4),(4..23],(39...)
>>> r.difference_update(IntervalSet.less_than(25))
>>> print r
(39...)
>>> r2 = IntervalSet.all()
>>> r.difference_update(r2)
>>> print r
<Empty>
```

discard(*obj*)

Removes a value from the object

This function removes an Interval or discrete value from an IntervalSet.

```
>>> r = IntervalSet.all()
>>> r.discard(4)
>>> print r
(...4),(4...)
>>> r.discard(Interval(23, 39, lower_closed=False))
>>> print r
(...4),(4..23],(39...)
>>> r.discard(Interval.less_than(25))
```

(continues on next page)

(continued from previous page)

```
>>> print r
(39...)
```

intersection_update(*other*)

Removes values not found in the parameter

Removes elements not found in other. other can be any iterable object

```
>>> r = IntervalSet.all()
>>> r.intersection_update([4])
>>> print r
4
>>> r = IntervalSet.all()
>>> r.intersection_update(
...   IntervalSet([Interval(23, 39, lower_closed=False)]))
>>> print r
(23..39]
>>> r.intersection_update(IntervalSet.less_than(25))
>>> print r
(23..25)
>>> r2 = IntervalSet.all()
>>> r.intersection_update(r2)
>>> print r
(23..25)
```

pop()

Returns and discards an Interval or value from the IntervalSet

```
>>> s = IntervalSet([7, Interval.less_than(2), 2, 0])
>>> l = []
>>> l.append(str(s.pop()))
>>> l.append(str(s.pop()))
>>> "(...2)" in l
False
>>> "(...2]" in l
True
>>> "7" in l
True
>>> print s
<Empty>
>>> i = s.pop()
Traceback (most recent call last):
    ...
KeyError: 'pop from an empty IntervalSet'
```

remove(*obj*)

Removes a value from the object

This function removes an Interval, discrete value, or set from an IntervalSet. If the object is not in the set, a KeyError is raised.

```
>>> r = IntervalSet.all()
>>> r.remove(4)
>>> print r
```

(continues on next page)

(continued from previous page)

```
(...4),(4...)
>>> r.remove(Interval(23, 39, lower_closed=False))
>>> print r
(...4),(4..23],(39...)
>>> r.remove(Interval.less_than(25))
Traceback (most recent call last):
    ...
KeyError: '(...25)'
```

symmetric_difference_update(*other*)

Updates the object as though doing an xor with the parameter

Removes elements found in other and adds elements in other that are not in self. other can be any iterable object.

```
>>> r = IntervalSet.empty()
>>> r.symmetric_difference_update([4])
>>> print r
4
>>> r.symmetric_difference_update(
...   IntervalSet([Interval(23, 39, lower_closed=False)]))
>>> print r
4,(23..39]
>>> r.symmetric_difference_update(IntervalSet.less_than(25))
>>> print r
(...4),(4..23],[25..39]
>>> r2 = IntervalSet.all()
>>> r.symmetric_difference_update(r2)
>>> print r
4,(23..25),(39...)
```

update(*other*)

Adds elements from the given value to the object

Adds elements from other to self. other can be any iterable object.

```
>>> r = IntervalSet()
>>> r.update([4])
>>> print r
4
>>> r.update(IntervalSet([Interval(23, 39, lower_closed=False)]))
>>> print r
4,(23..39]
>>> r.update(IntervalSet.less_than(25))
>>> print r
(...39]
>>> r2 = IntervalSet.all()
>>> r.update(r2)
>>> print r
(...)
```

class aurora.interval.**Largest**

Bases: `object`

Class representing the universal largest value

This type doesn't do much; it implements a pseudo-value that's larger than everything but itself.

```
>>> infinity = Largest()
>>> greatest = Largest()
>>> 6234 < infinity
True
>>> 6234 == infinity
False
>>> 6234 > infinity
False
>>> infinity > infinity
False
>>> infinity == greatest
True
```

class aurora.interval.**Smallest**

Bases: object

Represents the smallest value

This type doesn't do much; it implements a pseudo-value that's smaller than everything but itself.

```
>>> negInf = Smallest()
>>> smallest = Smallest()
>>> -264 < negInf
False
>>> -264 == negInf
False
>>> -264 > negInf
True
>>> negInf < negInf
False
>>> negInf == smallest
True
```

class aurora.interval.**TimePeriod**(***kwargs*)

Bases: *aurora.interval.Interval*

Methods

adjacent_to(other)	Tells whether an Interval is adjacent to the object without overlap
all()	Returns an interval encompassing all values
between(a, b[, closed])	Returns an interval between two values
comes_before(other)	Tells whether an interval lies before the object
equal_to(a)	Returns an point interval
greater_than(a)	Returns interval of all values greater than the given value
greater_than_or_equal_to(a)	Returns interval of all values greater than or equal to the given value
join(other)	Combines two continous Intervals
less_than(a)	Returns interval of all values less than the given value

continues on next page

Table 7 – continued from previous page

less_than_or_equal_to(a)	Returns an interval containing the given values and everything less
none()	Returns an empty interval
overlaps(other)	Tells whether the given interval overlaps the object
pad(pad_width)	@note: added 20140618 kkappler returns a padded interval; Intent was to add a little so that open-intervals, or adjacent intervals would easily make an interval set

duration	

duration()

aurora.interval.**generate_interval_list**(*startTime*, *endTime*, *delta*)

generic function used for specifying timeintervals to load probably place in gmi.core.interval

Ideally would support endtime, number of segments or other

aurora.interval.**merge_interval_list**(*interval_list*)

Can think of a stack of unassembled parts. Put the first part in your left hand, then draw the next part in your right. Check if the pieces mate, if so, attach and keep in your left hand, draw the next piece, and so on. Once a piece does not fit, place the assembled piece in the left hand down and transfer the new 'no-fit' piece from your right to your left hand and contine ...

KEY: This assumes the interval list is sorted. If the intervals are not sorted then you need to use set union. This is way faster though.

9.5 General Helper Functions

PYTHON MODULE INDEX

a

INDEX

E

F

G

I

J

L

M

N

O

P

R

S

T

U