Space Physics WebServices Client Documentation

Release 0.10.1

Alexis Jeandet

Feb 03, 2022

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Stable release

To install Space Physics WebServices Client, run this command in your terminal:

```
$ pip install speasy
# or
$ pip install --user speasy
```

This is the preferred method to install Space Physics WebServices Client, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.2 From sources

The sources for Space Physics WebServices Client can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/SciQLop/speasy
```

Or download the tarball:

```
$ curl  -OL https://github.com/SciQLop/speasy/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

TWO

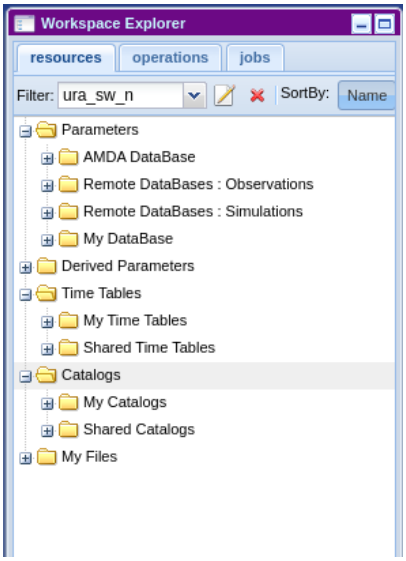
MODULES

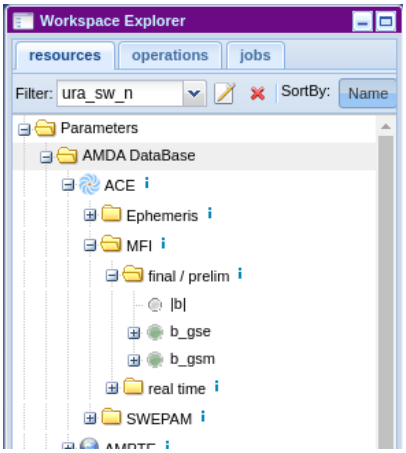
2.1 AMDA

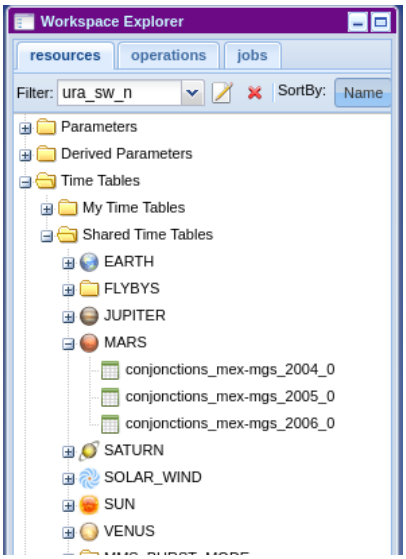
AMDA is one of the main data providers handled by speasy. Most products are either available using directly the AMDA module or using `speasy.get_data()`. The following documentation will focus on AMDA module specific usage.

2.1.1 Basics: Getting data from AMDA

AMDA distributes several products such as Parameters, user Parameters, Datasets, Timetables, user Timetables, Catalogs and user Catalogs. Speasy makes them accessible thanks to this module with `get_data()` or their dedicated methods such as `get_parameter()`, `get_user_parameter()`,... Note that you can browse the list of all available products from AMDA Workspace:







This module provides two kinds of operations, **list** or **get** and so user methods are prefixed with one of them.

- **get** methods retrieve the given product from AMDA server, they takes at least the product identifier and time range for time series
- **list** methods list available products of a given type on AMDA, they return a list of indexes that can be passed to a **get** method

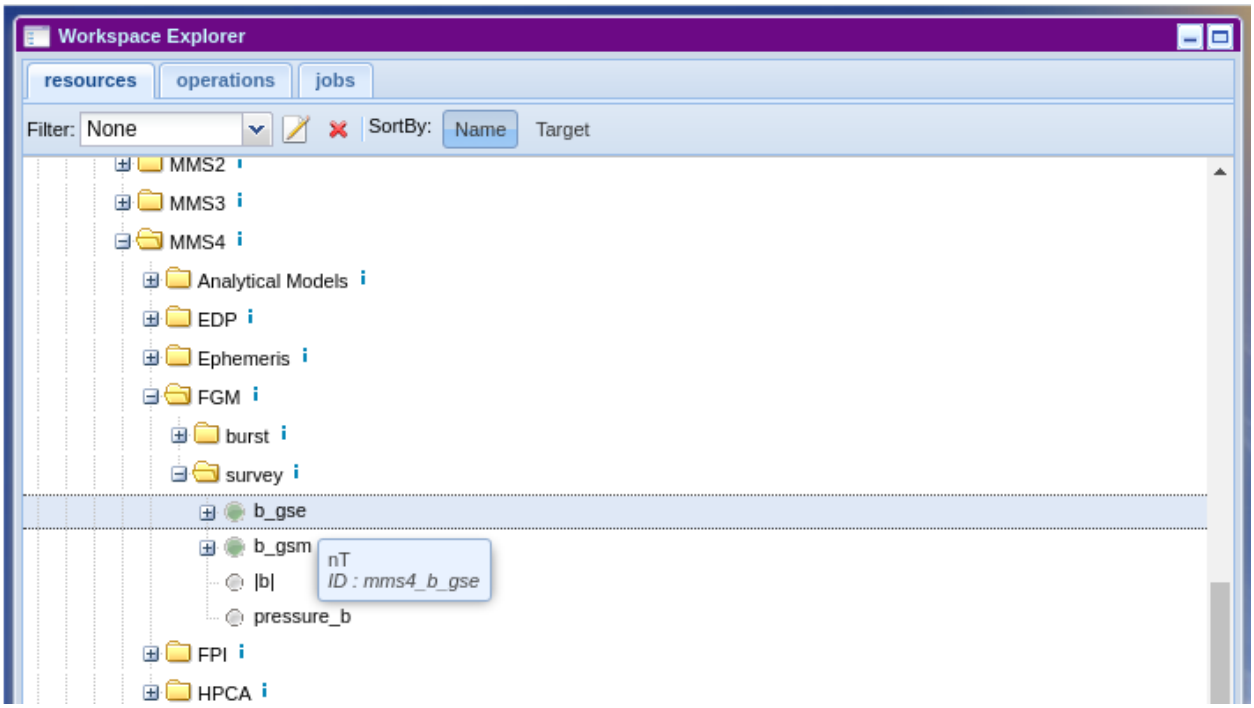
Parameters

Let's start with a simple example, we want to download the first parameter available on AMDA:

```
>>> from speasy import amda
>>> first_param_index=amda.list_parameters()[0]
>>> print(first_param_index)
<ParameterIndex: |b|>
>>> first_param=amda.get_parameter(first_param_index, "2018-01-01", "2018-01-02")
>>> first_param.columns
['imf_mag']
>>> len(first_param.time)
5400
```

Usually you already know which product you want to download, two scenarios are available:

1. You are an AMDA web interface user, so you want some specific product from AMDA Workspace. You need first to get your product id, you will find the id from the tooltip while hovering any product (Dataset, Parameter, Timetable or Catalog):



Then simply:

```
>>> from speasy import amda
>>> mms4_fgm_btot=amda.get_parameter('mms4_b_tot', "2018-01-01", "2018-01-02")
>>> mms4_fgm_btot.columns
['mms4_b_tot']
>>> len(mms4_fgm_btot.time)
986745
```

2. Second scenario, your are not much familiar with AMDA, then you can simply browse speasy dynamic inventory. In the following example, we alias AMDA data tree as amdatree, note that Python completion works and you will be able to discover AMDA products directly from your Python terminal or notebook:

```
>>> from speasy import amda
>>> from speasy.inventory.data_tree import amda as amdatree
>>> mms4_fgm_btot=amda.get_parameter(amdatree.Parameters.MMS.MMS4.FGM.mms4_fgm_srvy.mms4_
↪b_tot, "2018-01-01", "2018-01-02")
>>> mms4_fgm_btot.columns
['mms4_b_tot']
>>> len(mms4_fgm_btot.time)
986745
```

See `get_parameter()` or `get_data()` for more details.

Catalogs and TimeTables

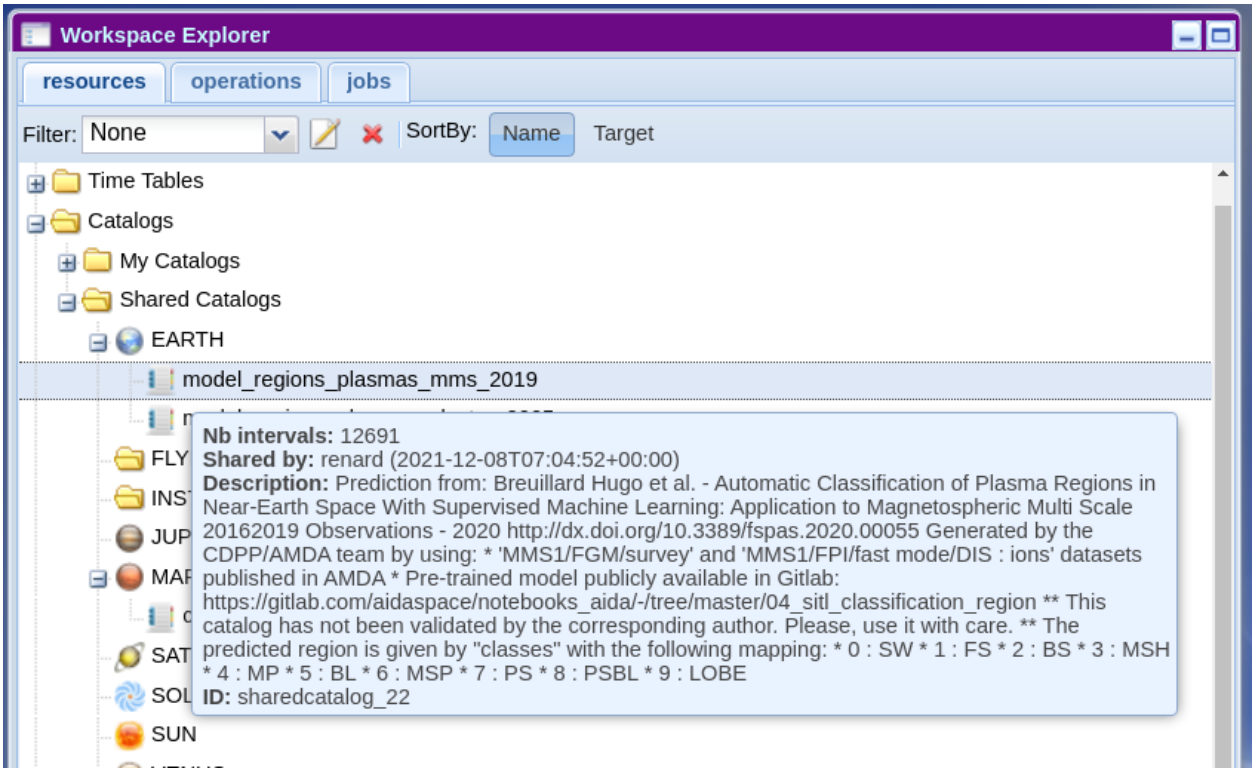
Downloading Catalogs and TimeTables from AMDA is similar to Parameters. For example let's assume you want to download the first available catalog:

```
>>> from speasy import amda
>>> first_catalog_index=amda.list_catalogs()[0]
>>> print(first_catalog_index)
<CatalogIndex: model_regions_plasmas_mms_2019>
>>> first_catalog=amda.get_catalog(first_catalog_index)
>>> first_catalog
<Catalog: model_regions_plasmas_mms_2019>
>>> len(first_catalog)
12691
>>> print(first_catalog[1])
<Event: 2019-01-01T00:24:04+00:00 -> 2019-01-01T00:24:04+00:00 | {'classes': '1'}>
```

Exactly the same with a TimeTable:

```
>>> from speasy import amda
>>> first_timetable_index=amda.list_timetables()[0]
>>> print(first_timetable_index)
<TimetableIndex: FTE_c1>
>>> first_timetable=amda.get_timetable(first_timetable_index)
>>> first_timetable
<TimeTable: FTE_c1>
>>> len(first_timetable)
782
>>> print(first_timetable[1])
<DateTimeRange: 2001-02-02T17:29:29+00:00 -> 2001-02-02T17:29:30+00:00>
```

As with Parameters you can also use the ID found on AMDA web user interface:



Then simply:

```
>>> from speasy import amda
>>> catalog_mms_2019=amda.get_catalog("sharedcatalog_22")
>>> catalog_mms_2019
<Catalog: model_regions_plasmas_mms_2019>
>>> len(catalog_mms_2019)
12691
>>> print(catalog_mms_2019[1])
<Event: 2019-01-01T00:24:04+00:00 -> 2019-01-01T00:24:04+00:00 | {'classes': '1'}>
```

And also alternatively you can use the dynamic inventory:

```
>>> from speasy import amda
>>> from speasy.inventory.data_tree import amda as amdatree
>>> catalog_mms_2019=amda.get_catalog(amdatree.Catalogs.SharedCatalogs.EARTH.model_
↪regions_plasmas_mms_2019)
>>> catalog_mms_2019
<Catalog: model_regions_plasmas_mms_2019>
>>> len(catalog_mms_2019)
12691
>>> print(catalog_mms_2019[1])
<Event: 2019-01-01T00:24:04+00:00 -> 2019-01-01T00:24:04+00:00 | {'classes': '1'}>
```

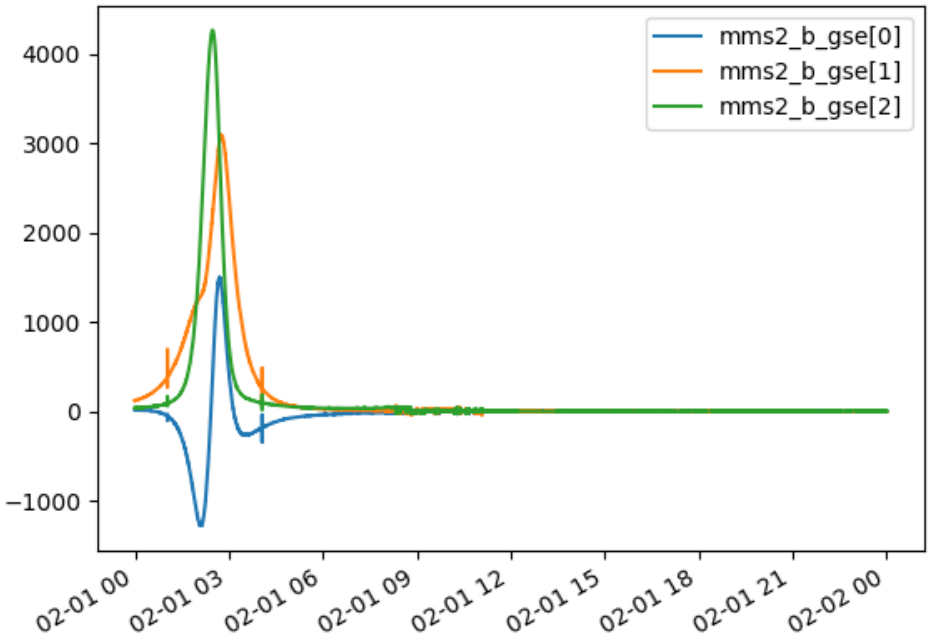
2.1.2 Some examples using AMDA products

My first plot from AMDA

In this example we will use AMDA module to retrieve and plot MMS2 FGM data, feel free to change the code and experiment!

```
>>> import matplotlib.pyplot as plt
>>> import speasy as spz
>>> from speasy.inventory.data_tree import amda as amda_tree
>>> mms2_b_gse = spz.amda.get_parameter(amda_tree.Parameters.MMS.MMS2.FGM.mms2_fgm_srvy.
↪mms2_b_gse, "2020-02-01", "2020-02-02")
>>> # Check that mms2_b_gse isn't empty
>>> len(mms2_b_gse)
1382895
>>> # Then you can use the SpeasyVariable plot method for quick plots
>>> mms2_b_gse.plot()
>>> plt.show()
```

Then you should get something like this:



Note:

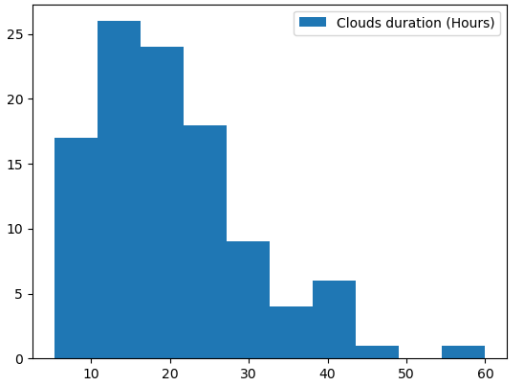
- Depending on your matplotlib backend and if you are using Jupyter Notebooks or a simple python terminal you may need to adapt this example.
- Speasy is not a plotting package, to produce publication ready figures, use something like matplotlib or seaborn directly.

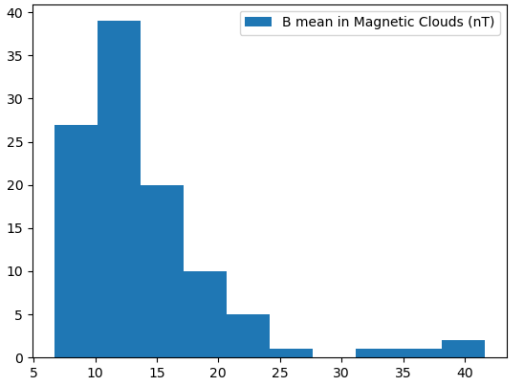
Using timetables to download data

In this example we will use AMDA to first retrieve a public timetable containing time intervals where Magnetic Clouds were detected with Wind spacecraft. Then download the magnetic field magnitude measured with MFI instrument for each interval where a Magnetic cloud was found. Once we have magnetic field measurements inside each cloud, we will as an example plot the average distribution.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> import speasy as spz
>>> from speasy.inventory.data_tree import amda as amda_tree
>>> Magnetic_Clouds = spz.amda.get_timetable(amda_tree.TimeTables.SharedTimeTables.SOLAR_
↪WIND.Magnetic_Clouds)
>>> print(Magnetic_Clouds.meta['description'])
Magnetic Clouds from WIND/MFI 1995-2007 -- Estimated start and end times from a magnetic␣
↪field model [Lepping et al., 1990] which assumes that the field within the magnetic␣
↪cloud is force free, i.e., so that the electrical current and the magnetic field are␣
↪parallel and proportional in strength everywhere within its volume -- see http://
↪lepmfi.gsfc.nasa.gov/mfi/mag_cloud_pub1.html ;
        Historic: From old AMDA;
        Creation Date :  2013-11-22T13:52:50;
>>> # Check that the timetable has at least some events (as expected)
>>> len(Magnetic_Clouds)
106
>>> # Then we can plot their duration distribution
>>> def duration(event):
...     return (event.stop_time.timestamp() - event.start_time.timestamp())/3600
...
>>> clouds_duration = [duration(cloud) for cloud in Magnetic_Clouds]
>>> plt.hist(clouds_duration, label="Clouds duration (Hours)")
>>> plt.legend()
>>> plt.show()
>>> # Now let's get MFI data for each cloud
>>> b_mfi_coulds = [ spz.amda.get_parameter(amda_tree.Parameters.Wind.MFI.wnd_mfi_kp.wnd_
↪bmag, cloud.start_time, cloud.stop_time) for cloud in Magnetic_Clouds ]
>>> # compute mean of B for each cloud and ignore NaNs
>>> b_mean_mfi_clouds = [ np.nanmean(cloud.data) for cloud in b_mfi_coulds ]
>>> plt.hist(b_mean_mfi_clouds, label="B mean in Magnetic Clouds (nT)")
>>> plt.legend()
>>> plt.show()
```

Then you should get something like these plots:





Note:

- Depending on your matplotlib backend and if you are using Jupyter Notebooks or a simple python terminal you may need to adapt this example.
- Speasy is not a plotting package, to produce publication ready figures, use something like matplotlib or seaborn directly.

2.1.3 Advanced: AMDA module configuration options

AMDA user login

Most AMDA features are available without login except user created product from web user interface. You can configure speasy to store your AMDA login, from your favourite python terminal:

```
>>> from speasy import config
>>> config.amda_username.set('my_username')
>>> config.amda_password.set('my_password')
>>> # check that your login/password are correctly set
>>> config.amda_username.get(), config.amda_password.get()
('my_username', 'my_password')
```

Then if you correctly typed your login you should be able to list and get user products:

```
>>> from speasy import amda
>>> # list user products
>>> amda.list_user_parameters()
[<ParameterIndex: test_param>]
>>> amda.list_user_catalogs()
[<CatalogIndex: MyCatalog>]
>>> amda.list_user_timetables()
[<TimetableIndex: test_alexis>, <TimetableIndex: test_alexis2>, <TimetableIndex: tt3>]
>>> # get my first user catalog
>>> amda.get_user_catalog(amda.list_user_catalogs()[0])
<Catalog: MyCatalog>
```

AMDA cache retention

While parameter download cache is not configurable and relies on product version to decide if local data is up to date compared to remote data. Requests like catalogs or time-tables download have a different dedicated cache based on duration, by default they will be cached for 15 minutes. As a consequence if a time-table has changed on AMDA servers it might take up to the configured duration to see it. This cache has been designed with interactive usage of speasy in mind where we want to minimize penalty of running multiple times the same command/line.

To change this cache duration value:

```
>>> from speasy import config
>>> # set cache duration to 900 seconds
>>> config.amda_user_cache_retention.set('900')
>>> config.amda_user_cache_retention.get()
'900'
```

2.2 SSCWEB

SSCWeb provides trajectories for our solar system space objects such as planets, moons and spacecrafts in different coordinate systems. It's integration into speasy makes easy to get any available object trajectory on any time range.

2.2.1 Basics: Getting data from SSCWeb module

First you need to ensure that the trajectory you want to get is available with this module. The easiest solution is use speasy dynamic inventory so you will always get an up to date inventory:

```
>>> import speasy as spz
>>> # Let's only print the first 10 trajectories
>>> print(list(spz.inventory.flat_inventories.ssc.parameters.keys())[:10])
['ace', 'active', 'aec', 'aed', 'aee', 'aerocube6a', 'aerocube6b', 'aim', 'akebono',
↪'alouette1']
```

Note that you can also use your python terminal completion and browse *spz.inventory.data_tree.ssc.Trajectories* to find your trajectory. Once you have found your trajectory, you may also want to chose in which coordinates system your data will be downloaded. The following coordinates systems are available: **geo**, **gm**, **gse**, **gsm**, **sm**, **geitod**, **geij2000**. By default **gse** is used. Now you can get your trajectory:

```
>>> import speasy as spz
>>> # Let's assume you wanted to get MMS1 trajectory
>>> mms1_traj = spz.ssc.get_trajectory(spz.inventory.data_tree.ssc.Trajectories.mms1,
↪"2018-01-01", "2018-02-01", 'gsm')
>>> mms1_traj.columns
['X', 'Y', 'Z']
>>> mms1_traj.data
<Quantity [[57765.77891127, 39928.64689416, 36127.69757491],
           [57636.78726753, 39912.67690181, 36075.18117495],
           [57507.67093183, 39896.65117739, 36022.43945697],
           ...,
           [74135.04374424,   741.72325874, 27240.73393024],
           [74007.246673  ,   795.05699457, 27220.37053627],
           [73879.18392451,   848.35181084, 27199.87604795]] km>
```

2.3 CDAWEB

Speasy provides access to the following Web Services:

- *AMDA*
- *SSCWeb*
- *CDAWeb*

While you can download any data with `speasy.get_data()`, each web service have specificities and might expose extra features through their dedicated modules.

CHAPTER

THREE

SPEASY EXAMPLES GALLERY

Browse example folder on MyBinder:

The following section was generated from `docs/examples/AMDA.ipynb`

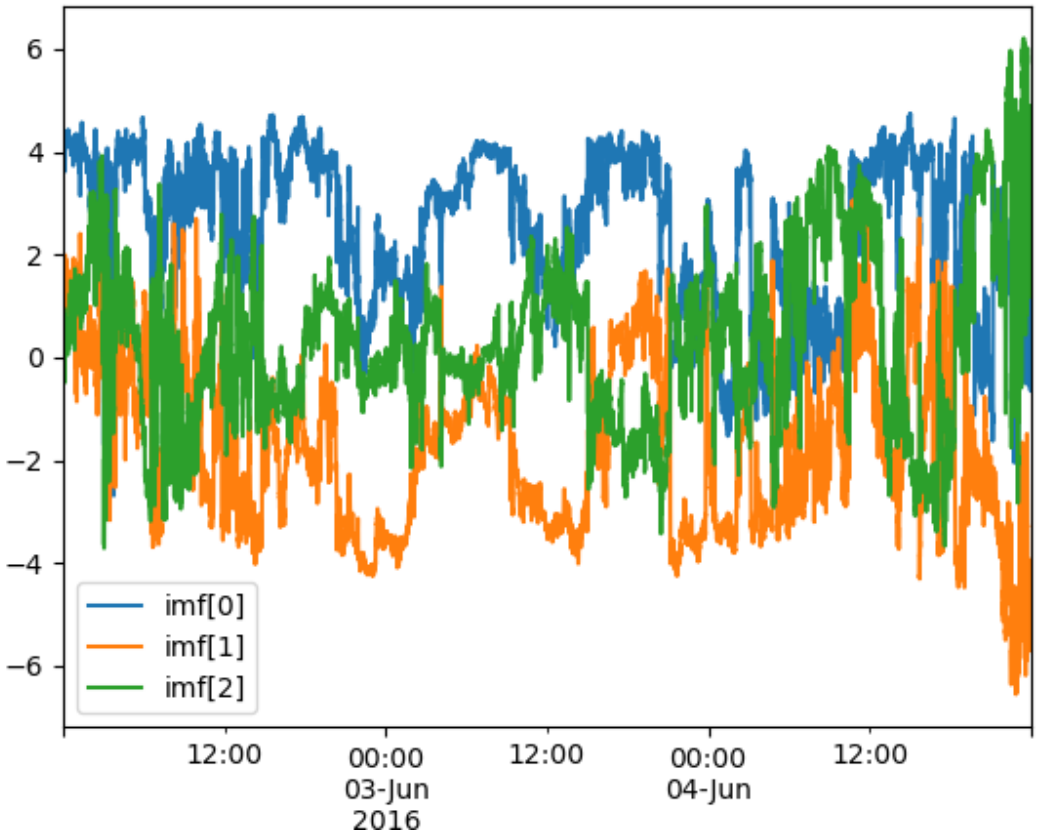
3.1 AMDA first steps

[1]:
```
import speasy as spz
%matplotlib widget
from speasy.inventory.data_tree import amda as amda_tree
# Use this instead if you are not using jupyterlab yet
#%matplotlib notebook
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
```

3.1.1 A simple example with ACE IMF data

[2]:
```
ace_mag = spz.get_data(amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, datetime(2016,6,2),␣
↪datetime(2016,6,5))
ace_mag.plot()
```

[2]:
```
<AxesSubplot:>
```



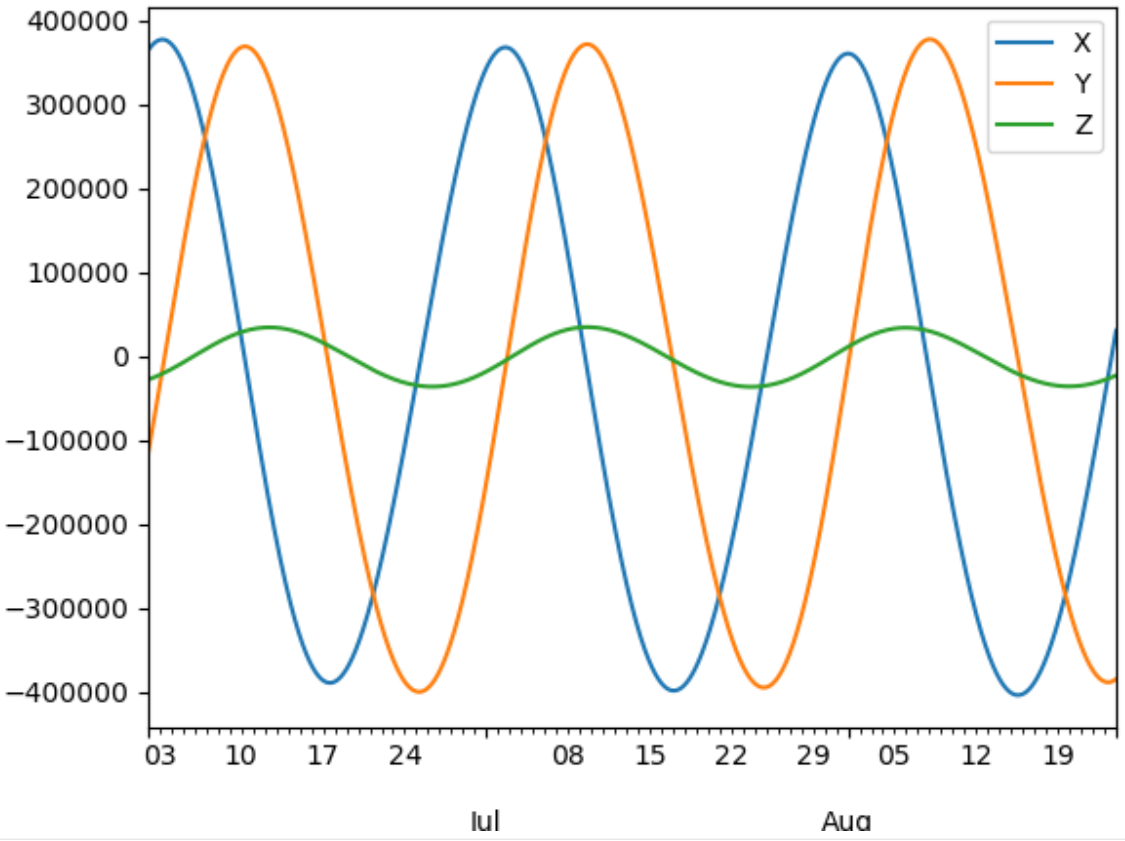
The following section was generated from docs/examples/SSCWeb.ipynb

3.2 SSCWeb first steps

[1]:
```
import speasy as spz
from speasy.inventory.data_tree import ssc as ssc_tree
%matplotlib widget
# Use this instead if you are not using jupyterlab yet
#%matplotlib notebook
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
from astropy import units
import numpy as np
```

[2]:
```
moon_orbit = spz.get_data(ssc_tree.Trajectories.moon,
                          datetime(2019,6,2), datetime(2019,8,24), coordinate_system='gse
↪')
moon_orbit.plot()
```

[2]:
```
<AxesSubplot:>
```

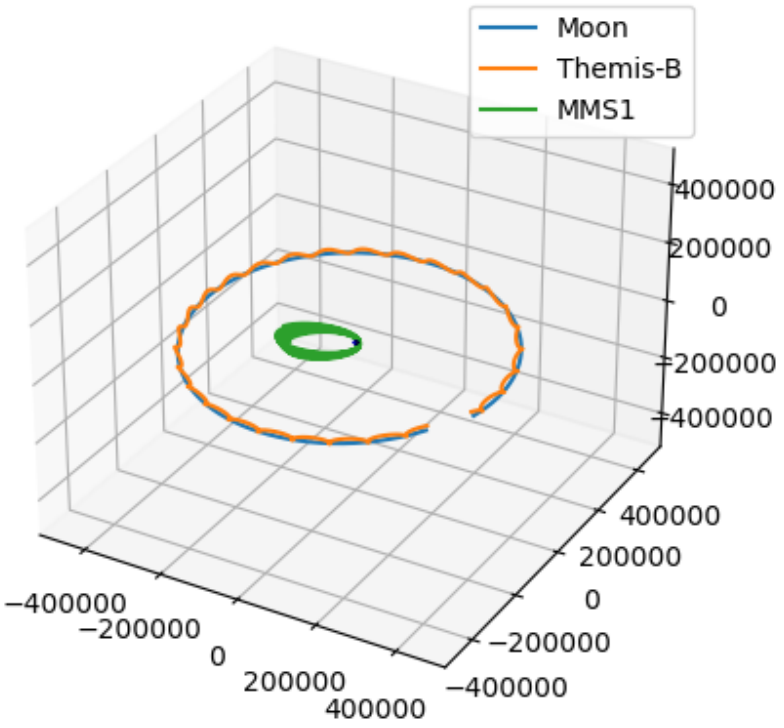


[3]:
```
def plot_traj(var, ax, label):
    ax.plot(var.data[:,0],var.data[:,1],var.data[:,2], label=label)

def plot_earth(ax):
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(0, np.pi, 100)
    x = 6371 * np.outer(np.cos(u), np.sin(v))
    y = 6371 * np.outer(np.sin(u), np.sin(v))
    z = 6371 * np.outer(np.ones(np.size(u)), np.cos(v))
    ax.plot_surface(x, y, z, color='b')

themisb_orbit = spz.get_data(ssc_tree.Trajectories.themisb,
                         datetime(2019,6,2), datetime(2019,6,30), coordinate_system='gse
↪')

mms1_orbit = spz.get_data(ssc_tree.Trajectories.mms1,
                         datetime(2019,6,2), datetime(2019,6,30), coordinate_system='gse
↪')

moon_orbit = spz.get_data(ssc_tree.Trajectories.moon,
                         datetime(2019,6,2), datetime(2019,6,30), coordinate_system='gse
↪')
```

(continues on next page)

(continued from previous page)

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
plot_traj(moon_orbit, ax, 'Moon')
plot_traj(themisb_orbit, ax, 'Themis-B')
plot_traj(mms1_orbit, ax, 'MMS1')
plot_earth(ax)
ax.set_xlim(-50e4,50e4)
ax.set_ylim(-50e4,50e4)
ax.set_zlim(-50e4,50e4)
ax.legend()
plt.show()
```



The following section was generated from `docs/examples/Caches.ipynb`

3.3 Speasy caches levels analysis

[1]:
```
import speasy as spz
from speasy.inventory.data_tree import amda as amda_tree
%matplotlib widget
```

(continues on next page)

(continued from previous page)

```
# Use this instead if you are not using jupyterlab yet
#%matplotlib notebook
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import time
import numpy as np
```

First ensure that speasy is setup to use SciQLop cache

[2]:
```
spz.config.proxy_url.set('http://sciqlop.lpp.polytechnique.fr/cache')
spz.config.proxy_enabled.set('True')
```

[3]:
```
start_time = datetime(2016,6,2)
stop_time = datetime(2016,6,8)
reference_data = spz.get_data(amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, start_time,␣
↪stop_time)
print(f"Data shape: {reference_data.data.shape}")
print(f"Data size in Bytes: {reference_data.data.nbytes}")
```

```
Data shape: (32400, 3)
Data size in Bytes: 777600
```

[4]:
```
def times(f,*args, n=10,**kwargs):
    def time_once():
        start = time.perf_counter_ns()
        f(*args, **kwargs)
        stop = time.perf_counter_ns()
        return (stop - start)/1e6
    return [time_once() for _ in range(n)]
```

3.3.1 Cache level comparison

Then request data several times with all 3 configurations:

- **without any cache**, each time speasy will download data from AMDA
- **with remote cache only**, each time speasy download data from our remote cahe hosted here
- **with local cahe**, each time after the first request speasy will load data from your disk

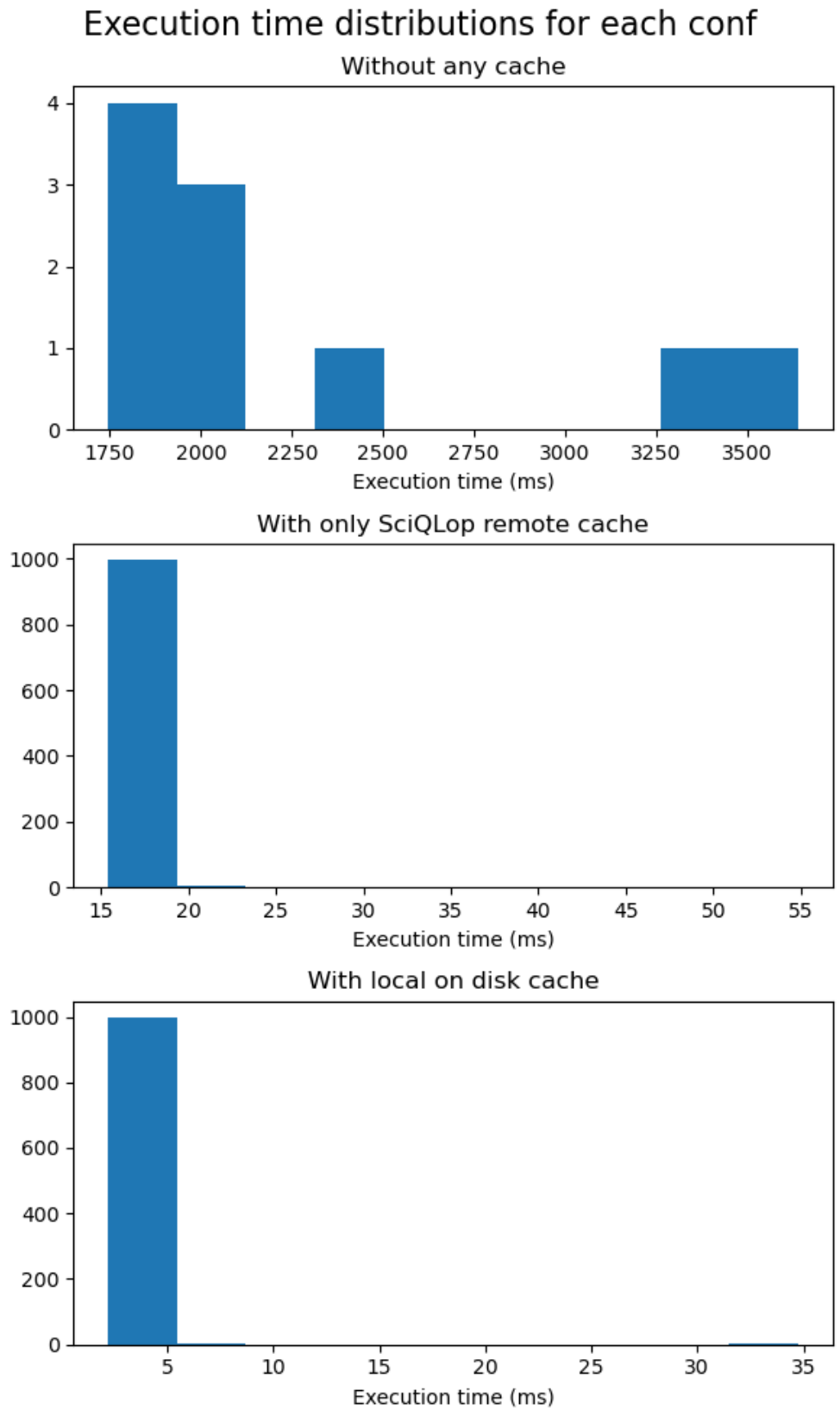
[5]:
```
durations_without_any_cache = times(spz.get_data, amda_tree.Parameters.ACE.MFI.ace_imf_
↪all.imf, start_time, stop_time, disable_cache=True, disable_proxy=True, n=10)
durations_with_remote_cache = times(spz.get_data, amda_tree.Parameters.ACE.MFI.ace_imf_
↪all.imf, start_time, stop_time, disable_cache=True, n=1000)
durations_with_local_cache = times(spz.get_data, amda_tree.Parameters.ACE.MFI.ace_imf_
↪all.imf, start_time, stop_time, n=1000)
```

[6]:
```
fig, axs = plt.subplots(3, 1, figsize=(6, 10))
for i,data,title in ((0,durations_without_any_cache,'Without any cache'),
                         (1,durations_with_remote_cache,'With only SciQLop remote cache
↪'),
                         (2,durations_with_local_cache,'With local on disk cache') ):
```

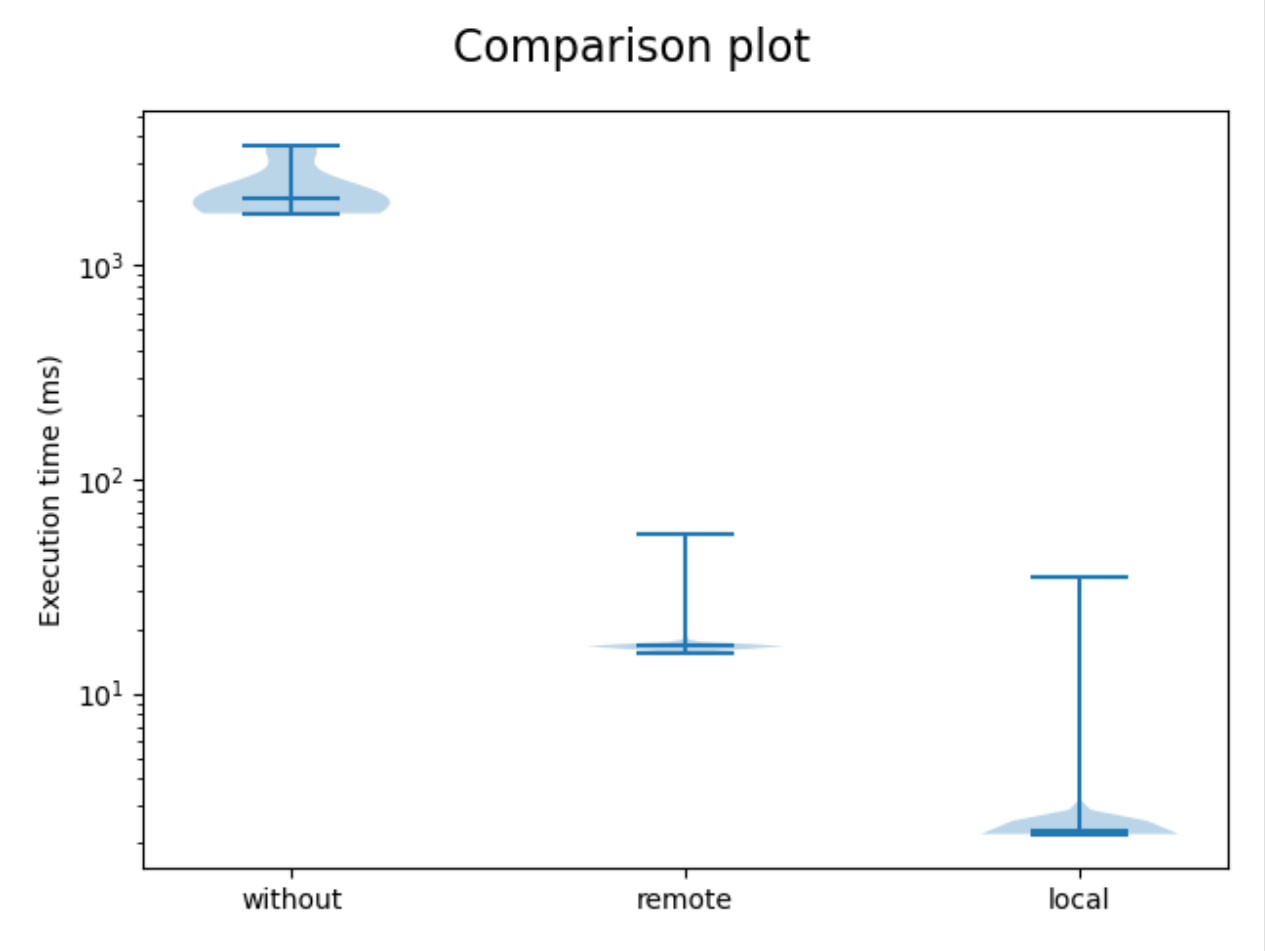
(continues on next page)

(continued from previous page)

```
    axs[i].hist(data)
    axs[i].set_xlabel('Execution time (ms)')
    axs[i].set_title(title)

fig.suptitle('Execution time distributions for each conf', fontsize=16)
plt.tight_layout()
plt.show()
```


Execution time distributions for each conf
Without any cache
4
3
2
1
0
1750
2000
2250
2500
2750
3000
3250
3500
Execution time (ms)
With only SciQLop remote cache
1000
800
600
400
200
0
15
20
25
30
35
40
45
50
55
Execution time (ms)
With local on disk cache
1000
800
600
400
200
0
5
10
15
20
25
30
35
Execution time (ms)

[7]:
```
fig, ax = plt.subplots()
ax.violinplot([durations_without_any_cache, durations_with_remote_cache,durations_with_
↪local_cache,], showmeans=False, showmedians=True)
ax.set_xticks([1,2,3],labels=['without', 'remote', 'local'])
ax.set_ylabel('Execution time (ms)')
plt.semilogy()
fig.suptitle('Comparison plot', fontsize=16)
plt.tight_layout()
plt.show()
```



3.3.2 Scaling

On disk cache scaling

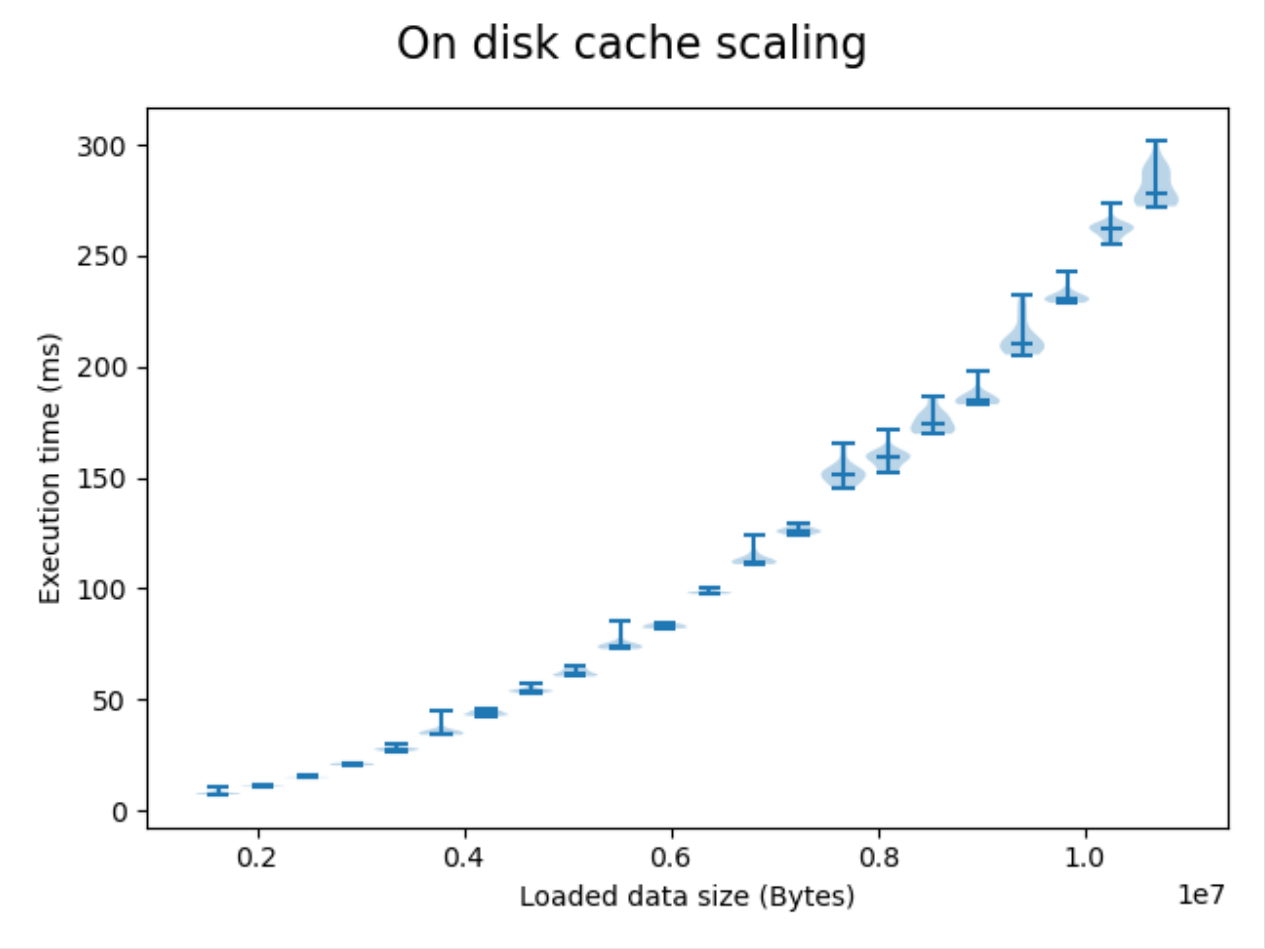
[8]:
```
start_time = datetime(2016,6,2)
def scaling_point(delta):
    stop_time=start_time+timedelta(hours=delta)
    data = spz.get_data(amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, start_time, stop_
↪time)
    capacity = data.data.nbytes
    t=times(spz.get_data, amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, start_time, stop_
↪time, n=20)
```

(continues on next page)

(continued from previous page)

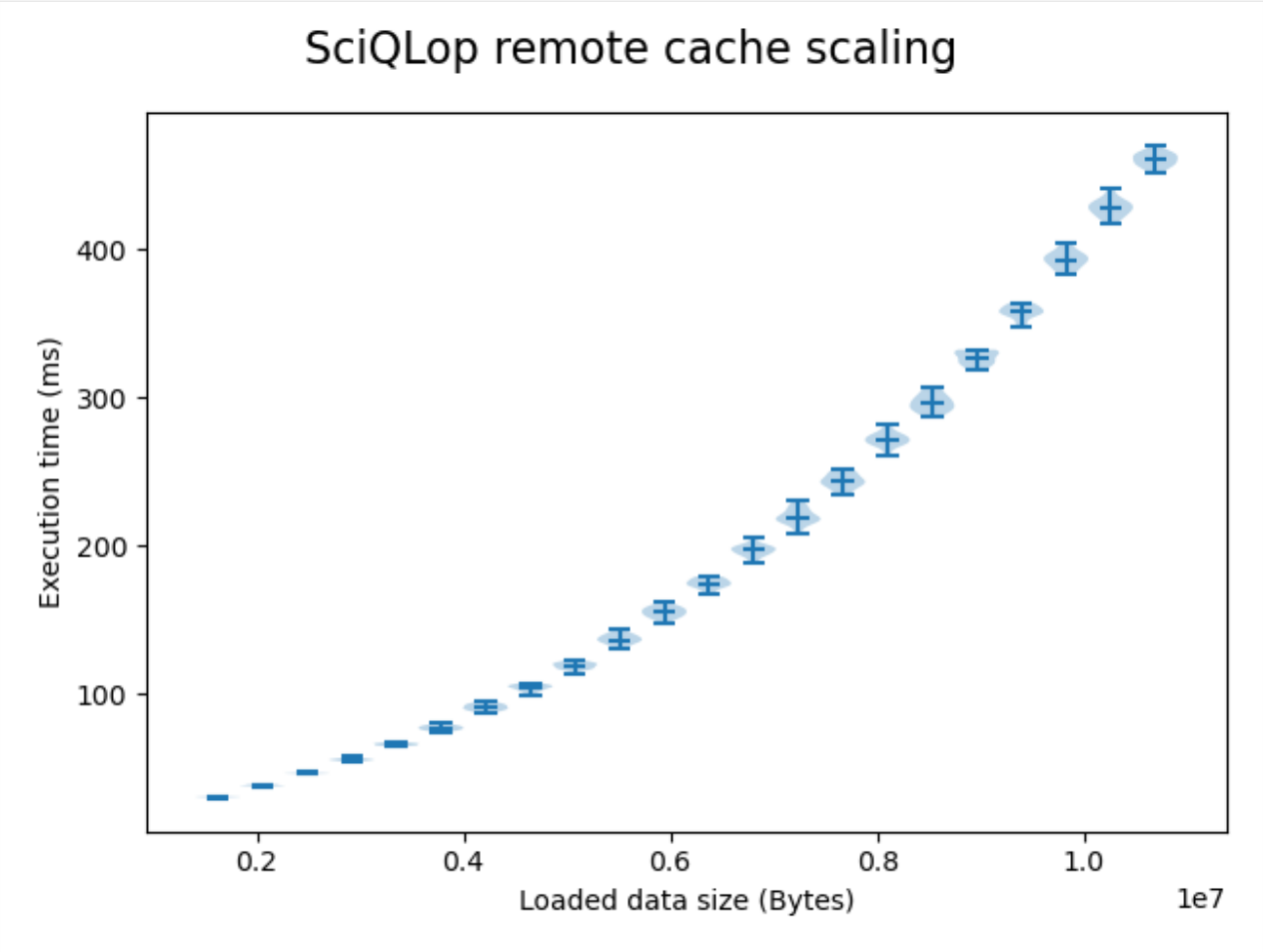
```
    return capacity,t

deltas=range(300,2000,80)
values = [scaling_point(delta) for delta in deltas]
```

[9]:
```
fig, ax = plt.subplots()
stats = [t for c,t in values]
capacities = np.array([c for c,t in values])
ax.violinplot(stats,positions=capacities,widths=np.gradient(capacities), showmeans=False,
↪ showmedians=True)
ax.set_ylabel('Execution time (ms)')
ax.set_xlabel('Loaded data size (Bytes)')
fig.suptitle('On disk cache scaling', fontsize=16)
plt.tight_layout()
plt.show()
```



SciQLop remote cache scaling

[10]:
```
start_time = datetime(2016,6,2)
def scaling_point(delta):
    stop_time=start_time+timedelta(hours=delta)
    data = spz.get_data(amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, start_time, stop_
↪time, disable_cache=True)
    capacity = data.data.nbytes
    t=times(spz.get_data, amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, start_time, stop_
↪time, disable_cache=True, n=20)
    return capacity,t

deltas=range(300,2000,80)
values = [scaling_point(delta) for delta in deltas]


fig, ax = plt.subplots()
stats = [t for c,t in values]
capacities = np.array([c for c,t in values])
ax.violinplot(stats,positions=capacities,widths=np.gradient(capacities), showmeans=False,
↪ showmedians=True)
ax.set_ylabel('Execution time (ms)')
ax.set_xlabel('Loaded data size (Bytes)')
fig.suptitle('SciQLop remote cache scaling', fontsize=16)
plt.tight_layout()
plt.show()
```


SciQLop remote cache scaling
Execution time (ms)
100
200
300
400
0.2
0.4
0.6
0.8
1.0
1e7
Loaded data size (Bytes)

CHAPTER

FOUR

SPEASY DEVELOPER DOCUMENTATION

4.1 Indices and tables

- genindex
- modindex
- search

CHAPTER

FIVE

HISTORY

5.1 0.10.0 (2022-02-03)

- Adds support for all AMDA products, even private ones
- Adds support for AMDA credentials
- Adds dynamic inventory for AMDA and SSC
- Adds possibility to set config values from ENV
- Drops Python 3.6 support and adds 3.10
- New API documentation using numpydoc
- New user documentation using numpydoc
- Most code examples are tested with doctest
- Renames SSCWeb module get_orbit to get_trajectory

5.2 0.9.1 (2021-11-25)

- Fix AMDA module bug #24 downloading multidimensional data fails

5.3 0.9.0 (2021-07-29)

- Adds SPWC migration tool
- Rename SpwcVariable to SpeasyVariable

5.4 0.8.3 (2021-07-28)

- Package renamed from SPWC to SPEASY
- Some doc and CI improvements

5.5 0.8.2 (2021-04-20)

- sscweb trajectories are always in km

5.6 0.8.1 (2021-04-18)

- Fixes minimum request duration for sscweb

5.7 0.8.0 (2021-04-18)

- Full support for trajectories and 0.2 proxy version

5.8 0.7.2 (2020-11-13)

- ccsweb/proxy: Fix missing coordinate system parameter

5.9 0.7.1 (2020-11-13)

- Fix project URL on PyPi

5.10 0.7.0 (2020-11-13)

- SSCWEB support to get satellites trajectories.
- Few bug fixes.
- Totally disabled cdf support for now.

5.11 0.1.0 (2019-12-07)

- First release on PyPI.

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/SciQLop/speasy/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

Space Physics WebServices Client could always use more documentation, whether as part of the official Space Physics WebServices Client docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/SciQLop/speasy/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *SPEASY* for local development.

1. Fork the *SPEASY* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/speasy.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv speasy
$ cd speasy/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
$ make test-all
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python from 3.6 to 3.9, and for PyPy. Check https://github.com/SciQLop/speasy/actions and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_speasy
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

GH Actions will then deploy to PyPI if tests pass.

CHAPTER

SEVEN

CREDITS

7.1 Development Lead

- Alexis Jeandet <alexis.jeandet@member.fsf.org>

7.2 Contributors

7.2.1 AMDA Webservice

- Alexandre Schulz <alexandre.schulz@irap.omp.eu>
- Benjamin Renard <benjamin.renard@irap.omp.eu>

Speasy is an open source Python client for Space Physics web services such as CDAWEB or AMDA. Most space physics data analysis starts with finding which server provides which dataset then figuring out how to download them. This can be difficult specially for students or newcomers, Speasy try to remove all difficulties by providing an unique and simple API to access them all. Speasy aims to support as much as possible web services and also cover a maximum of features they propose.

CHAPTER

EIGHT

QUICKSTART

Installing Speasy with pip (*more details here*):

```
$ pip install speasy
# or
$ pip install --user speasy
```

Getting data is as simple as:

```
import speasy as spz
ace_mag = spz.get_data('amda/imf', "2016-6-2", "2016-6-5")
```

Where amda is the webservice and imf is the product id you will get with this request.

Using the dynamic inventory this can be even simpler:

```
import speasy as spz
amda_tree = spz.inventory.data_tree.amda
ace_mag = spz.get_data(amda_tree.Parameters.ACE.MFI.ace_imf_all.imf, "2016-6-2", "2016-6-
↪5")
```

Will produce the exact same result than previous example but has the advantage to be easier to manipulate since you can discover available data from your favourite Python environment completion such as IPython or notebooks (might not work from IDEs).

This also works with SSCWEB, you can easily download trajectories:

```
import speasy as spz
sscweb_tree = spz.inventory.data_tree.ssc
solo = spz.get_data(sscweb_tree.Trajectories.solarorbiter, "2021-01-01", "2021-02-01")
```

CHAPTER

NINE

FEATURES

- Simple and intuitive API (spz.get_data to get them all)
- Pandas DataFrame like interface for variables
- Quick functions to convert a variable to a Pandas DataFrame
- Local cache to avoid repeating twice the same request
- Can take advantage of SciQLop dedicated poxy as a community backed ultra fast cache
- Full support of AMDA API.

CHAPTER

TEN

EXAMPLES

See *here* for a complete list of examples.

Go to developers doc