ALMA Science Archive School

Archive Query Tools

Toma Bădescu, Aida Ahmadi, George Bendo

OPTICON RadioNet Pilot

EUROPEAN ARC
ALMA Regional Centre

Query Tools

- ALMA Science Archive Web Interface
  - ESO, NRAO, NAOJ
- pyVO - ADQL queries through Table Access Protocol (TAP) service
  - Documentation
  - Examples for ALMA
- Astroquery
  - Documentation
- ALminer: ALMA Archive Mining & Visualization Toolkit
  - Documentation
  - Tutorial

pyVO

Python **V**irtual **O**bservatory

pyVO - Installation and setup

- **Py**thon **V**irtual **O**bservatory (pyVO) package for searching and accessing data from archives that use VO standards.
- officially supported by the ALMA archive developers, unlike astroquery.

Installation:

```
>>> pip install pyvo
```

To use pvVO, the package needs to be imported into python using

```
>>> import pyvo
```

Access to the archive needs to be set up using

```
>>> service = pyvo.dal.TAPService("https://almascience.eso.org/tap")
```

pyVO - Searches

Once pyVO is set up, searches are typically done in two lines of code

- The first line sets up input for a query
- The second line executes the search

For example, the following code performs a cone search in the ALMA archive at the coordinates of RA=204.253958 and Dec=-29.865417 and with a radius of 0.006 degrees:

```
>>> query =f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE \
            INTERSECTS(CIRCLE('ICRS',204.253958,-29.865417,0.006),s_region)=1\
            """
>>> output = service.search(query).to_table().to_pandas()
```

The object output is a pandas.DataFrame .

pyVO - Display Search Results

Search results can be displayed in multiple ways.

For example, the Proposal IDs associated with a search can be displayed using the following:

```
>>> print(output['proposal_id'])
```

The RA and Declination of all fields identified from a search can be plotted using the following:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(output['s_ra'],output['s_dec'],marker='o',linestyle='none')
>>> plt.xlabel('RA')
>>> plt.ylabel('Dec’)
```

For reference, the columns in the table from a search can be displayed using the following:

```
>>> print(output.columns)
```

pyVO - Search Parameters

pyVO allows for searching using a variety of parameters.

In this example, pyVO is used to search by the ALMA Program ID:

```
>>> query = f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE proposal_id like '%2018.1.01131.S%'\
            """
>>> output = service.search(query).to_table().to_pandas()
```

In this next example, pyVO is used to search by polarization mode for full polarization data:

```
>>> query = f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE pol_states like '%/XX/XY/YX/YY/%' \
            """
>>> output = service.search(query).to_table().to_pandas()
```

pyVO - Advanced Search

More generalized searches are possible. This search will return all the results from Cycle 6 (Program IDs beginning with 2018):

```
>>> query = f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE proposal_id like '%2018.1.01131.S%'\
            """
>>> output = service.search(query).to_table().to_pandas()
```

Frequency searches use their own syntax, as shown in this example:

```
>>> query = f"""
            SELECT member_ous_uid, target_name, frequency, bandwidth
            FROM ivoa.obscore
            WHERE (frequency - 0.5 * bandwidth/1e9) < {115.35}
            AND (frequency + 0.5 * bandwidth/1e9) > {115.20}
            """
>>> output = service.search(query).to_table().to_pandas()
```

pyVO - Advanced Search

pyVO can be combined with astropy.coordinates to resolve a source name and search at that position, as shown in this example:

```
>>> import astropy
>>> coordinates = astropy.coordinates.SkyCoord.from_name("Cen A")
>>> ra=float(coordinates.ra.degree)
>>> dec=float(coordinates.dec.degree)
>>> query = f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE INTERSECTS(CIRCLE('ICRS',{ra},{dec},0.006),s_region)=1 \
            """
>>> output = service.search(query).to_table().to_pandas()
```

pyVO - Advanced Search

This is different from searching for the ALMA source name (the name assigned when the proposal was written). A search by the ALMA search name is shown below. Note that this search may not return all of the observations of the target but just the ones using this specific name!

```
>>> query = f""" \
            SELECT * \
            FROM ivoa.obscore \
            WHERE target_name like '%{CenA}%' \
            """
>>> output = service.search(query).to_table().to_pandas()
```

pyVO - Advanced Search

It's also possible to combine search criteria, as shown in this example that searches the archive for all Band 3 full polarization data from Cycle 7:

```
>>> query = f"""SELECT * FROM ivoa.obscore WHERE proposal_id like '%2019%' \
             AND band_list like '3' \
             AND pol_states like '%/XX/XY/YX/YY/%' \
             """
>>> output = service.search(query).to_table().to_pandas()
```

Here is another example showing how to combine a position and frequency search:

```
>>> query = f"""SELECT member_ous_uid, target_name, frequency, bandwidth \
            FROM ivoa.obscore   \
            WHERE INTERSECTS(CIRCLE('ICRS',204.253958,-29.865417,0.006),s_region)=1
\
            AND (frequency - 0.5 * bandwidth/1e9) < {115.15} \
            AND (frequency + 0.5 * bandwidth/1e9) > {115.00} \
            """
>>> output = service.search(query).to_table().to_pandas()
```

pyVO - Downloading Data

Data can be downloaded using the following example code:

```
>>> import os
>>> datalink = pyvo.dal.adhoc.DatalinkResults.from_result_url( \
  f"https://almascience.eso.org/datalink/sync?ID={'uid___A001_X135b_X6b'}")
>>> for dl in datalink:
>>>       dl.cachedataset(filename=os.path.basename(dl['access_url']))
```

The Member OUSs in the input for this command need to be formatted with underscores (_), whereas the member_ous_id column from pyVO searches are in a different format. The colon (:) and slashes (/) in the pyVO search results need to be replaced with underscores in this command, as shown in this example:

pyVO search output: uid://A001/X135b/X6b

pyVO download string: uid___A001_X135b_X6b

astroquery

Basic astroquery commands for searching the archive by source name, location, observing configuration, and more

Quick intro: python lists and list comprehension

List Slices:

```
A = [1,2,3,5,10]
print(A[:3])
[1,2,3]
print(A[0])
>>>  1
```

Numpy arrays:

```
A = np.array([1,2,3])
A[[True,True,False]]
>>> [1,2]
```

For loops:

Get only elements greater than 4 from A.

```
B = [a for a in A if a > 4]
```

Is equivalent to:

```
B=[]
for a in A:
    If a > 4:
        B.append(a)
```

Nested for loops:

Get lists in A = [[1,2,5],[0,1,3],[4,5,6]] that contain 2.

```
B = [aa for aa in A if any([a == 2 for a in aa])]
```

Is equivalent to:

```
Flag = False
B = []
for aa in A:
    for a in aa:
        if a == 2:
            Flag = True
            break
    if Flag: B.append(aa)
    Flag = False
```

Quick intro: SkyCoords, units, astropy tables

Units class defines and transforms units:

```
from astropy.coordinates import SkyCoord
from astropy import units as u
angle = 10 * u.deg
print(angle)
print(angle.value)
print(angle.to(u.arcmin))
10.0 deg
10.0
600.0 arcmin
force = 1*u.N
area = 1*u.m**2
print(force/area)
>>> 1. N / m2
print((force/area).to(u.Pa) )
>>> 1. Pa
```

- Units package: variable type, a python quantity
- Value and unit can be accessed separately:

var.value and var.unit

- Automatic conversion respecting laws of physics

u.N/u.m**2 -> u.Pa

- Exercise: write a quantity using units for energy, time, area, and frequency and convert it to Jy.

Check the help page for astropy.units and never be off by a factor 1000 because of some miliJansky ever again! :)

Quick intro: SkyCoords, units, astropy tables

SkyCoords class defines and transforms astronomical coordinates and operations with coordinates:

```
from astropy.coordinates import SkyCoord
from astropy import units as u
c = SkyCoord(ra=10.625*u.degree, dec=41.2*u.degree,
frame='icrs')
c = SkyCoord(10.625, 41.2, frame='icrs', unit='deg')
c = SkyCoord('00h42m30s', '+41d12m00s', frame='icrs')
c = SkyCoord('00h42.5m', '+41d12m')
c = SkyCoord('00 42 30 +41 12 00', unit=(u.hourangle,
u.deg))
c = SkyCoord('00:42.5 +41:12', unit=(u.hourangle, u.deg))
```

There are multiple valid ways of defining coordinates. Functions exist to calculate distance between coordinates, transform from one reference frame or epoch to another, etc. Check the help page for astropy.coordinates.

Quick intro: SkyCoords, units, astropy tables

Astropy Tables store data in columns, each column having a name.

```
from astropy.table import Table
import numpy as np
a = [0.1245, 0.456]
b = [124.234, 231.234]
c = [3.1, 5.6]
ex_table= Table([a,b,c],names = ("ra","dec","dist"))
# There are multiple valid ways of defining a table
new_table = Table()
new_table["ra"] = [0.1245, 0.456]
new_table["dec"] = [124.234, 231.234]
new_table["dist"] = [3.1, 5.6]
# It's easy to add a new column,e.g. "L" column
ex_table["L"] = [9.84,9.95]
```

- Data from a table column can be retrieved by using

table_name["column_name"]

- Using indices retrieves the ith+1 row:

table_name[0] retrieves the first row.

- To see a list of all table columns:

print(table_name.colnames)

For more information on tables, check the help page.

First step: authentication

Authentication with an ALMA central authentication center (CAS) is not required to perform general queries, but it is needed to retrieve proprietary data.

Astroquery module version **0.4.7**.dev8076 has to be installed.

```
from astroquery.alma import alma = Alma()
```

```
alma.login("username")
```

Instantiates the Alma Class, initiates login with “username” (optional). You will be asked for your password.

Query by object name

This will search the ALMA archive for products targeting an object.

The object name is retrieved by astropy, looking in 'simbad', 'ned', or 'vizier'.

```
result = alma.query_object('m83', public=True, science=True, payload=None)
```

parameters:

public - boolean, retrieve only public data

science - boolean, retrieve only science observation

payload - dict, to pass additional constraints to the query, will be discussed later

The result is an astropy table.

Query a Region

User can search for all observations that cover a region around a central position

```
from astropy import units as u
from astropy import coordinates
galactic_center = coordinates.SkyCoord(0*u.deg, 0*u.deg, frame='galactic')
results = Alma.query_region(coordinates = galactic_center,radius = 1*u.deg)
results = Alma.query_region("17h45m40.04s −29d00m28.1s", 1)
results = Alma.query_region("17 45 40.04  −29 00 28.1", 1)
results = Alma.query_region("17:45:40.04  −29:00:28.1", 1)
```

coordinates - gives central position can be SkyCoord object or string of coordinates

radius - can be a number (automatically converted to degrees) or an astropy.unit for angle. All above queries are equivalent.

An extra dict object can be passed to a payload parameter for additional constraints

Other parameters and the payload dictionary

All query commands accept additional parameters. These can be encapsulated in a python dictionary or simply added as function parameters. The keys for the alma archive are given by the Alma.help() command.

```
result = alma.query_region('M83', radius=25*u.arcmin, pi_name='*Smith*')
```

Retrieve all observations following the given constraints, having a PI whose name contains the word “Smith”

```
extra_params = dict(band_list=[3,7],sensitivity_10kms=”<10”)
result = alma.query_region('M83', radius=25*u.arcmin, payload = extra_params)
```

Query around M83, for band 3 and 7 observations, with a sensitivity at a resolution of 10km/s better than 10 mJy/beam.

Other parameters and the payload dictionary

All astroquery search parameters:

Source name (astropy Resolver)	source_name_resolver
Source name (ALMA)	source_name_alma
RA Dec (Sexagesimal)	ra_dec
Galactic (Degrees)	galactic
Angular resolution (arcsec)	spatial_resolution
Largest angular scale (arcsec)	spatial_scale_max
Field of view (arcsec)	fov
Frequency (GHz)	frequency
Bandwidth (Hz)	bandwidth
Spectral resolution (KHz)	spectral_resolution
Band	band_list
Observation date	start_date
Integration time (s)	integration_time
Pol type (Single, Dual, Full)	polarisation_type
Line sens 10 km/s (mJy/beam)	line_sensitivity
Continuum sensitivity (mJy/beam)	continuum_sensitivity
Water vapour (mm)	water_vapour
Project code	project_code
Project title	project_title
PI name	pi_name
Proposal authors	proposal_authors
Project abstract	project_abstract
Publication count	publication_count
Science keyword	science_keyword
Bibcode	bibcode
Title	pub_title
First author	first_author
Authors	authors
Abstract	pub_abstract
Year	publication_year
Public data only	public_data
Science observations only	science_observations

Filtering information after retrieval

Downloaded astropy tables have more categories (columns) than search parameters available for an Alma astroquery.

Feasible to retrieve a large table from a more general query and filter it in the python script

Example: check frequency coverage

```
from astroquery.utils import parse_frequency_support

m83_data = alma.query_object('M83')

m83_data['freq_cover'] = [parse_frequency_support(item['frequency_support']) for item in m83_data]

our_freq = 95 * u.GHz

new_m83_data = m83_data[[any([item[0] < our_freq < item[1] for item in row]) for row in m83_data['freq_cover']]]
```

parse_frequency_support can take a string from the 'frequency_support' column and transform it in to a list of arrays, each array containing the start and end frequency of each spectral window, i.e. item[0] and item[1]

new_m83_data now contains only m83_data table rows where our_freq is observed

All m83_data categories stored in: m83_data.colnames

Downloading Archival Data

```
m83_data = alma.query_object('M83')
uids = np.unique(m83_data['member_ous_uid'])
link_list = alma.get_data_info(uids[:3],
expand_tarfiles=True)
alma.cache_location = '/big/external/drive/'
alma.download_files(link_list, cache=True)
```

Or:

```
alma.retrieve_data_from_uid(uids[0])
```

Downloadable data sets are uniquely identified by their observed unit set id (“member_ous_id”).

Queries can return multiple results corresponding to the same ous_id, in the case of a project with multiple observing configurations for a science goal.

The uids variable here holds only unique ous_ids, after applying np.unique to the list of uids given by m83_data['member_ous_uid'].

Incomplete downloads are held at cache_location.

Downloaded files are extracted if expand_tarfiles is set to True.

Downloading Archival Data

Download only the fits files from a project, e.g.,
the first one on the uids list:

```
link_list = alma.get_data_info(uids[0], expand_tarfiles=True)
fits_urls = [url for url in link_list if '.fits' in url]
filelist = alma.download_files(fits_urls)
```

link_list is a list of links to each file in the project identified by the ous_id stored at uids[0]

fits_urls contains only links to files with the “*.fits” ending

data files are downloaded from those links in fits_urls

Query with ADQL through astroquery

Most basic usage:

```
Query = “select * from ivoa.obscore where frequency > 100 and frequency < 120 and science_keyword like ‘*galaxies*’ ”
results = alma.query_tap(Query)
```

Selects all entries from the database that follow the constraints set after the keyword where. Multiple constraints can be chained with the and keyword.

The keywords can be displayed with the alma.help_tap() command.

Installation

pip install alminer

Dependencies

numpy
matplotlib
pandas
pyvo
astropy
astroquery

ALminer

ALMA archive mining and visualization toolkit

Documentation: https://alminer.readthedocs.io/

ALminer

ALMA archive mining and visualization toolkit

Python-based code to effectively **query**, **analyse**, and **visualise** the ALMA Science Archive + **download** ALMA data products and/or raw data

Documentation: https://alminer.readthedocs.io/

Tutorial Jupyter Notebook: launch Jupyter Notebook

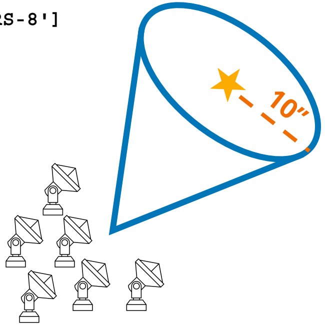
I-TRAIN video: https://bit.ly/ALminer_I-TRAIN_video

Query

- **Target name** and a **search radius** around them:

```
myquery = alminer.target(["Orion KL", "AB Aur"], search_radius=5.0)
```


```
================================
alminer.target results
================================
Target = Orion KL
--------------------------------
Number of projects = 35
Number of observations = 91
Number of unique subbands = 287
Total number of subbands = 403
40 target(s) with ALMA data = ['Orion KL', 'Orion H2O maser outburst', 'OrionField1-2', 'OrionField1-1', 'OrionKL',
'orion-IRc2', 'f1', 'f3', 'f14', 'f12', 'f11', 'f15', 'f13', 'f10', 'Orion_Source_I', 'OMC1_SE', 'orion_kl', 'BN',
'OMC1_NW', 'BN-KL', 'Orion_KL', 'f23', 'OrionKL-SV', 'OMC-1', 'ONC', 'Orion_BNKL_source_I', 'Orion', 'OMC-1_Region5',
'OMC-1_Region2', 'OMC-1_Region4', '104', 'HC602_HC606_HC608', 'Orion_KL_Field_1_Orion_Hot_Core',
'Orion_KL_Field_3_North-west_Clump', 'Orion_KL_Field_2_SMA1', 'ONC_Mosaic', 'f16', 'Orion1', 'Orion-KL', 'ORS-8']
--------------------------------
Target = AB Aur
--------------------------------
Number of projects = 3
Number of observations = 3
Number of unique subbands = 17
Total number of subbands = 17
3 target(s) with ALMA data = ['AB_Auriga', 'AB_Aur', 'ab_aurigae']
--------------------------------
```



- **Positions** in the sky and a **search radius** around them:

```
alminer.conesearch(ra=201.365063, dec=-43.019112, search_radius=10.0)
```

Query

- **Any (string-type) keywords** defined in ALMA TAP system

  Very powerful tool for querying topics of interest, especially when keywords are combined!

words in quotations are interpreted as a 'PHRASE'

```
alminer.keysearch({"proposal_abstract":[" 'high-mass star formation' "]})
```

PHRASE

spaces are interpreted with 'AND' logic

```
alminer.keysearch({"proposal_abstract":[" 'high-mass star formation' outflow disk "]})
```

PHRASE AND AND

when multiple values are provided for a given keyword, they are queried using 'OR' logic

```
alminer.keysearch({"proposal_abstract":[" 'high-mass star formation' ", " 'massive star formation' "]})
```

PHRASE OR PHRASE

when multiple keywords are provided, they are queried using 'AND' logic

```
alminer.keysearch({"proposal_abstract":["'star formation'"], "scientific_category":["'Galaxy evolution'"]})
```

PHRASE AND PHRASE

Analyze

- Query results are in the form of **PANDAS DataFrame** that can be used to further narrow down the search

```
observations = alminer.keysearch({'science_keyword':["'Galaxy chemistry'"]})
```

```
selected = observations[(observations['ang_res_arcsec'] < 0.5) &
                        (observations['vel_res_kms'] < 1.0)]
```

- **Line coverage** based on frequency

```
alminer.line_coverage(observations,
                      line_freq=220.5,
                      z=0,
                      line_name="My favourite line",
                      print_targets=True)
```

```
--------------------------------
Summary of 'My favourite line' observations at 220.5 GHz
--------------------------------
Number of projects = 6
Number of observations = 7
Number of unique subbands = 7
Total number of subbands = 7
5 target(s) with ALMA data = ['Arp220', 'ngc_3256',
'IRAS_13120-5453', 'ngc253', 'NGC_253']
--------------------------------
```

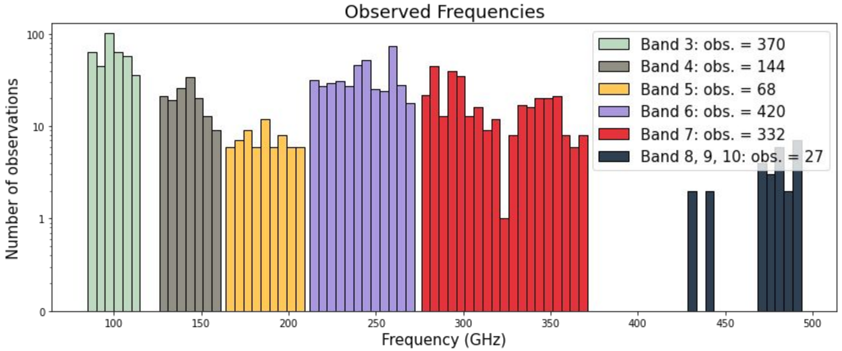
- Coverage of **CO, ^{13}CO, C^{18}O** lines

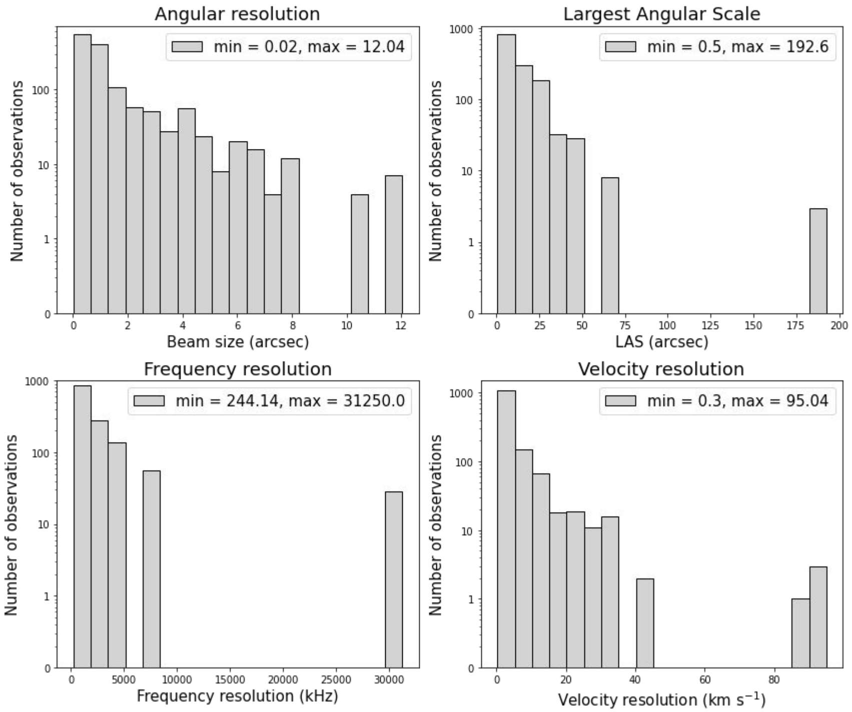
```
alminer.CO_lines(observations, z=1, print_targets=True)
```

Visualize

- Plot an **Overview** of the observations

```
alminer.plot_overview(observations, savefig="galaxy_chemistry")
```

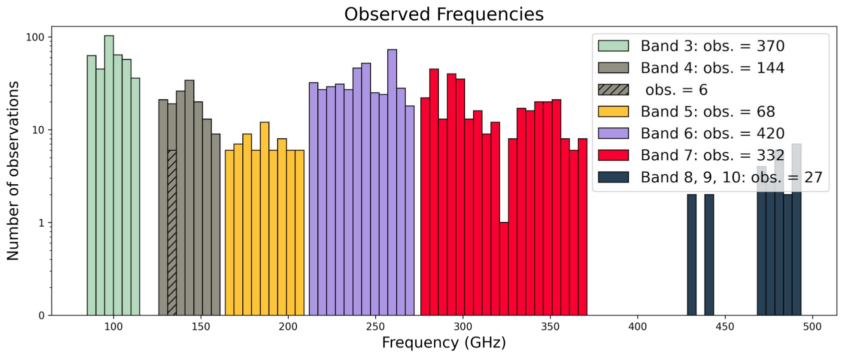


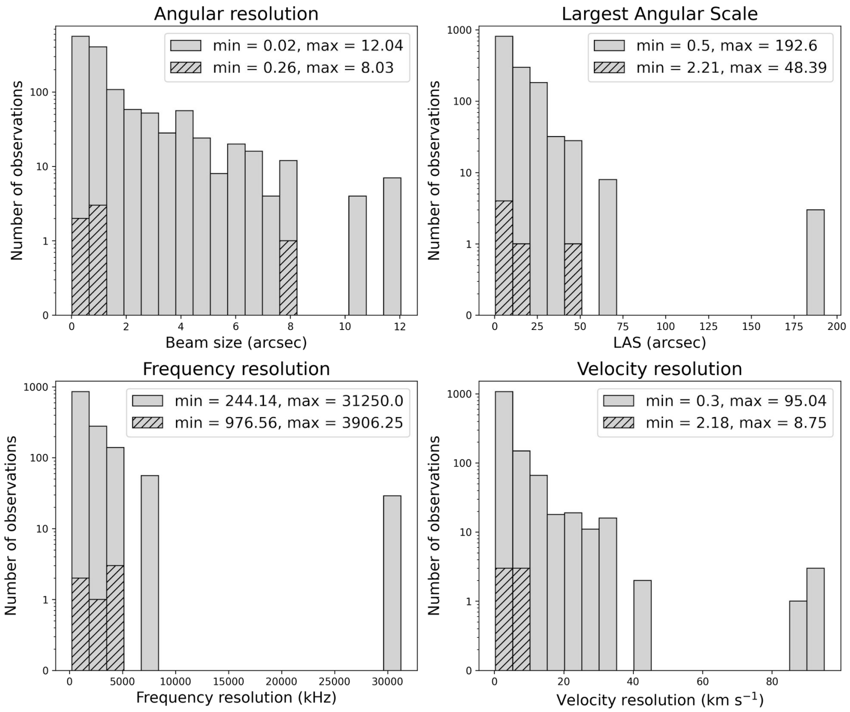


Visualize

- Plot an **overview** of the observations and highlight a given **frequency**

```
alminer.plot_line_overview(observations, line_freq=400.0, z=2)
```

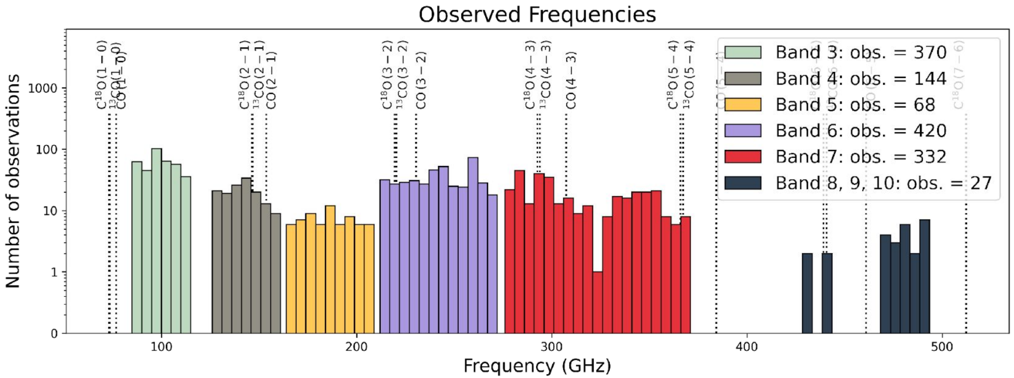


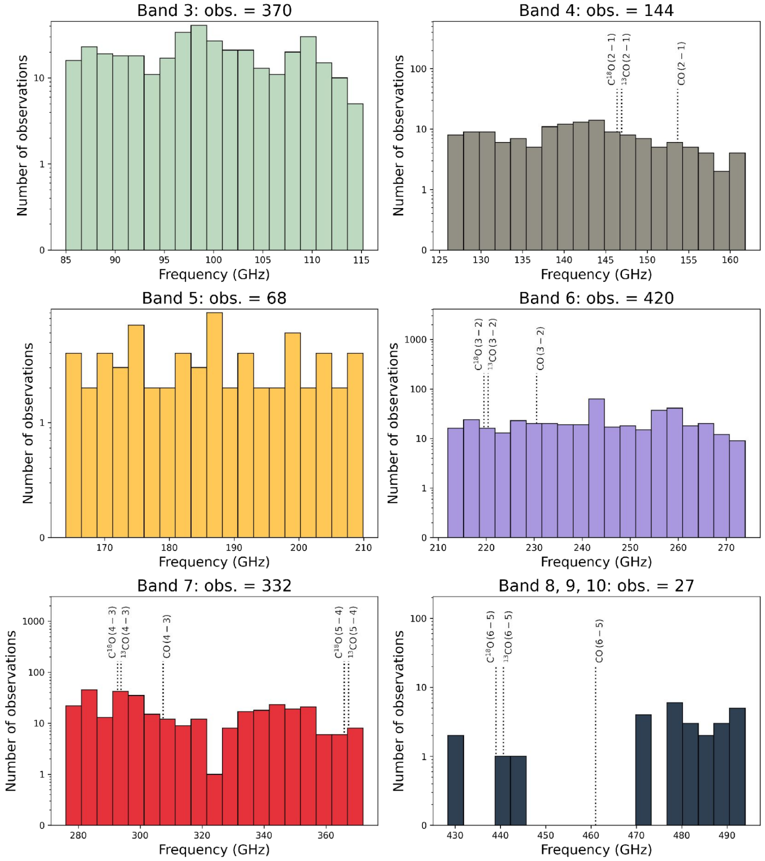


Visualize

- Plot an overview of the **observed frequencies in each band and highlight CO lines**

```
alminer.plot_bands(observations,
                   mark_CO=True,
                   z=0.5)
```





And more!

- **Save** the **DataFrames** and **plots**
- **Advance query options** through TAP
- Download **raw/products** data

```
alminer.download_data(observations, fitsonly=True, dryrun=True, location='./data',
                      filename_must_include=['_sci', '.pbcor', 'cont'], print_urls=True)
```

ALminer resources

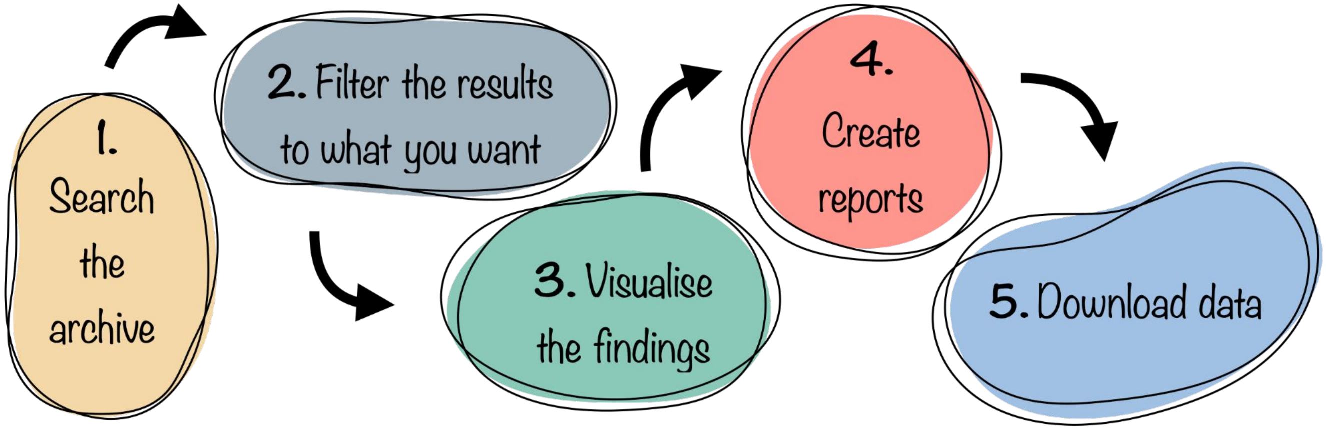
Documentation: https://alminer.readthedocs.io/

GitHub: https://github.com/emerge-erc/ALminer

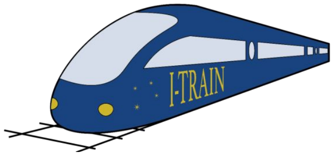
Extensive tutorial

Static version at https://alminer.readthedocs.io/

Live Jupyter Notebook launch Jupyter Notebook



I-TRAIN video: https://bit.ly/ALminer_I-TRAIN_video

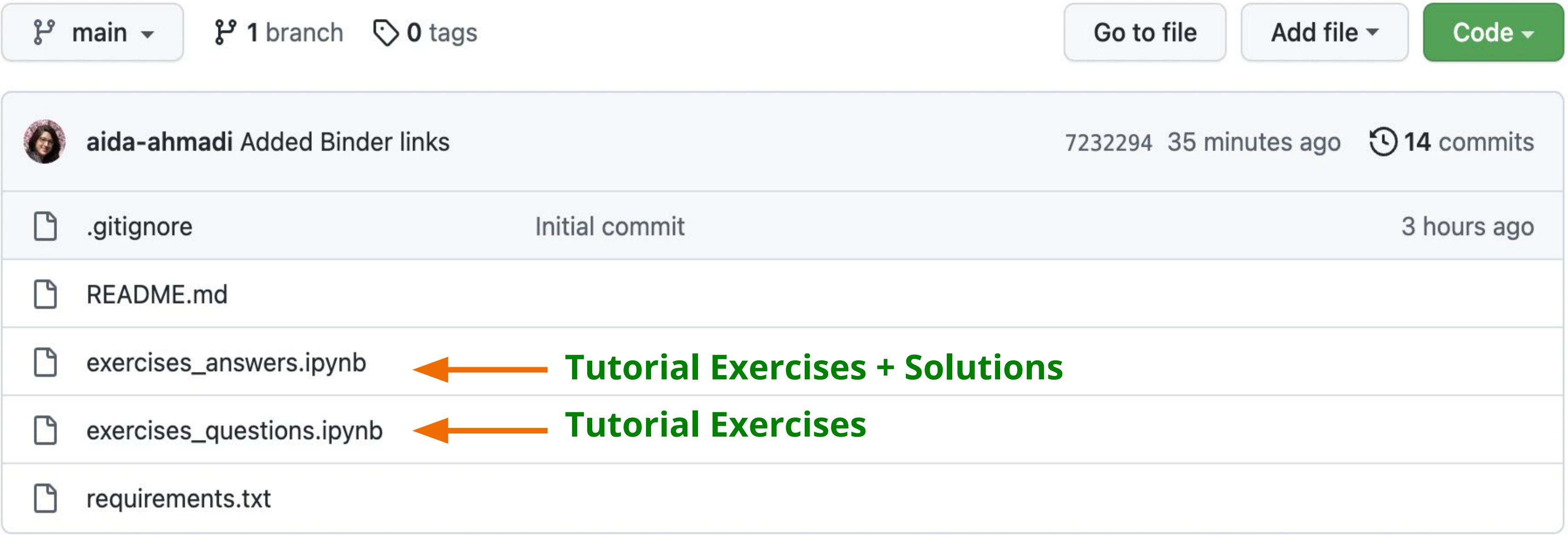


Exercises

Hands-on Session

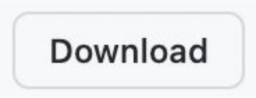
Go to the following link: https://github.com/aida-ahmadi/ASA-School-2022

Go to the following link: https://github.com/aida-ahmadi/ASA-School-2022



Download notebooks

Click on a notebook to open

Right click on 

Save Link As...

Go to the following link: https://github.com/aida-ahmadi/ASA-School-2022

See static notebooks

Tutorial Jupyter Notebook (Static versions)

Exercises

Exercises + Solutions

Work on the cloud

Tutorial Jupyter Notebook (Interactive versions)

Exercises

Exercises + Solutions

✅ **Benefit**: No need to install any of the packages & Jupyter Notebook

❌**Caveat**: It times out if you are inactive for too long