Astroinformatics 2022
Session 13: Planetary Motion and Orbital Integration

Kinoshita Daisuke

12 December 2022
publicly accessible version

About this file...

- Important information about this file
  - The author of this file is Kinoshita Daisuke.
  - The original version of this file was used for the course "Astroinformatics" (course ID: AS6095) offered at Institute of Astronomy, National Central University from September 2022 to January 2023.
  - The file is provided in the hope that it will be useful, but there is no guarantee for the correctness. Use this file at your own risk.
  - If you are willing to use this file for your study, please feel free to use. I'll be very happy to receive feedback from you.
  - If you are willing to use this file for your teaching, please contact to Kinoshita Daisuke. When you use this file partly or entirely, please mention clearly that the author of the original version is Kinoshita Daisuke. Please help me to improve the contents of this file by sending your feedback.
  - Contact address: `https://www.instagram.com/daisuke23888/`

For this session, we study orbital motions of planets and asteroids. Also, we try orbital integration using the Python module named "`rebound`".

1 Sample Python scripts for this session

Sample Python scripts for this session can be downloaded from GitHub repository. Visit following GitHub repository.

- `https://github.com/kinoshitadaisuke/ncu_astroinformatics_202209`

1.1 Executing sample Python scripts on a terminal emulator

If you prefer to execute sample Python scripts for this session on a terminal emulator, download `.py` files from GitHub repository.

1.2 Executing sample Python scripts on JupyterLab

If you prefer to execute sample Python scripts for this session on JupyterLab (or Jupyter Notebook), download `.ipynb` file from GitHub repository.

1.3 Executing sample Python scripts using Binder

If you prefer to execute sample Python scripts for this session on Binder, visit following web page.

- `https://mybinder.org/v2/gh/kinoshitadaisuke/ncu_astroinformatics_202209/HEAD`

Start your favourite web browser and go to above web page. (Fig. 1) In a minute or two, you see JupyterLab working on your web browser. (Fig. 2) Go to the directory (folder) "`s12`". (Fig. 3) Choose the file "`ai202209_s12.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

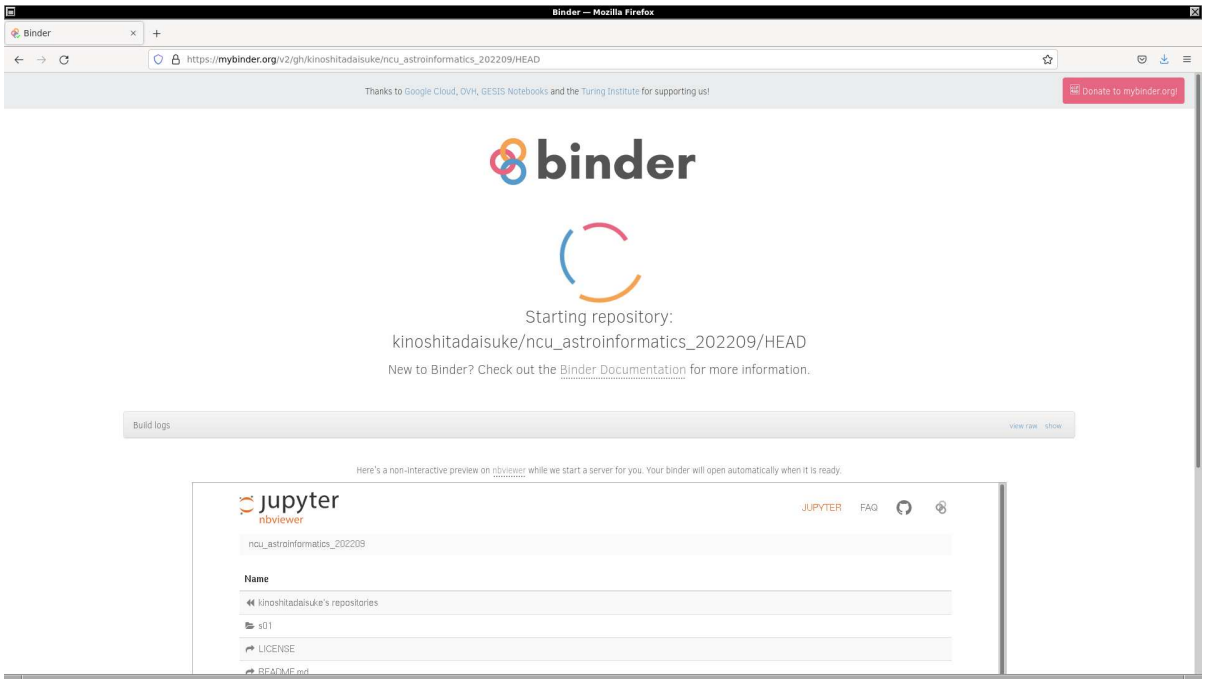

Figure 1: Using Binder to execute sample Python scripts for this session.

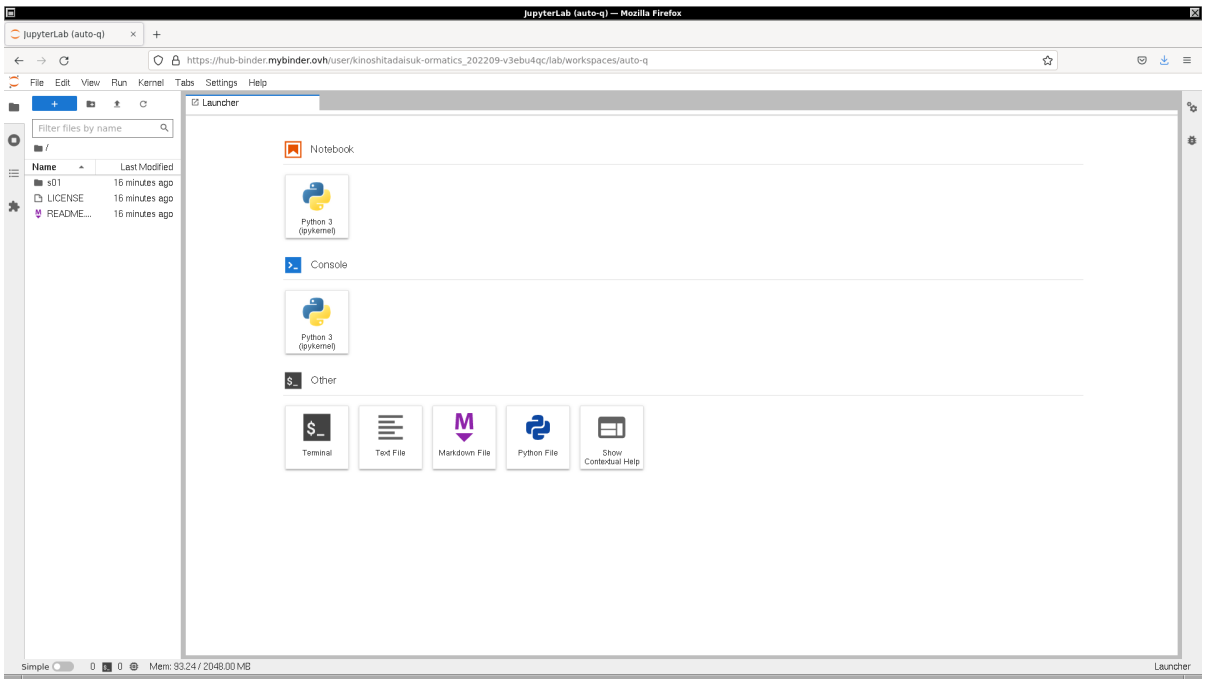

Figure 2: Using Binder to execute sample Python scripts for this session.

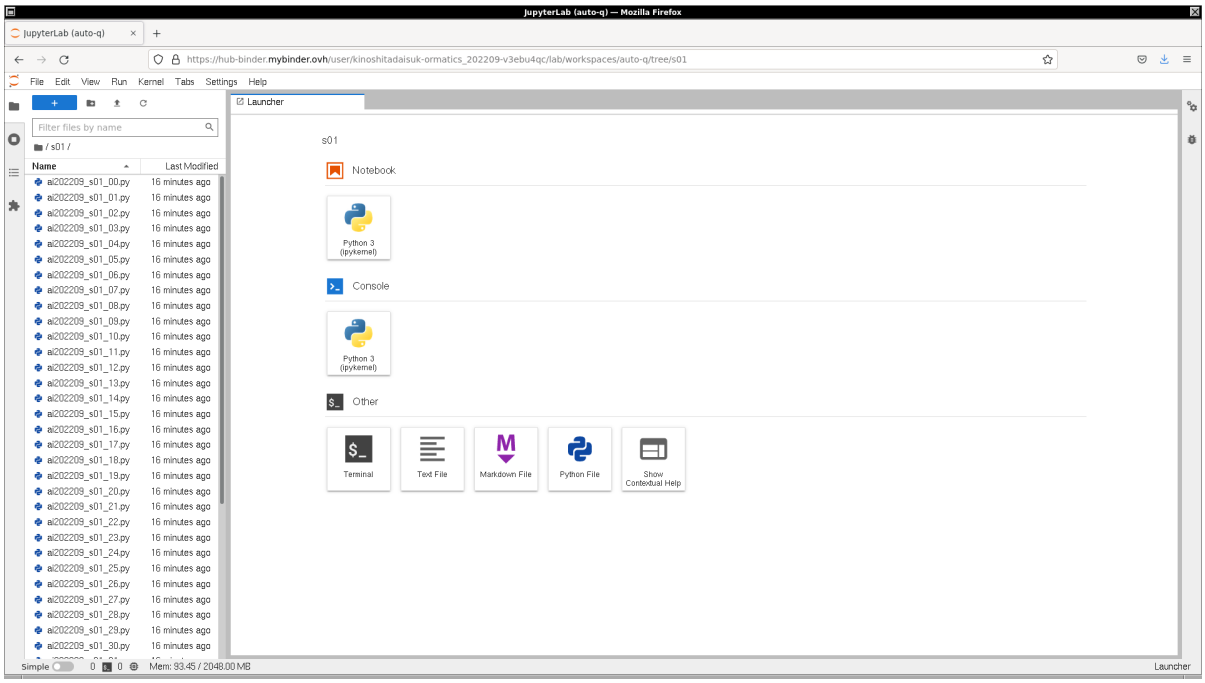

Figure 3: Using Binder to execute sample Python scripts for this session.

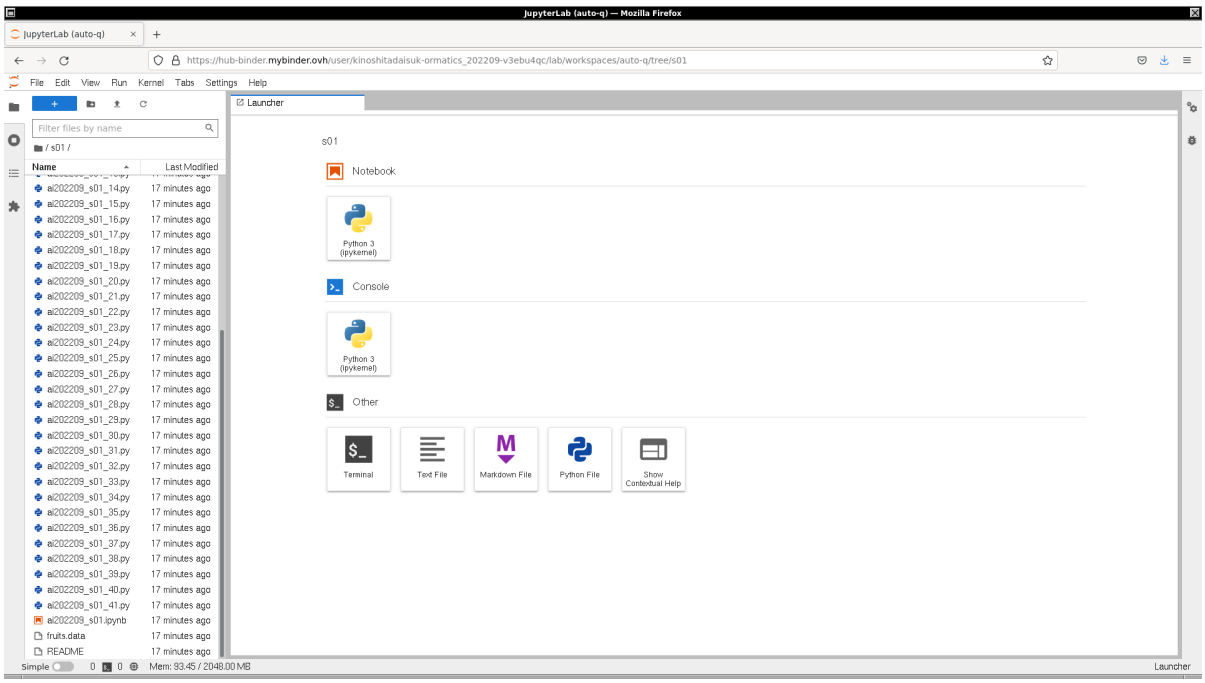

Figure 4: Using Binder to execute sample Python scripts for this session.

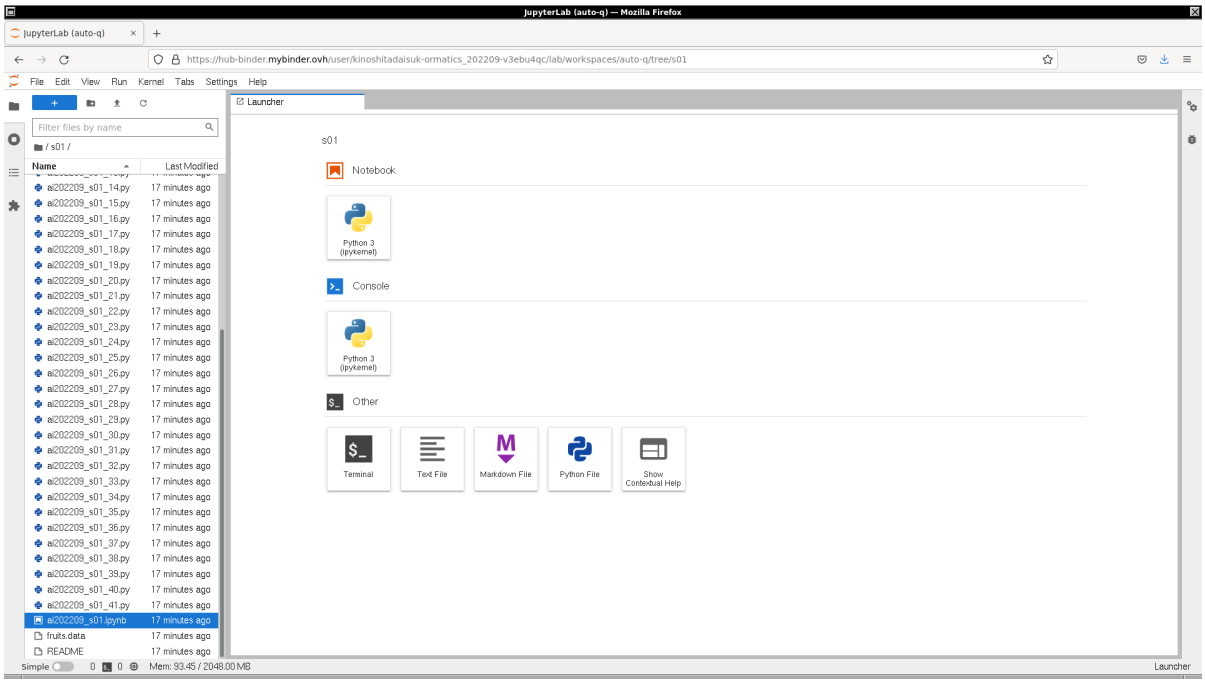

Figure 5: Using Binder to execute sample Python scripts for this session.

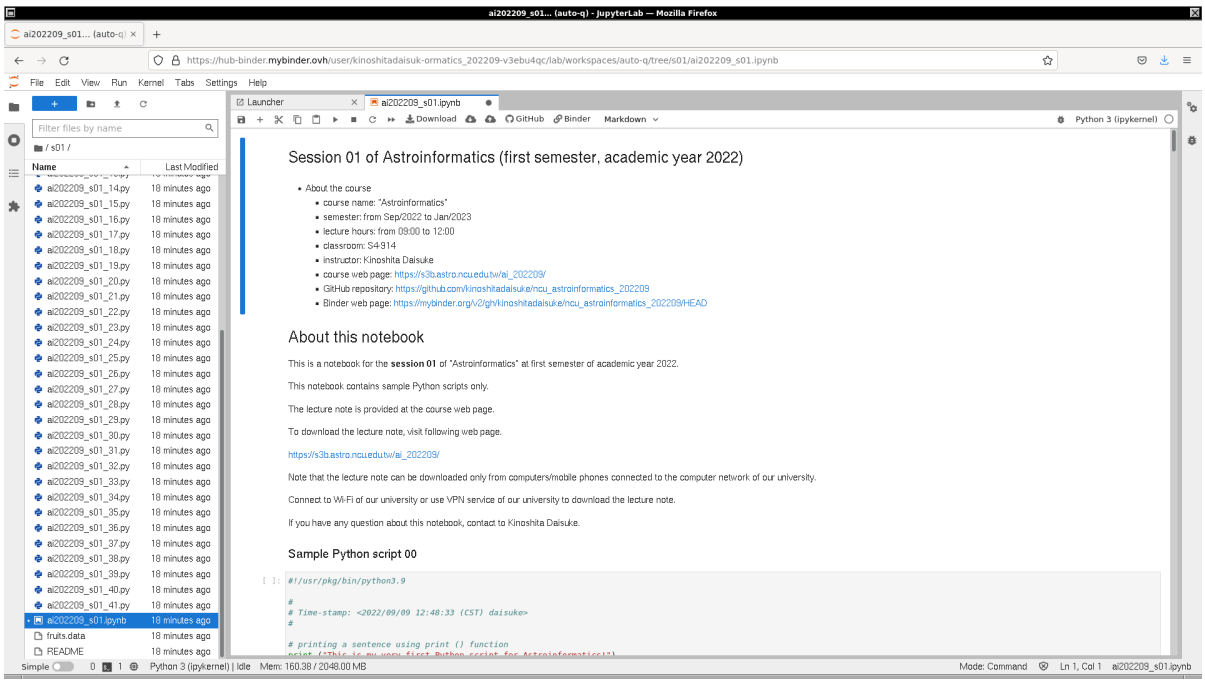

Figure 6: Using Binder to execute sample Python scripts for this session.

2 Check of availability of required Python packages

2.1 Availability check of `rebound` package

For this session, "`rebound`" package is needed. Check whether you have "`rebound`" package properly installed on your computer.

Python Code 1: ai202209_s13_00_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/08 14:15:20 (CST) daisuke>
#

# check of availability of rebound module
try:
    # importing rebound module
    import rebound
except:
    # if rebound module is not installed, print an error message
    print (f"The module 'rebound' is not installed on your computer.")
    print (f"The module 'rebound' is required for this session.")
    print (f"Visit following web page and install the package 'rebound'.")
    print (f"  https://rebound.readthedocs.io/")
    print (f"After the installation, try to run this script again.")
else:
    # if rebound module is found, print following message
    print (f"The module 'rebound' is found on your computer.")
finally:
    # print that the check of availability of rebound module is finished
    print (f"A test of availability check of 'rebound' module is now finished.")
```

Execute above script to check the availability of "`rebound`" package. If you do not have "`rebound`" package installed on your computer, you see following message.

```
% chmod a+x ai202209_s13_00_00.py
% ./ai202209_s13_00_00.py
The module 'rebound' is not installed on your computer.
The module 'rebound' is required for this session.
Visit following web page and install the package 'rebound'.
  https://rebound.readthedocs.io/
After the installation, try to run this script again.
A test of availability check of 'rebound' module is now finished.
```

If you see above message, install "`rebound`" package, and try to execute the script again after the installation.
If you have "`rebound`" package, you see following message.

```
% ./ai202209_s13_00_00.py
The module 'rebound' is found on your computer.
A test of availability check of 'rebound' module is now finished.
```

Try following practice.

Practice 13-01

Make a Python script to check the availability of "`numpy`" module.

About the exception handling, read the Section "Exceptions" of the session 01 "Basic Python Programming".

2.2 More flexible Python script for availability check of packages

Make a more flexible Python script for availability check of Python packages. Use "`argparse`" module. Here is an example.

Python Code 2: ai202209_s13_00_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 00:02:32 (CST) daisuke>
#

# importing argparse module
import argparse

#
# command-line argument analysis
#

# constructing parser object
desc = f"availability check of Python modules"
parser = argparse.ArgumentParser (description=desc)

# adding options
parser.add_argument ('module', type=str, nargs='+', \
                     help=f"module name (e.g. astropy)")

# analysis of command-line arguments
args = parser.parse_args ()

#
# input parameters
#

# list of module names for availability check
list_modules = args.module

#
# availability check of modules
#

for module in list_modules:
    # check of availability of rebound module
    try:
        # importing module
        imported = __import__ (module)
    except:
        # if rebound module is not installed, print an error message
        print (f"The module '{module}' is NOT installed on your computer.")
    else:
        # if rebound module is found, print following message
        print (f"The module '{module}' is found on your computer.")
        print (f"{imported}")
    finally:
        # print that the check of availability of rebound module is finished
        print (f"A test of availability check of '{module}' module is now finished.")
        print (f"")
```

Execute above script to check availability of Numpy package.

```
% chmod a+x ai202209_s13_00_01.py
% ./ai202209_s13_00_01.py -h
usage: ai202209_s13_00_01.py [-h] module [module ...]

availability check of Python modules

positional arguments:
  module      module name (e.g. numpy)

optional arguments:
  -h, --help  show this help message and exit

% ./ai202209_s13_00_01.py numpy
The module 'numpy' is found on your computer.
<module 'numpy' from '/usr/pkg/lib/python3.9/site-packages/numpy/__init__.py'>
A test of availability check of 'numpy' module is now finished.
```

Execute the script to check availability of both "`numpy`" module and "`scipy`" module.

```
% ./ai202209_s13_00_01.py numpy scipy
The module 'numpy' is found on your computer.
<module 'numpy' from '/usr/pkg/lib/python3.9/site-packages/numpy/__init__.py'>
A test of availability check of 'numpy' module is now finished.

The module 'scipy' is found on your computer.
<module 'scipy' from '/usr/pkg/lib/python3.9/site-packages/scipy/__init__.py'>
A test of availability check of 'scipy' module is now finished.
```

Execute the script to check availability of "`astropy`", "`astroquery`", "`photutils`", and "`ginga`".

```
% ./ai202209_s13_00_01.py astropy astroquery photutils ginga
The module 'astropy' is found on your computer.
<module 'astropy' from '/usr/pkg/lib/python3.9/site-packages/astropy/__init__.py
'>
A test of availability check of 'astropy' module is now finished.

The module 'astroquery' is found on your computer.
<module 'astroquery' from '/usr/pkg/lib/python3.9/site-packages/astroquery/__ini
t__.py'>
A test of availability check of 'astroquery' module is now finished.

The module 'photutils' is NOT installed on your computer.
A test of availability check of 'photutils' module is now finished.

The module 'ginga' is NOT installed on your computer.
A test of availability check of 'ginga' module is now finished.
```

Try following practice.

Practice 13-02

Make your own Python script to check the availability of Python modules. Use "`argparse`" module.

3 Using NASA/JPL Horizons System

NASA/JPL Horizons System can be used to retrieve information of solar system bodies.

3.1 Using NASA/JPL Horizons System on a web browser

Start your favourite web browser, such as Firefox. Visit NASA/JPL Horizons System's web page. (Fig. 7)

- Horizons System: `https://ssd.jpl.nasa.gov/horizons/`

Click the link "App" to go to the web interface of the Horizons System. (Fig. 8) Push the button "Generate Ephemeris" to generate the ephemeris. (Fig. 9) Scroll down and find RA and Dec of the target body. (Fig. 10)

Change the settings "Ephemeris Type", "Target Body", "Observer Location", "Time Specification", and "Table Settings" and then generate the ephemeris.

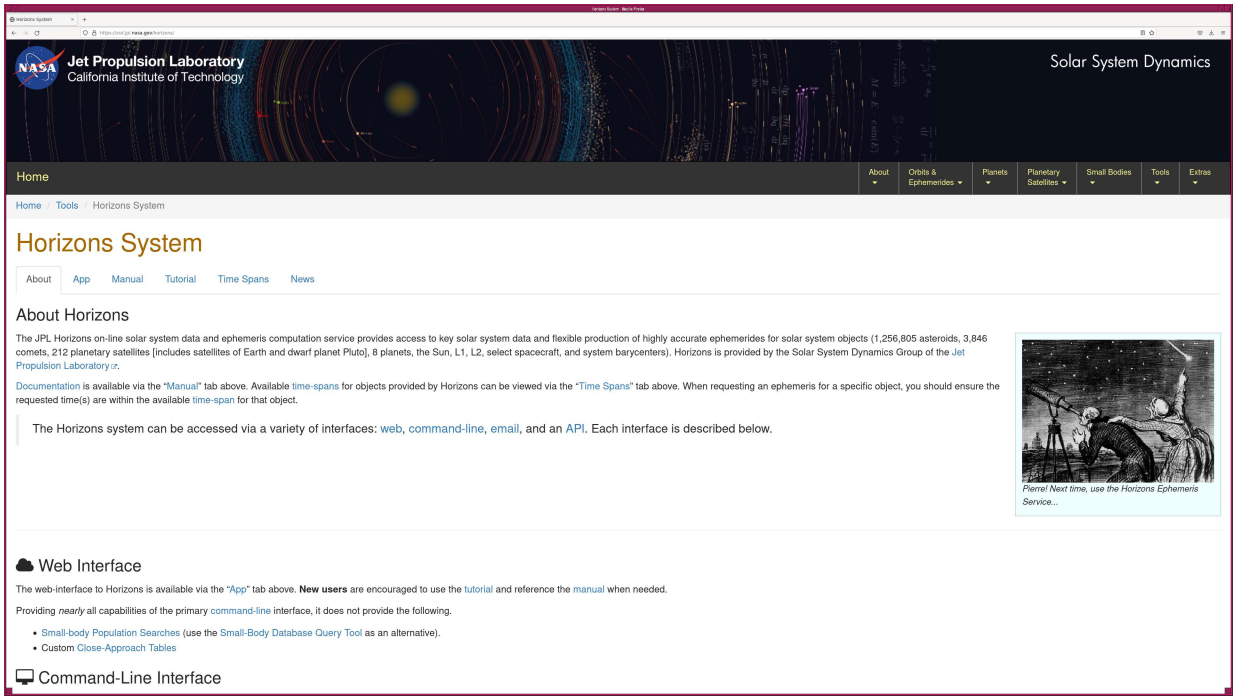

Figure 7: NASA/JPL Horizons System web page.

Try following practice.

Practice 13-03

Use the web interface of the NASA/JPL Horizons System to retrieve RA and Dec of Mars as observed at Mauna Kea from 01/Jan/2023 to 01/Feb/2023 with 1 hour interval.

3.2 Using NASA/JPL Horizons System using Astroquery

Astroquery package can be used to retrieve positions of solar system bodies.

3.2.1 Retrieving positions of target body

Make a Python script to retrieve the ephemeris of the Sun as observed at Lulin Observatory at 04:00:00 (UT) on 21/Mar/2023.

Python Code 3: ai202209_s13_01_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 11:05:47 (CST) daisuke>
#

# importing astropy module
```

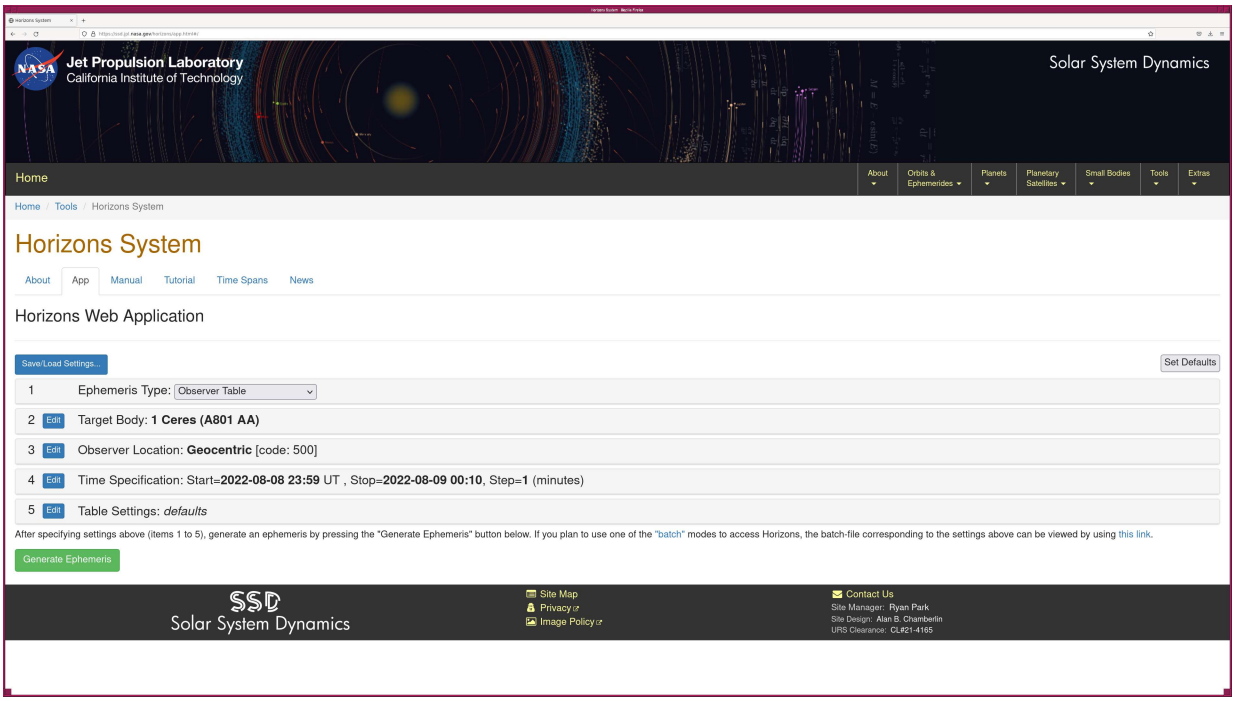

Figure 8: The web interface of NASA/JPL Horizons System web page.

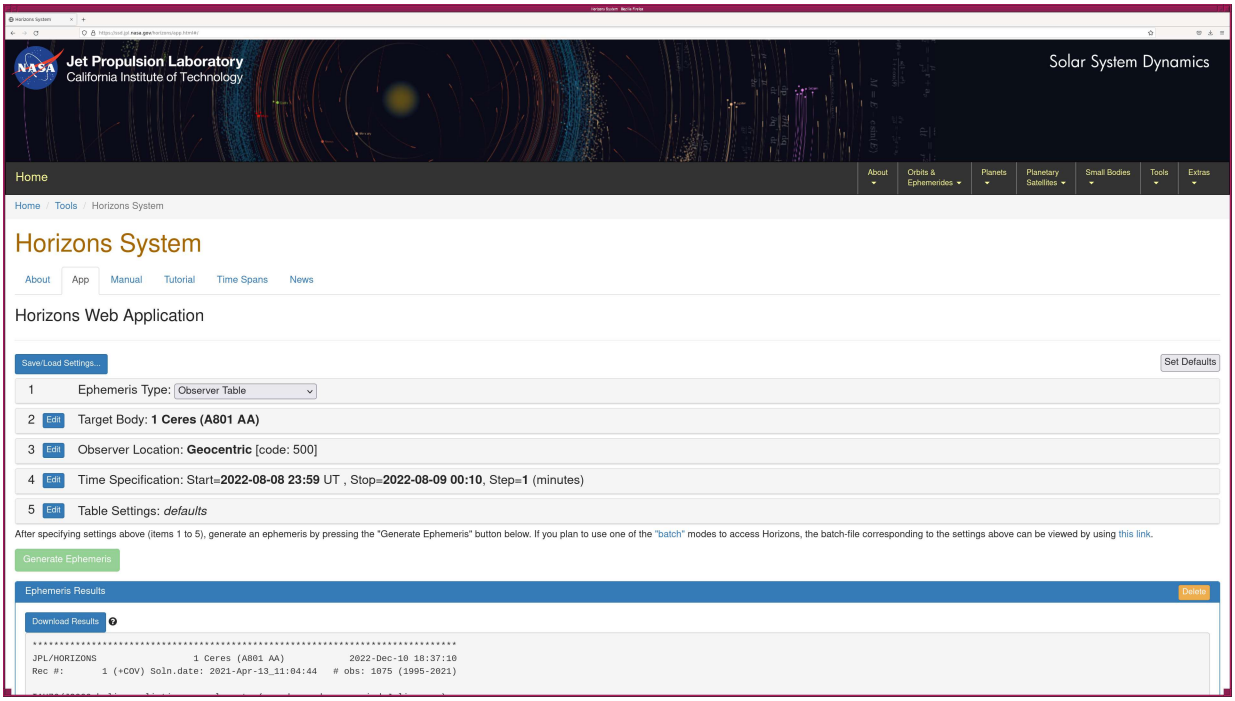

Figure 9: Generated ephemeris of NASA/JPL Horizons System.

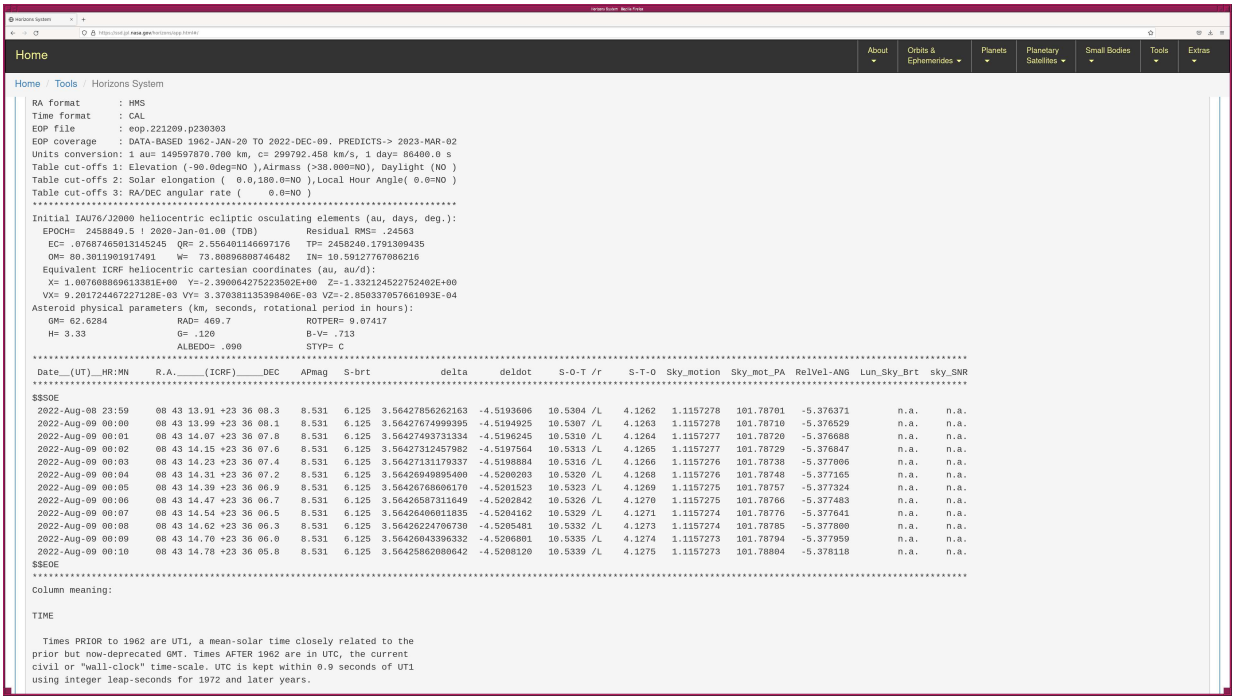

Figure 10: RA and Dec of the target object generated by NASA/JPL Horizons System web page.

```
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# target name
target = 'Sun'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t  = astropy.time.Time ('2023-03-21T04:00:00', scale='utc', format='isot')
jd = t.jd

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, epochs=jd)

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing object type of "ephemerides"
print (type (ephemerides))

# printing ephemerides
print (ephemerides)
```

Execute above script to get the ephemeris.

```
% chmod a+x ai202209_s13_01_00.py
% ./ai202209_s13_01_00.py
JPLHorizons instance "Sun"; location=D35, epochs=[2460024.6666666665], id_type=None
<class 'astropy.table.table.Table'>
targetname       datetime_str          datetime_jd    ...  PABLon   PABLat
   ---               ---                    d         ...   deg      deg
---------- ----------------------- ----------------- ... -------- -------
  Sun (10) 2023-Mar-21 04:00:00.000 2460024.666666667 ... 359.9568 -0.0013
```

The results of the query is returned as a Astropy Table.
Make a Python script to show the column names of the Astropy Table returned by Astroquery package.

Python Code 4: ai202209_s13_01_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 12:43:34 (CST) daisuke>
#

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# target name
target = 'Sun'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t  = astropy.time.Time ('2023-03-21T04:00:00', scale='utc', format='isot')
jd = t.jd

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, epochs=jd)

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing column information of ephemerides
for col in ephemerides.columns:
    print (f"{col}")
```

Execute above script to print names of Astropy Table's columns.

```
% chmod a+x ai202209_s13_01_01.py
% ./ai202209_s13_01_01.py
JPLHorizons instance "Sun"; location=D35, epochs=[2460024.6666666665], id_type=None
targetname
datetime_str
datetime_jd
solar_presence
```

```
flags
RA
DEC
RA_app
DEC_app
RA_rate
DEC_rate
AZ
EL
AZ_rate
EL_rate
sat_X
sat_Y
sat_PANG
siderealtime
airmass
magextinct
V
surfbright
illumination
illum_defect
sat_sep
sat_vis
ang_width
PDObsLon
PDObsLat
PDSunLon
PDSunLat
SubSol_ang
SubSol_dist
NPole_ang
NPole_dist
EclLon
EclLat
r
r_rate
delta
delta_rate
lighttime
vel_sun
vel_obs
elong
elongFlag
alpha
lunar_elong
lunar_illum
sat_alpha
sunTargetPA
velocityPA
OrbPlaneAng
constellation
TDB-UT
ObsEclLon
ObsEclLat
NPole_RA
NPole_DEC
GlxLon
GlxLat
solartime
```

```
earth_lighttime
RA_3sigma
DEC_3sigma
SMAA_3sigma
SMIA_3sigma
Theta_3sigma
Area_3sigma
RSS_3sigma
r_3sigma
r_rate_3sigma
SBand_3sigma
XBand_3sigma
DoppDelay_3sigma
true_anom
hour_angle
alpha_true
PABLon
PABLat
```

Make a Python script to print RA, Dec, Az, El, Ecliptic longitude, Ecliptic latitude, Galactic longitude, Galactic latitude, V-band magnitude, heliocentric distance, and geocentric distance of the Sun as observed at Lulin Observatory on 04:00:00 (UT) on 21/Mar/2023.

Python Code 5: ai202209_s13_01_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 12:42:58 (CST) daisuke>
#

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# target name
target = 'Sun'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t  = astropy.time.Time ('2023-03-21T04:00:00', scale='utc', format='isot')
jd = t.jd

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, epochs=jd)

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'V', 'r', and 'delta'
```

```
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script to obtain the position of the Sun.

```
% chmod a+x ai202209_s13_01_02.py
% ./ai202209_s13_01_02.py
JPLHorizons instance "Sun"; location=D35, epochs=[2460024.6666666665], id_type=None
Sun (10)
  2023-Mar-21 04:00:00.000
    RA           = 359.96087
    DEC          = -0.0184
    AZ           = 177.585547
    EL           = 66.620371
    ObsEclLon    = 0.2727666
    ObsEclLat    = -0.0010784
    GlxLon       = 96.23898
    GlxLat       = -60.18983
    V            = -26.751
    r            = 0.0
    delta        = 0.99591325202664
```

Try following practice.

Practice 13-04

Use Astroquery to retrieve the position of Uranus as observed at Palomar Observatory at 16:00:00 (UT) on 01/Jan/2023.

Make a Python script to retrieve the positions of Jupiter at multiple time epochs.

Python Code 6: ai202209_s13_01_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:24:20 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_hr  = astropy.units.hour

# target name
target = 'Jupiter'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'
```

```
# date/time
t_start = astropy.time.Time ('2023-01-21T12:00:00', scale='utc', format='isot')
t_end   = t_start + 6.0 * u_hr
t_step  = '1h'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'V', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script to get the positions of Jupiter at multiple time epochs.

```
% chmod a+x ai202209_s13_01_03.py
% ./ai202209_s13_01_03.py
JPLHorizons instance "Jupiter"; location=D35, epochs={'start': '2023-01-21T12:00:
00.000', 'stop': '2023-01-21T18:00:00.000', 'step': '1h'}, id_type=None
Traceback (most recent call last):
  File "/amd/ogikubo/root/volume1/nas0/daisuke/backup/daisuke/tex/astro/NCU/Lectu
re/Astroinformatics_202209/session_13/script_13/./ai202209_s13_01_03.py", line 39
, in <module>
    ephemerides = obj.ephemerides ()
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/class_or_instance.p
y", line 25, in f
    return self.fn(obj, *args, **kwds)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/process_asyncs.py",
 line 29, in newmethod
    result = self._parse_result(response, verbose=verbose)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
e 1295, in _parse_result
    data = self._parse_horizons(response.text)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
e 1152, in _parse_horizons
    raise ValueError(('Ambiguous target name; provide '
ValueError: Ambiguous target name; provide unique id:
  ID#      Name                               Designation  IAU/aliases/other
  -------  ---------------------------------- -----------  -------------------
        5  Jupiter Barycenter
      599  Jupiter
```

An error is raised for ambiguous target name. For Jupiter, there are two choices. One is the barycentre of Jovian system and the other is the centre of Jupiter. Choose the ID "599".

Python Code 7: ai202209_s13_01_04.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:25:33 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_hr  = astropy.units.hour

# target name
target = '599'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2022-12-12T10:00:00', scale='utc', format='isot')
t_end   = t_start + 6.0 * u_hr
t_step  = '1h'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'V', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script to get the positions of centre of Jupiter.

```
% chmod a+x ai202209_s13_01_04.py
% ./ai202209_s13_01_04.py
JPLHorizons instance "599"; location=D35, epochs={'start': '2022-12-12T10:00:00.0
00', 'stop': '2022-12-12T16:00:00.000', 'step': '1h'}, id_type=None
Jupiter (599)
  2022-Dec-12 10:00
```

```
    RA            = 359.688
    DEC           = -1.63151
    AZ            = 161.559776
    EL            = 63.835826
    ObsEclLon     = 359.3830338
    ObsEclLat     = -1.372705
    GlxLon        = 94.374299
    GlxLat        = -61.546914
    V             = -2.486
    r             = 4.951456365146
    delta         = 4.70042512389832
  2022-Dec-12 11:00
    RA            = 359.69019
    DEC           = -1.6303
    AZ            = 196.245342
    EL            = 64.110981
    ObsEclLon     = 359.3855236
    ObsEclLat     = -1.3724648
    GlxLon        = 94.379531
    GlxLat        = -61.54675
    V             = -2.486
    r             = 4.951455436906
    delta         = 4.70107849319664
  2022-Dec-12 12:00
    RA            = 359.69239
    DEC           = -1.62909
    AZ            = 223.686744
    EL            = 57.077594
    ObsEclLon     = 359.3880179
    ObsEclLat     = -1.3722232
    GlxLon        = 94.384773
    GlxLat        = -61.546584
    V             = -2.485
    r             = 4.951454508596
    delta         = 4.70173459516902
  2022-Dec-12 13:00
    RA            = 359.69461
    DEC           = -1.62787
    AZ            = 240.468289
    EL            = 46.133686
    ObsEclLon     = 359.3905246
    ObsEclLat     = -1.3719837
    GlxLon        = 94.39004
    GlxLat        = -61.54642
    V             = -2.485
    r             = 4.951453579879
    delta         = 4.70239325310086
  2022-Dec-12 14:00
    RA            = 359.69685
    DEC           = -1.62665
    AZ            = 251.218877
    EL            = 33.532491
    ObsEclLon     = 359.3930513
    ObsEclLat     = -1.3717496
    GlxLon        = 94.395349
    GlxLat        = -61.546263
    V             = -2.485
    r             = 4.951452650453
    delta         = 4.70305412013865
```

```
  2022-Dec-12 15:00
    RA            = 359.69912
    DEC           = -1.62542
    AZ            = 259.039778
    EL            = 20.200544
    ObsEclLon     = 359.3956043
    ObsEclLat     = -1.3715239
    GlxLon        = 94.400713
    GlxLat        = -61.546114
    V             = -2.484
    r             = 4.951451720056
    delta         = 4.70371670301957
  2022-Dec-12 16:00
    RA            = 359.70143
    DEC           = -1.6242
    AZ            = 265.502007
    EL            = 6.5366
    ObsEclLon     = 359.3981887
    ObsEclLat     = -1.3713089
    GlxLon        = 94.406142
    GlxLat        = -61.545976
    V             = -2.484
    r             = 4.951450788477
    delta         = 4.70438039583253
```

Try following practice.

Practice 13-05

Use Astroquery to retrieve the position of Uranus as observed at Palomar Observatory from 00:00:00 (UT) on 01/Jan/2023 to 00:00:00 (UT) on 05/Jan/2023 with 1 hour interval.

Make a Python script to retrieve the positions of the asteroid (4) Vesta.

Python Code 8: ai202209_s13_01_05.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:39:04 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = 'Vesta'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2022-12-12T12:00:00', scale='utc', format='isot')
```

```
t_end   = t_start + 5.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'V', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script to get the positions of (4) Vesta.

```
% chmod a+x ai202209_s13_01_05.py
% ./ai202209_s13_01_05.py
JPLHorizons instance "Vesta"; location=D35, epochs={'start': '2022-12-12T12:00:00
.000', 'stop': '2022-12-17T12:00:00.000', 'step': '1d'}, id_type=smallbody
4 Vesta (A807 FA)
  2022-Dec-12 12:00
    RA           = 340.84386
    DEC          = -15.49116
    AZ           = 229.801018
    EL           = 34.262262
    ObsEclLon    = 336.7869012
    ObsEclLat    = -6.8512642
    GlxLon       = 47.718091
    GlxLat       = -58.102545
    V            = 7.93
    r            = 2.399270152828
    delta        = 2.43060810284826
  2022-Dec-13 12:00
    RA           = 341.16569
    DEC          = -15.33466
    AZ           = 230.467856
    EL           = 33.901797
    ObsEclLon    = 337.1357886
    ObsEclLat    = -6.8239314
    GlxLon       = 48.249563
    GlxLat       = -58.308208
    V            = 7.941
    r            = 2.40023103187
    delta        = 2.4439264936354
  2022-Dec-14 12:00
    RA           = 341.48978
```

```
    DEC           = -15.17713
    AZ            = 231.127233
    EL            = 33.538047
    ObsEclLon     = 337.487306
    ObsEclLat     = -6.796837
    GlxLon        = 48.790507
    GlxLat        = -58.513732
    V             = 7.951
    r             = 2.401190739683
    delta         = 2.45721485277129
  2022-Dec-15 12:00
    RA            = 341.81609
    DEC           = -15.01861
    AZ            = 231.779293
    EL            = 33.17111
    ObsEclLon     = 337.8414098
    ObsEclLat     = -6.76998
    GlxLon        = 49.34106
    GlxLat        = -58.719007
    V             = 7.962
    r             = 2.402149257414
    delta         = 2.47047168924644
  2022-Dec-16 12:00
    RA            = 342.14457
    DEC           = -14.85909
    AZ            = 232.424182
    EL            = 32.80108
    ObsEclLon     = 338.1980586
    ObsEclLat     = -6.7433597
    GlxLon        = 49.901359
    GlxLat        = -58.923918
    V             = 7.972
    r             = 2.403106566277
    delta         = 2.4836954818475
  2022-Dec-17 12:00
    RA            = 342.47518
    DEC           = -14.6986
    AZ            = 233.062042
    EL            = 32.428048
    ObsEclLon     = 338.5572141
    ObsEclLat     = -6.7169748
    GlxLon        = 50.471544
    GlxLat        = -59.128352
    V             = 7.982
    r             = 2.404062647555
    delta         = 2.49688467329481
```

The positions of the asteroid (4) Vesta can be obtained by specifying the string "4" as "id" and the string "smallbody" as "id_type".

Python Code 9: ai202209_s13_01_06.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:38:52 (CST) daisuke>
#

# importing astropy module
```

```
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = '4'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2022-12-12T12:00:00', scale='utc', format='isot')
t_end   = t_start + 5.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'V', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_06.py
% ./ai202209_s13_01_06.py
JPLHorizons instance "4"; location=D35, epochs={'start': '2022-12-12T12:00:00.000
', 'stop': '2022-12-17T12:00:00.000', 'step': '1d'}, id_type=smallbody
4 Vesta (A807 FA)
  2022-Dec-12 12:00
    RA           = 340.84386
    DEC          = -15.49116
    AZ           = 229.801018
    EL           = 34.262262
    ObsEclLon    = 336.7869012
    ObsEclLat    = -6.8512642
    GlxLon       = 47.718091
    GlxLat       = -58.102545
```

```
    V            = 7.93
    r            = 2.399270152828
    delta        = 2.43060810284826
  2022-Dec-13 12:00
    RA           = 341.16569
    DEC          = -15.33466
    AZ           = 230.467856
    EL           = 33.901797
    ObsEclLon    = 337.1357886
    ObsEclLat    = -6.8239314
    GlxLon       = 48.249563
    GlxLat       = -58.308208
    V            = 7.941
    r            = 2.40023103187
    delta        = 2.4439264936354
  2022-Dec-14 12:00
    RA           = 341.48978
    DEC          = -15.17713
    AZ           = 231.127233
    EL           = 33.538047
    ObsEclLon    = 337.487306
    ObsEclLat    = -6.796837
    GlxLon       = 48.790507
    GlxLat       = -58.513732
    V            = 7.951
    r            = 2.401190739683
    delta        = 2.45721485277129
  2022-Dec-15 12:00
    RA           = 341.81609
    DEC          = -15.01861
    AZ           = 231.779293
    EL           = 33.17111
    ObsEclLon    = 337.8414098
    ObsEclLat    = -6.76998
    GlxLon       = 49.34106
    GlxLat       = -58.719007
    V            = 7.962
    r            = 2.402149257414
    delta        = 2.47047168924644
  2022-Dec-16 12:00
    RA           = 342.14457
    DEC          = -14.85909
    AZ           = 232.424182
    EL           = 32.80108
    ObsEclLon    = 338.1980586
    ObsEclLat    = -6.7433597
    GlxLon       = 49.901359
    GlxLat       = -58.923918
    V            = 7.972
    r            = 2.403106566277
    delta        = 2.4836954818475
  2022-Dec-17 12:00
    RA           = 342.47518
    DEC          = -14.6986
    AZ           = 233.062042
    EL           = 32.428048
    ObsEclLon    = 338.5572141
    ObsEclLat    = -6.7169748
    GlxLon       = 50.471544
```

```
    GlxLat        = -59.128352
    V             = 7.982
    r             = 2.404062647555
    delta         = 2.49688467329481
```

The positions of the asteroid (4) Vesta can be obtained by specifying the string "4;" as "id".

Python Code 10: ai202209_s13_01_07.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 16:27:41 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = '4;'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2022-12-12T12:00:00', scale='utc', format='isot')
t_end   = t_start + 5.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=None, id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'V', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_07.py
% ./ai202209_s13_01_07.py
JPLHorizons instance "4;"; location=D35, epochs={'start': '2022-12-12T12:00:00.00
0', 'stop': '2022-12-17T12:00:00.000', 'step': '1d'}, id_type=None
4 Vesta (A807 FA)
  2022-Dec-12 12:00
    RA           = 340.84386
    DEC          = -15.49116
    AZ           = 229.801018
    EL           = 34.262262
    ObsEclLon    = 336.7869012
    ObsEclLat    = -6.8512642
    GlxLon       = 47.718091
    GlxLat       = -58.102545
    V            = 7.93
    r            = 2.399270152828
    delta        = 2.43060810284826
  2022-Dec-13 12:00
    RA           = 341.16569
    DEC          = -15.33466
    AZ           = 230.467856
    EL           = 33.901797
    ObsEclLon    = 337.1357886
    ObsEclLat    = -6.8239314
    GlxLon       = 48.249563
    GlxLat       = -58.308208
    V            = 7.941
    r            = 2.40023103187
    delta        = 2.4439264936354
  2022-Dec-14 12:00
    RA           = 341.48978
    DEC          = -15.17713
    AZ           = 231.127233
    EL           = 33.538047
    ObsEclLon    = 337.487306
    ObsEclLat    = -6.796837
    GlxLon       = 48.790507
    GlxLat       = -58.513732
    V            = 7.951
    r            = 2.401190739683
    delta        = 2.45721485277129
  2022-Dec-15 12:00
    RA           = 341.81609
    DEC          = -15.01861
    AZ           = 231.779293
    EL           = 33.17111
    ObsEclLon    = 337.8414098
    ObsEclLat    = -6.76998
    GlxLon       = 49.34106
    GlxLat       = -58.719007
    V            = 7.962
    r            = 2.402149257414
    delta        = 2.47047168924644
  2022-Dec-16 12:00
    RA           = 342.14457
    DEC          = -14.85909
    AZ           = 232.424182
    EL           = 32.80108
```

```
    ObsEclLon    = 338.1980586
    ObsEclLat    = -6.7433597
    GlxLon       = 49.901359
    GlxLat       = -58.923918
    V            = 7.972
    r            = 2.403106566277
    delta        = 2.4836954818475
  2022-Dec-17 12:00
    RA           = 342.47518
    DEC          = -14.6986
    AZ           = 233.062042
    EL           = 32.428048
    ObsEclLon    = 338.5572141
    ObsEclLat    = -6.7169748
    GlxLon       = 50.471544
    GlxLat       = -59.128352
    V            = 7.982
    r            = 2.404062647555
    delta        = 2.49688467329481
```

Try following practice.

Practice 13-06

Use Astroquery to retrieve the position of your favourite asteroid as observed at Greenwich Observatory from 00:00:00 (UT) on 01/Mar/2023 to 00:00:00 (UT) on 05/Mar/2023 with 1 hour interval.

Make a Python script to retrieve the positions of the comet 2P/Encke.

Python Code 11: ai202209_s13_01_08.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:56:04 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = '2P'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2023-01-01T12:00:00', scale='utc', format='isot')
t_end   = t_start + 3.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
```

```
                                        location=obs_site, \
                                        epochs={'start': t_start.isot, \
                                                'stop': t_end.isot, \
                                                'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_08.py
% ./ai202209_s13_01_08.py
JPLHorizons instance "2P"; location=D35, epochs={'start': '2023-01-01T12:00:00.00
0', 'stop': '2023-01-04T12:00:00.000', 'step': '1d'}, id_type=smallbody
Traceback (most recent call last):
  File "/amd/ogikubo/root/volume1/nas0/daisuke/backup/daisuke/tex/astro/NCU/Lectu
re/Astroinformatics_202209/session_13/script_13/./ai202209_s13_01_08.py", line 39
, in <module>
    ephemerides = obj.ephemerides ()
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/class_or_instance.p
y", line 25, in f
    return self.fn(obj, *args, **kwds)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/process_asyncs.py",
 line 29, in newmethod
    result = self._parse_result(response, verbose=verbose)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
e 1295, in _parse_result
    data = self._parse_horizons(response.text)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
e 1152, in _parse_horizons
    raise ValueError(('Ambiguous target name; provide '
ValueError: Ambiguous target name; provide unique id:
    Record #  Epoch-yr  >MATCH DESIG<  Primary Desig  Name
    --------  --------  -------------  -------------  -------------------------
    90000031    1786    2P             2P              Encke
    90000032    1796    2P             2P              Encke
    90000033    1805    2P             2P              Encke
    90000034    1819    2P             2P              Encke
    90000035    1822    2P             2P              Encke
    90000036    1825    2P             2P              Encke
    90000037    1828    2P             2P              Encke
    90000038    1832    2P             2P              Encke
    90000039    1835    2P             2P              Encke
    90000040    1838    2P             2P              Encke
    90000041    1842    2P             2P              Encke
    90000042    1845    2P             2P              Encke
```

```
    90000043    1848    2P              2P              Encke
    90000044    1852    2P              2P              Encke
    90000045    1855    2P              2P              Encke
    90000046    1858    2P              2P              Encke
    90000047    1862    2P              2P              Encke
    90000048    1865    2P              2P              Encke
    90000049    1868    2P              2P              Encke
    90000050    1872    2P              2P              Encke
    90000051    1875    2P              2P              Encke
    90000052    1878    2P              2P              Encke
    90000053    1881    2P              2P              Encke
    90000054    1885    2P              2P              Encke
    90000055    1888    2P              2P              Encke
    90000056    1891    2P              2P              Encke
    90000057    1895    2P              2P              Encke
    90000058    1898    2P              2P              Encke
    90000059    1901    2P              2P              Encke
    90000060    1904    2P              2P              Encke
    90000061    1908    2P              2P              Encke
    90000062    1911    2P              2P              Encke
    90000063    1914    2P              2P              Encke
    90000064    1918    2P              2P              Encke
    90000065    1921    2P              2P              Encke
    90000066    1924    2P              2P              Encke
    90000067    1928    2P              2P              Encke
    90000068    1931    2P              2P              Encke
    90000069    1934    2P              2P              Encke
    90000070    1937    2P              2P              Encke
    90000071    1941    2P              2P              Encke
    90000072    1947    2P              2P              Encke
    90000073    1951    2P              2P              Encke
    90000074    1954    2P              2P              Encke
    90000075    1957    2P              2P              Encke
    90000076    1961    2P              2P              Encke
    90000077    1964    2P              2P              Encke
    90000078    1967    2P              2P              Encke
    90000079    1971    2P              2P              Encke
    90000080    1974    2P              2P              Encke
    90000081    1977    2P              2P              Encke
    90000082    1980    2P              2P              Encke
    90000083    1984    2P              2P              Encke
    90000084    1987    2P              2P              Encke
    90000085    1990    2P              2P              Encke
    90000086    1994    2P              2P              Encke
    90000087    1995    2P              2P              Encke
    90000088    1998    2P              2P              Encke
    90000089    2004    2P              2P              Encke
    90000090    2016    2P              2P              Encke
```

For comets, the year of an apparition has to be specified. Choose "90000090".

Python Code 12: ai202209_s13_01_09.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 15:56:31 (CST) daisuke>
#
```

```
# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = '90000090'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2023-01-01T12:00:00', scale='utc', format='isot')
t_end   = t_start + 3.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_09.py
% ./ai202209_s13_01_09.py
JPLHorizons instance "90000090"; location=D35, epochs={'start': '2023-01-01T12:00
:00.000', 'stop': '2023-01-04T12:00:00.000', 'step': '1d'}, id_type=smallbody
2P/Encke
  2023-Jan-01 12:00
    RA           = 342.09185
    DEC          = -3.15367
    AZ           = 253.706783
    EL           = 26.461081
    ObsEclLon    = 342.5959687
    ObsEclLat    = 4.10866
    GlxLon       = 66.75212
```

```
    GlxLat        = -52.057948
    Tmag          = 20.592
    r             = 3.340524256967
    delta         = 3.689442268364
  2023-Jan-02 12:00
    RA            = 342.25044
    DEC           = -3.09436
    AZ            = 254.197404
    EL            = 25.759842
    ObsEclLon     = 342.7654012
    ObsEclLat     = 4.1034163
    GlxLon        = 66.99694
    GlxLat        = -52.135085
    Tmag          = 20.594
    r             = 3.335294331551
    delta         = 3.69865234366708
  2023-Jan-03 12:00
    RA            = 342.41189
    DEC           = -3.03365
    AZ            = 254.681616
    EL            = 25.059786
    ObsEclLon     = 342.9380257
    ObsEclLat     = 4.098322
    GlxLon        = 67.247428
    GlxLat        = -52.21289
    Tmag          = 20.597
    r             = 3.330042838229
    delta         = 3.70772081465256
  2023-Jan-04 12:00
    RA            = 342.57615
    DEC           = -2.97155
    AZ            = 255.159718
    EL            = 24.360951
    ObsEclLon     = 343.1137949
    ObsEclLat     = 4.0933775
    GlxLon        = 67.503563
    GlxLat        = -52.291308
    Tmag          = 20.599
    r             = 3.324769716513
    delta         = 3.71664528151287
```

For retrieving positions based on the latest orbital elements, use "`DES=2P; CAP;`" as "`id`".

Python Code 13: ai202209_s13_01_10.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 16:18:28 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
```

```
u_day = astropy.units.day

# target name
target = 'DES=2P; CAP;'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'

# date/time
t_start = astropy.time.Time ('2023-01-01T12:00:00', scale='utc', format='isot')
t_end   = t_start + 3.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_10.py
% ./ai202209_s13_01_10.py
JPLHorizons instance "DES=2P; CAP;"; location=D35, epochs={'start': '2023-01-01T1
2:00:00.000', 'stop': '2023-01-04T12:00:00.000', 'step': '1d'}, id_type=smallbody
2P/Encke
  2023-Jan-01 12:00
    RA           = 342.09185
    DEC          = -3.15367
    AZ           = 253.706783
    EL           = 26.461081
    ObsEclLon    = 342.5959687
    ObsEclLat    = 4.10866
    GlxLon       = 66.75212
    GlxLat       = -52.057948
    Tmag         = 20.592
    r            = 3.340524256967
    delta        = 3.689442268364
  2023-Jan-02 12:00
    RA           = 342.25044
    DEC          = -3.09436
    AZ           = 254.197404
```

```
    EL           = 25.759842
    ObsEclLon    = 342.7654012
    ObsEclLat    = 4.1034163
    GlxLon       = 66.99694
    GlxLat       = -52.135085
    Tmag         = 20.594
    r            = 3.335294331551
    delta        = 3.69865234366708
  2023-Jan-03 12:00
    RA           = 342.41189
    DEC          = -3.03365
    AZ           = 254.681616
    EL           = 25.059786
    ObsEclLon    = 342.9380257
    ObsEclLat    = 4.098322
    GlxLon       = 67.247428
    GlxLat       = -52.21289
    Tmag         = 20.597
    r            = 3.330042838229
    delta        = 3.70772081465256
  2023-Jan-04 12:00
    RA           = 342.57615
    DEC          = -2.97155
    AZ           = 255.159718
    EL           = 24.360951
    ObsEclLon    = 343.1137949
    ObsEclLat    = 4.0933775
    GlxLon       = 67.503563
    GlxLat       = -52.291308
    Tmag         = 20.599
    r            = 3.324769716513
    delta        = 3.71664528151287
```

The positions of comet 2P/Encke based on the latest orbital elements can also be retrieved by giving "`NAME=Encke; CAP;`" as "`id`".

Python Code 14: ai202209_s13_01_11.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 16:31:53 (CST) daisuke>
#

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

# units
u_day = astropy.units.day

# target name
target = 'NAME=Encke; CAP;'

# observatory (D35 ==> Lulin Observatory)
obs_site = 'D35'
```

```
# date/time
t_start = astropy.time.Time ('2023-01-01T12:00:00', scale='utc', format='isot')
t_end   = t_start + 3.0 * u_day
t_step  = '1d'

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type='smallbody', id=target, \
                                       location=obs_site, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': t_step})

# printing a query object
print (obj)

# getting ephemerides
ephemerides = obj.ephemerides ()

# printing 'targetname', 'datetime_str', 'RA', 'DEC', 'AZ', 'EL',
# 'ObsEclLon', 'ObsEclLat', 'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta'
columns = ['RA', 'DEC', 'AZ', 'EL', 'ObsEclLon', 'ObsEclLat', \
           'GlxLon', 'GlxLat', 'Tmag', 'r', 'delta']
print (f"{ephemerides[0]['targetname']}")
for eph in ephemerides:
    print (f"  {eph['datetime_str']}")
    for col in columns:
        print (f"    {col:12s} = {eph[col]}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_11.py
% ./ai202209_s13_01_11.py
JPLHorizons instance "NAME=Encke; CAP;"; location=D35, epochs={'start': '2023-01-
01T12:00:00.000', 'stop': '2023-01-04T12:00:00.000', 'step': '1d'}, id_type=small
body
2P/Encke
  2023-Jan-01 12:00
    RA           = 342.09185
    DEC          = -3.15367
    AZ           = 253.706783
    EL           = 26.461081
    ObsEclLon    = 342.5959687
    ObsEclLat    = 4.10866
    GlxLon       = 66.75212
    GlxLat       = -52.057948
    Tmag         = 20.592
    r            = 3.340524256967
    delta        = 3.689442268364
  2023-Jan-02 12:00
    RA           = 342.25044
    DEC          = -3.09436
    AZ           = 254.197404
    EL           = 25.759842
    ObsEclLon    = 342.7654012
    ObsEclLat    = 4.1034163
    GlxLon       = 66.99694
    GlxLat       = -52.135085
    Tmag         = 20.594
```

```
    r             = 3.335294331551
    delta         = 3.69865234366708
  2023-Jan-03 12:00
    RA            = 342.41189
    DEC           = -3.03365
    AZ            = 254.681616
    EL            = 25.059786
    ObsEclLon     = 342.9380257
    ObsEclLat     = 4.098322
    GlxLon        = 67.247428
    GlxLat        = -52.21289
    Tmag          = 20.597
    r             = 3.330042838229
    delta         = 3.70772081465256
  2023-Jan-04 12:00
    RA            = 342.57615
    DEC           = -2.97155
    AZ            = 255.159718
    EL            = 24.360951
    ObsEclLon     = 343.1137949
    ObsEclLat     = 4.0933775
    GlxLon        = 67.503563
    GlxLat        = -52.291308
    Tmag          = 20.599
    r             = 3.324769716513
    delta         = 3.71664528151287
```

Try following practice.

Practice 13-07

Use Astroquery to retrieve the position of your favourite comet as observed at Paranal Observatory from 00:00:00 (UT) on 01/Apr/2023 to 00:00:00 (UT) on 05/Apr/2023 with 1 hour interval.

Make a Python script which can be used to retrieve positions of a solar system object for a given name or designation as observed at specified location for the specified period of time. Use "`argparse`".

Python Code 15: ai202209_s13_01_12.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/09 17:14:09 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
```

```
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='D35', \
                     help='location of observer (default: D35)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# sending a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})
```

```
# getting ephemerides
ephemerides = obj.ephemerides ()

# printing ephemerides
print (f"{ephemerides[0]['targetname']}")
print (f"-" * 60)
print (f" {'Date/Time':17s}  {'RA':9s} {'Dec':9s}  {'Az':9s} {'El':9s}")
print (f"=" * 60)
for eph in ephemerides:
    print (f" {eph['datetime_str']:17s}", \
           f" {eph['RA']:9.5f} {eph['DEC']:+9.5f}", \
           f" {eph['AZ']:9.5f} {eph['EL']:+9.5f}")
print (f"-" * 60)
```

Execute above script.

```
% chmod a+x ai202209_s13_01_12.py
% ./ai202209_s13_01_12.py -h
usage: ai202209_s13_01_12.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location of observer (default: D35)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_12.py -i Mars -l D35 \
? -s 2022-12-12T12:00:00 -e 2022-12-13T12:00:00 -d 1h
Traceback (most recent call last):
  File "/amd/ogikubo/root/volume1/nas0/daisuke/backup/daisuke/tex/astro/NCU/Lectu
re/Astroinformatics_202209/session_13/script_13/./ai202209_s13_01_12.py", line 83
, in <module>
    ephemerides = obj.ephemerides ()
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/class_or_instance.p
y", line 25, in f
    return self.fn(obj, *args, **kwds)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/utils/process_asyncs.py",
 line 29, in newmethod
    result = self._parse_result(response, verbose=verbose)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
e 1295, in _parse_result
    data = self._parse_horizons(response.text)
  File "/usr/pkg/lib/python3.9/site-packages/astroquery/jplhorizons/core.py", lin
```

```
e 1152, in _parse_horizons
    raise ValueError(('Ambiguous target name; provide '
ValueError: Ambiguous target name; provide unique id:
  ID#      Name                                Designation  IAU/aliases/other
  -------  ----------------------------------- -----------  -------------------
        4  Mars Barycenter
      499  Mars
       -3  Mars Orbiter Mission (MOM) (spacecr              Mangalyaan
      -41  Mars Express (spacecraft)                        MEX
      -53  Mars Odyssey (spacecraft)
     -530  Mars Pathfinder (spacecraft)                     MPF
      -74  Mars Reconnaissance Orbiter (spacec              MRO
      -76  Mars Science Laboratory (spacecraft              Curiosity MSL
     -143  ExoMars16 TGO (spacecraft)
     -168  Mars2020 (spacecraft)                           Perserverance Ingenu

% ./ai202209_s13_01_12.py -i 499 -l D35 \
? -s 2022-12-12T12:00:00 -e 2022-12-13T12:00:00 -d 1h
Mars (499)
------------------------------------------------------------
 Date/Time           RA         Dec         Az         El
============================================================
 2022-Dec-12 12:00   72.50866 +24.92262   77.24942 +43.90244
 2022-Dec-12 13:00   72.49085 +24.92241   79.92448 +57.44453
 2022-Dec-12 14:00   72.47288 +24.92211   81.24135 +71.07860
 2022-Dec-12 15:00   72.45481 +24.92169   72.67543 +84.64171
 2022-Dec-12 16:00   72.43672 +24.92115  281.77361 +81.28569
 2022-Dec-12 17:00   72.41869 +24.92049  278.85012 +67.65843
 2022-Dec-12 18:00   72.40080 +24.91973  280.65946 +54.04005
 2022-Dec-12 19:00   72.38312 +24.91887  283.53483 +40.53186
 2022-Dec-12 20:00   72.36569 +24.91794  287.09985 +27.20806
 2022-Dec-12 21:00   72.34856 +24.91697  291.39788 +14.16506
 2022-Dec-12 22:00   72.33175 +24.91599  296.64389  +1.54399
 2022-Dec-12 23:00   72.31524 +24.91502  303.20701 -10.43602
 2022-Dec-13 00:00   72.29903 +24.91411  311.63822 -21.42245
 2022-Dec-13 01:00   72.28305 +24.91326  322.65439 -30.84569
 2022-Dec-13 02:00   72.26726 +24.91251  336.87541 -37.84086
 2022-Dec-13 03:00   72.25159 +24.91186  354.02266 -41.34209
 2022-Dec-13 04:00   72.23595 +24.91133   12.09918 -40.59776
 2022-Dec-13 05:00   72.22027 +24.91090   28.37728 -35.78302
 2022-Dec-13 06:00   72.20448 +24.91058   41.46312 -27.83999
 2022-Dec-13 07:00   72.18849 +24.91035   51.50869 -17.79943
 2022-Dec-13 08:00   72.17227 +24.91018   59.22623  -6.42128
 2022-Dec-13 09:00   72.15578 +24.91004   65.28659  +5.81000
 2022-Dec-13 10:00   72.13899 +24.90991   70.17566 +18.59563
 2022-Dec-13 11:00   72.12192 +24.90976   74.20770 +31.74814
 2022-Dec-13 12:00   72.10458 +24.90955   77.54625 +45.14469
------------------------------------------------------------
% ./ai202209_s13_01_12.py -t smallbody -i Pallas -l D35 \
? -s 2022-12-12T12:00:00 -e 2022-12-13T12:00:00 -d 1h
2 Pallas (A802 FA)
------------------------------------------------------------
 Date/Time           RA         Dec         Az         El
============================================================
 2022-Dec-12 12:00  107.43683 -31.38594  120.64772  -8.49993
 2022-Dec-12 13:00  107.43124 -31.39081  126.30769  +3.02183
 2022-Dec-12 14:00  107.42553 -31.39566  133.63522 +13.61344
 2022-Dec-12 15:00  107.41971 -31.40047  143.06876 +22.80698
 2022-Dec-12 16:00  107.41380 -31.40522  154.98916 +29.93718
```

```
 2022-Dec-12 17:00  107.40783 -31.40991  169.28581 +34.19412
 2022-Dec-12 18:00  107.40183 -31.41454  184.85101 +34.90091
 2022-Dec-12 19:00  107.39583 -31.41910  199.81520 +31.92697
 2022-Dec-12 20:00  107.38985 -31.42360  212.68954 +25.78588
 2022-Dec-12 21:00  107.38393 -31.42805  223.02809 +17.28412
 2022-Dec-12 22:00  107.37809 -31.43245  231.09160  +7.15762
 2022-Dec-12 23:00  107.37234 -31.43681  237.32891  -4.05376
 2022-Dec-13 00:00  107.36670 -31.44115  242.11130 -15.98428
 2022-Dec-13 01:00  107.36116 -31.44549  245.63875 -28.38610
 2022-Dec-13 02:00  107.35573 -31.44982  247.86044 -41.07835
 2022-Dec-13 03:00  107.35038 -31.45418  248.21788 -53.89704
 2022-Dec-13 04:00  107.34509 -31.45856  244.49422 -66.58905
 2022-Dec-13 05:00  107.33985 -31.46298  225.16041 -78.16200
 2022-Dec-13 06:00  107.33461 -31.46744  151.19382 -80.72945
 2022-Dec-13 07:00  107.32935 -31.47193  118.57234 -70.34654
 2022-Dec-13 08:00  107.32404 -31.47647  112.32566 -57.80645
 2022-Dec-13 09:00  107.31866 -31.48103  111.82463 -44.99292
 2022-Dec-13 10:00  107.31316 -31.48562  113.57698 -32.24865
 2022-Dec-13 11:00  107.30755 -31.49021  116.72953 -19.74742
 2022-Dec-13 12:00  107.30182 -31.49480  121.12051  -7.65930
-------------------------------------------------------------
% ./ai202209_s13_01_12.py -t smallbody -i "NAME=Halley; CAP;" -l D35 \
? -s 2022-12-12T12:00:00 -e 2022-12-13T12:00:00 -d 1h
1P/Halley
-------------------------------------------------------------
 Date/Time          RA         Dec         Az         El
=============================================================
 2022-Dec-12 12:00  125.79649  +2.07812   82.43532 -12.04430
 2022-Dec-12 13:00  125.79560  +2.07809   88.55335  +1.70480
 2022-Dec-12 14:00  125.79470  +2.07805   94.63111 +15.49084
 2022-Dec-12 15:00  125.79380  +2.07802  101.44625 +29.14826
 2022-Dec-12 16:00  125.79289  +2.07798  110.18916 +42.42582
 2022-Dec-12 17:00  125.79198  +2.07795  123.28579 +54.77793
 2022-Dec-12 18:00  125.79107  +2.07792  146.07080 +64.71353
 2022-Dec-12 19:00  125.79015  +2.07788  183.38385 +68.50307
 2022-Dec-12 20:00  125.78923  +2.07785  218.68386 +63.36554
 2022-Dec-12 21:00  125.78832  +2.07781  239.34883 +52.84750
 2022-Dec-12 22:00  125.78740  +2.07778  251.46256 +40.27920
 2022-Dec-12 23:00  125.78649  +2.07775  259.76922 +26.91353
 2022-Dec-13 00:00  125.78559  +2.07772  266.39833 +13.22135
 2022-Dec-13 01:00  125.78468  +2.07768  272.43349  -0.56932
 2022-Dec-13 02:00  125.78378  +2.07765  278.62723 -14.29714
 2022-Dec-13 03:00  125.78288  +2.07762  285.77323 -27.78058
 2022-Dec-13 04:00  125.78199  +2.07759  295.08673 -40.71736
 2022-Dec-13 05:00  125.78110  +2.07756  308.87002 -52.45845
 2022-Dec-13 06:00  125.78021  +2.07753  331.27390 -61.42643
 2022-Dec-13 07:00  125.77932  +2.07750    4.04662 -64.46997
 2022-Dec-13 08:00  125.77843  +2.07747   34.93841 -59.73729
 2022-Dec-13 09:00  125.77754  +2.07744   54.91206 -49.90202
 2022-Dec-13 10:00  125.77665  +2.07741   67.36072 -37.79149
 2022-Dec-13 11:00  125.77575  +2.07738   76.02122 -24.68867
 2022-Dec-13 12:00  125.77484  +2.07735   82.86139 -11.12834
-------------------------------------------------------------
```

Try following practice.

Practice 13-08

Use Astroquery to retrieve the position of trans-Neptunian object Sedna as observed at Kitt Peak National Observatory from 00:00:00 (UT) on 01/May/2023 to 00:00:00 (UT) on 05/May/2023 with 1 hour interval.

3.2.2 Retrieving orbital elements of target body

Make a Python script to retrieve the orbital elements of a given target body.

Python Code 16: ai202209_s13_01_13.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/10 02:21:14 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='Sun', \
                     help='location of observer (default: Sun)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
```

```
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting ephemerides
elements = obj.elements ()

print (elements)
```

Execute above script.

```
% chmod a+x ai202209_s13_01_13.py
% ./ai202209_s13_01_13.py -h
usage: ai202209_s13_01_13.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location of observer (default: Sun)
  -s START, --start START
```

```
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_13.py -t smallbody -i Flora \
? -s 2010-01-01T12:00:00 -e 2023-01-01T12:00:00 -d 1y
    targetname    datetime_jd ...          Q                 P
       ---             d      ...          AU                d
----------------- ----------- ... ----------------- -----------------
8 Flora (A847 UA)   2455198.0 ... 2.546035147861908 1192.583896100836
8 Flora (A847 UA)   2455563.0 ...   2.5465219181043 1193.131226344891
8 Flora (A847 UA)   2455928.0 ... 2.545629933388549  1192.91239778153
8 Flora (A847 UA)   2456294.0 ... 2.545733972680392 1193.304592373935
8 Flora (A847 UA)   2456659.0 ... 2.545000814398333 1192.962377630618
8 Flora (A847 UA)   2457024.0 ... 2.546344857368881 1193.511393635097
8 Flora (A847 UA)   2457389.0 ... 2.546505875563392 1192.734983658999
8 Flora (A847 UA)   2457755.0 ... 2.546854732883702 1192.883528114909
8 Flora (A847 UA)   2458120.0 ... 2.546737110974819 1192.859529931731
8 Flora (A847 UA)   2458485.0 ... 2.546341169996965 1192.916059737705
8 Flora (A847 UA)   2458850.0 ... 2.545561016804071  1193.57704359297
8 Flora (A847 UA)   2459216.0 ... 2.544508125847151 1193.017600840437
8 Flora (A847 UA)   2459581.0 ... 2.545337687299887 1193.288679088108
8 Flora (A847 UA)   2459946.0 ... 2.545300304501692 1192.799068264417
```

Show all the column names of Astropy Table.

Python Code 17: ai202209_s13_01_14.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/10 02:22:50 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
```

```
                      default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                      help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='Sun', \
                      help='location of observer (default: Sun)')
parser.add_argument ('-s', '--start', type=str, \
                      default='2000-01-01T12:00:00', \
                      help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                      default='2000-01-02T12:00:00', \
                      help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                      default='1h', \
                      help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting ephemerides
elements = obj.elements ()

print (elements.columns)
```

Execute above script.

```
% chmod a+x ai202209_s13_01_14.py
```

```
% ./ai202209_s13_01_14.py -h
usage: ai202209_s13_01_14.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location of observer (default: Sun)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_14.py -t smallbody -i Flora \
? -s 2010-01-01T12:00:00 -e 2023-01-01T12:00:00 -d 1y
<TableColumns names=('targetname','datetime_jd','datetime_str','H','G','e','q','i
ncl','Omega','w','Tp_jd','n','M','nu','a','Q','P')>
```

Make a Python script to retrieve the orbital elements of a given target body, and print semimajor axis, eccentricity, and inclination.

Python Code 18: ai202209_s13_01_15.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/10 02:28:44 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
```

```
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='Sun', \
                     help='location of observer (default: Sun)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting orbital elements
elements = obj.elements ()
```

```
# printing orbital elements
print (f"{elements[0]['targetname']}")
for ele in elements:
    print (f"{ele['datetime_str']:17s}", \
           f"{ele['a']:14.8f} {ele['e']:10.8f} {ele['incl']:12.8f}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_15.py
% ./ai202209_s13_01_15.py -h
usage: ai202209_s13_01_15.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location of observer (default: Sun)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_15.py -t smallbody -i Flora \
? -s 2010-01-01T12:00:00 -e 2023-01-01T12:00:00 -d 1y
8 Flora (A847 UA)
A.D. 2010-Jan-01 12:00:00.0000     2.20086638 0.15683313   5.88912467
A.D. 2011-Jan-01 12:00:00.0000     2.20153971 0.15670042   5.88758347
A.D. 2012-Jan-01 12:00:00.0000     2.20127052 0.15643666   5.88762234
A.D. 2013-Jan-01 12:00:00.0000     2.20175297 0.15623052   5.88768866
A.D. 2014-Jan-01 12:00:00.0000     2.20133200 0.15611857   5.88808145
A.D. 2015-Jan-01 12:00:00.0000     2.20200734 0.15637437   5.88766212
A.D. 2016-Jan-01 12:00:00.0000     2.20105226 0.15694930   5.88683614
A.D. 2017-Jan-01 12:00:00.0000     2.20123500 0.15701174   5.88666882
A.D. 2018-Jan-01 12:00:00.0000     2.20120548 0.15697382   5.88670258
A.D. 2019-Jan-01 12:00:00.0000     2.20127502 0.15675740   5.88677600
A.D. 2020-Jan-01 12:00:00.0000     2.20208808 0.15597602   5.88879622
A.D. 2021-Jan-01 12:00:00.0000     2.20139994 0.15585909   5.88909976
A.D. 2022-Jan-01 12:00:00.0000     2.20173339 0.15606081   5.88873874
A.D. 2023-Jan-01 12:00:00.0000     2.20113110 0.15636016   5.88908599
```

Try following practice.

Practice 13-09

Use Astroquery to retrieve the orbital elements of Saturn at 12:00:00 (UT) on 01/Jun/2023.

3.2.3 Retrieving state vectors of target body

Make a Python script to retrieve state vectors of a given target body.

Python Code 19: ai202209_s13_01_16.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 10:03:04 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='@ssb', \
                     help='location (default: solar system barycenter)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype
```

```
# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting state vectors
vectors = obj.vectors ()

# printing state vectors
print (vectors)
```

Execute above script.

```
% chmod a+x ai202209_s13_01_16.py
% ./ai202209_s13_01_16.py -h
usage: ai202209_s13_01_16.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location (default: solar system barycenter)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)
```

```
% ./ai202209_s13_01_16.py -t smallbody -i Pluto \
? -s 1970-01-01T12:00:00 -e 2020-01-01T12:00:00 -d 5y
 targetname datetime_jd ...       range             range_rate
    ---          d      ...         AU                AU / d
---------- ----------- ... ----------------- ----------------------
Pluto (999)   2440588.0 ... 31.67360295371376 -0.0005056852762988314
Pluto (999)   2442414.0 ... 30.82216668161667 -0.0004003756167182877
Pluto (999)   2444240.0 ... 30.17997750628867 -0.0002777816865948303
Pluto (999)   2446067.0 ... 29.78312461073157   -0.00013212352216518
Pluto (999)   2447893.0 ... 29.65647490194584  1.392081558880606e-05
Pluto (999)   2449719.0 ... 29.80808049384068  0.0001569655602276837
Pluto (999)   2451545.0 ... 30.22819632240769  0.0002913118509836849
Pluto (999)   2453372.0 ... 30.89147687003037  0.0004195386356305622
Pluto (999)   2455198.0 ... 31.76012265233979  0.0005164517504896146
Pluto (999)   2457024.0 ... 32.79165996716876  0.0005927472160966388
Pluto (999)   2458850.0 ... 33.94254928367312  0.0006479961519385469
```

Print all the column names of Astropy Table.

Python Code 20: ai202209_s13_01_17.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 10:03:19 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
parser.add_argument ('-l', '--location', type=str, default='@ssb', \
                     help='location (default: solar system barycenter)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
```

```
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting state vectors
vectors = obj.vectors ()

# printing state vectors
print (vectors.columns)
```

Execute above script.

```
% chmod a+x ai202209_s13_01_17.py
% ./ai202209_s13_01_17.py -h
usage: ai202209_s13_01_17.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]
```

```
ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location (default: solar system barycenter)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_17.py -t smallbody -i Pluto \
? -s 1970-01-01T12:00:00 -e 2020-01-01T12:00:00 -d 5y
<TableColumns names=('targetname','datetime_jd','datetime_str','x','y','z','vx','
vy','vz','lighttime','range','range_rate')>
```

Make a Python script to retrieve state vectors for a given target body, and print Cartesian coordinate (x, y, and z).

Python Code 21: ai202209_s13_01_18.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 10:03:36 (CST) daisuke>
#

# importing argparse module
import argparse

# importing astropy module
import astropy.time
import astropy.units

# importing astroquery module
import astroquery.jplhorizons

#
# command-line argument analysis
#

# constructing parser object
desc   = "ephemerides of solar system objects"
parser = argparse.ArgumentParser (description=desc)

# choices
choices_idtype = [None, 'smallbody', 'designation', \
                  'name', 'asteroid_name', 'comet_name']

# adding arguments
parser.add_argument ('-t', '--idtype', type=str, choices=choices_idtype, \
                     default=None, help='ID type (default: None)')
parser.add_argument ('-i', '--targetid', type=str, default='1', \
                     help='ID of target body (default: 1)')
```

```
parser.add_argument ('-l', '--location', type=str, default='@ssb', \
                     help='location (default: solar system barycenter)')
parser.add_argument ('-s', '--start', type=str, \
                     default='2000-01-01T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-e', '--end', type=str, \
                     default='2000-01-02T12:00:00', \
                     help='start date/time (default: 2001-01-01T12:00:00)')
parser.add_argument ('-d', '--timestep', type=str, \
                     default='1h', \
                     help='time step (e.g. 1h, 2d, 3y) (default: 1h)')

# parsing arguments
args = parser.parse_args ()

#
# input parameters
#

# ID type
idtype = args.idtype

# target body
target = args.targetid

# location
location = args.location

# start date/time
datetime_start = args.start

# end date/time
datetime_end = args.end

# time step
timestep = args.timestep

# date/time
t_start = astropy.time.Time (datetime_start, scale='utc', format='isot')
t_end   = astropy.time.Time (datetime_end, scale='utc', format='isot')

# making a query to NASA/JPL Horizons system
obj = astroquery.jplhorizons.Horizons (id_type=idtype, id=target, \
                                       location=location, \
                                       epochs={'start': t_start.isot, \
                                               'stop': t_end.isot, \
                                               'step': timestep})

# getting state vectors
vectors = obj.vectors ()

# printing state vectors
print (f"{vectors[0]['targetname']}")
for vec in vectors:
    print (f"{vec['datetime_str']:17s}", \
           f"{vec['x']:+12.6f} {vec['y']:+12.6f} {vec['z']:+12.6f}")
```

Execute above script.

```
% chmod a+x ai202209_s13_01_18.py
% ./ai202209_s13_01_18.py -h
usage: ai202209_s13_01_18.py [-h]
                             [-t {None,smallbody,designation,name,asteroid_name,c
omet_name}]
                             [-i TARGETID] [-l LOCATION] [-s START] [-e END]
                             [-d TIMESTEP]

ephemerides of solar system objects

optional arguments:
  -h, --help            show this help message and exit
  -t {None,smallbody,designation,name,asteroid_name,comet_name}, --idtype {None,s
mallbody,designation,name,asteroid_name,comet_name}
                        ID type (default: None)
  -i TARGETID, --targetid TARGETID
                        ID of target body (default: 1)
  -l LOCATION, --location LOCATION
                        location (default: solar system barycenter)
  -s START, --start START
                        start date/time (default: 2001-01-01T12:00:00)
  -e END, --end END     start date/time (default: 2001-01-01T12:00:00)
  -d TIMESTEP, --timestep TIMESTEP
                        time step (e.g. 1h, 2d, 3y) (default: 1h)

% ./ai202209_s13_01_18.py -t smallbody -i Pluto \
? -s 1970-01-01T12:00:00 -e 2020-01-01T12:00:00 -d 5y
Pluto (999)
A.D. 1970-Jan-01 12:00:00.0000   -30.417943    +2.122799    +8.570858
A.D. 1975-Jan-01 12:00:00.0000   -29.251147    -3.935615    +8.881854
A.D. 1980-Jan-01 12:00:00.0000   -27.101970    -9.859346    +8.894242
A.D. 1985-Jan-01 12:00:00.0000   -23.982188   -15.431531    +8.588190
A.D. 1990-Jan-01 12:00:00.0000   -19.972270   -20.425706    +7.962757
A.D. 1995-Jan-01 12:00:00.0000   -15.209998   -24.650619    +7.037371
A.D. 2000-Jan-01 12:00:00.0000    -9.882485   -27.961583    +5.850661
A.D. 2005-Jan-01 12:00:00.0000    -4.198201   -30.278981    +4.454411
A.D. 2010-Jan-01 12:00:00.0000    +1.625617   -31.584763    +2.909551
A.D. 2015-Jan-01 12:00:00.0000    +7.402166   -31.919846    +1.274487
A.D. 2020-Jan-01 12:00:00.0000   +12.974397   -31.362463    -0.396990
```

Try following practice.

Practice 13-10

Use Astroquery to retrieve the state vector of Neptune at 12:00:00 (UT) on 01/Jul/2023.

4 Playing with Horizons System

4.1 Distance between the Earth and Jovian Trojan asteroid (624) Hektor

Make a Python script to calculate the distance between the Earth and Jovian Trojan asteroid (624) Hektor.

Python Code 22: ai202209_s13_02_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 16:23:59 (CST) daisuke>
#
```

```
#
# Time-stamp: <2021/05/23 16:39:10 (CST) daisuke>
#

# importing numpy module
import numpy

# importing astroquery module
import astroquery.jplhorizons

# querying JPL Horizons
obj_earth   = astroquery.jplhorizons.Horizons (id_type=None, \
                                               id='399', \
                                               location='@ssb', \
                                               epochs={'start': '2000-01-01', \
                                                       'stop': '2023-01-01', \
                                                       'step': '1y'})
obj_hektor = astroquery.jplhorizons.Horizons (id_type='smallbody', \
                                              id='Hektor', \
                                              location='@ssb', \
                                               epochs={'start': '2000-01-01', \
                                                       'stop': '2023-01-01', \
                                                       'step': '1y'})

# state vector of the target object
v_0 = obj_earth.vectors ()
v_1 = obj_hektor.vectors ()

# printing result
print (f"# Distance between the Earth and Jovian Trojan asteroid (624) Hektor:")
print (f"# date/time, (x, y, z) of Earth, (x, y, z) of Hektor, distance in au")
for i in range ( len (v_0) ):
    datetime        = v_0['datetime_str'][i]
    datetime_fields = datetime.split ()
    dx   = v_0['x'][i] - v_1['x'][i]
    dy   = v_0['y'][i] - v_1['y'][i]
    dz   = v_0['z'][i] - v_1['z'][i]
    dist = numpy.sqrt (dx**2 + dy**2 + dz**2)
    print (f"  {datetime_fields[1]}", \
           f" {v_0['x'][i]:+6.3f} {v_0['y'][i]:+6.3f} {v_0['z'][i]:+6.3f}", \
           f" {v_1['x'][i]:+6.3f} {v_1['y'][i]:+6.3f} {v_1['z'][i]:+6.3f}", \
           f" {dist:6.3f}")
```

Execute above script to calculate the distance between Earth and Hektor.

```
% chmod a+x ai202209_s13_02_00.py
% ./ai202209_s13_02_00.py
# Distance between the Earth and Jovian Trojan asteroid (624) Hektor:
# date/time, (x, y, z) of Earth, (x, y, z) of Hektor, distance in au
  2000-Jan-01  -0.176 +0.966 +0.000  -2.125 +4.507 +1.211   4.220
  2001-Jan-01  -0.186 +0.962 +0.000  -4.212 +2.825 +0.480   4.462
  2002-Jan-01  -0.178 +0.962 +0.000  -5.068 +0.325 -0.389   4.946
  2003-Jan-01  -0.171 +0.963 -0.000  -4.450 -2.271 -1.145   5.484
  2004-Jan-01  -0.165 +0.966 -0.000  -2.559 -4.223 -1.574   5.927
  2005-Jan-01  -0.177 +0.966 -0.000  +0.053 -5.009 -1.568   6.181
  2006-Jan-01  -0.173 +0.970 -0.000  +2.648 -4.455 -1.143   6.220
  2007-Jan-01  -0.170 +0.972 -0.000  +4.563 -2.748 -0.421   6.034
  2008-Jan-01  -0.168 +0.974 -0.000  +5.322 -0.342 +0.408   5.660
```

```
  2009-Jan-01   -0.183 +0.971 -0.000   +4.733 +2.156 +1.135    5.182
  2010-Jan-01   -0.180 +0.970 -0.000   +2.933 +4.093 +1.569    4.680
  2011-Jan-01   -0.176 +0.969 -0.000   +0.364 +4.952 +1.589    4.322
  2012-Jan-01   -0.170 +0.968 -0.000   -2.300 +4.464 +1.178    4.260
  2013-Jan-01   -0.181 +0.964 -0.000   -4.321 +2.721 +0.435    4.518
  2014-Jan-01   -0.175 +0.965 -0.000   -5.101 +0.204 -0.430    5.004
  2015-Jan-01   -0.168 +0.967 -0.000   -4.424 -2.371 -1.173    5.535
  2016-Jan-01   -0.163 +0.970 -0.000   -2.505 -4.285 -1.587    5.969
  2017-Jan-01   -0.176 +0.971 -0.000   +0.107 -5.037 -1.570    6.216
  2018-Jan-01   -0.173 +0.974 -0.000   +2.689 -4.464 -1.140    6.250
  2019-Jan-01   -0.172 +0.976 -0.000   +4.590 -2.755 -0.420    6.064
  2020-Jan-01   -0.170 +0.977 -0.000   +5.349 -0.358 +0.405    5.692
  2021-Jan-01   -0.186 +0.973 +0.000   +4.775 +2.135 +1.131    5.219
  2022-Jan-01   -0.183 +0.971 +0.000   +2.999 +4.082 +1.570    4.719
  2023-Jan-01   -0.179 +0.969 +0.000   +0.449 +4.972 +1.602    4.358
```

Try following practice.

Practice 13-11

Use Astroquery to retrieve the state vectors of Neptune and Pluto at 12:00:00 (UT) on 01/Aug/2023, and calculate the distance between these two objects.

4.2 Close approaches of Mars

Make a Python script to calculate the distance between the Earth and the Mars, and visualise the change of the distance with time.

Python Code 23: ai202209_s13_02_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 18:07:16 (CST) daisuke>
#

# importing numpy module
import numpy

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# file name
file_fig = 'ai202209_s13_02_01.png'

# target list
# Earth, Mars
list_obj = ['399', '499']

# start date
date_start = '2015-01-01'
```

```
# end date
date_end   = '2040-01-01'

# step
step = '10d'

# making objects "fig", "canvas", and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111)

# labels
ax.set_xlabel ('Time [year]')
ax.set_ylabel ('Distance between Earth and Mars [au]')

# axes
ax.set_ylim (0, 3.5)
ax.grid ()

# getting positions of Earth and Mars
earth = astroquery.jplhorizons.Horizons (id=list_obj[0], id_type=None, \
                                         location='@ssb', \
                                         epochs={'start': date_start, \
                                                 'stop': date_end, \
                                                 'step': step})
# state vector of the target object
vec_earth = earth.vectors ()

mars = astroquery.jplhorizons.Horizons (id=list_obj[1], id_type=None, \
                                        location='@ssb', \
                                        epochs={'start': date_start, \
                                                'stop': date_end, \
                                                'step': step})
# state vector of the target object
vec_mars = mars.vectors ()

# date/time
datetime   = astropy.time.Time (vec_earth['datetime_jd'], format='jd')
datetime64 = numpy.array (datetime.isot, dtype='datetime64')

# distance between Earth and Mars
delta_x = vec_earth['x'] - vec_mars['x']
delta_y = vec_earth['y'] - vec_mars['y']
delta_z = vec_earth['z'] - vec_mars['z']
dist    = numpy.sqrt (delta_x**2 + delta_y**2 + delta_z**2)

# plotting data
ax.plot (datetime64, dist, \
         linestyle='-', linewidth=3, color='red', \
         label='Distance between Earth and Mars')

# showing legend
ax.legend ()

# saving the plot into a file
fig.savefig (file_fig, dpi=225)
```

Execute above script to calculate the distance.

```
% chmod a+x ai202209_s13_02_01.py
% ./ai202209_s13_02_01.py
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  119960 Dec 11 16:32 ai202209_s13_02_01.png
```

Display PNG file. (Fig. 11) There are close approaches of Mars every $\sim$ 2 years.

```
% feh -dF ai202209_s13_02_01.png
```

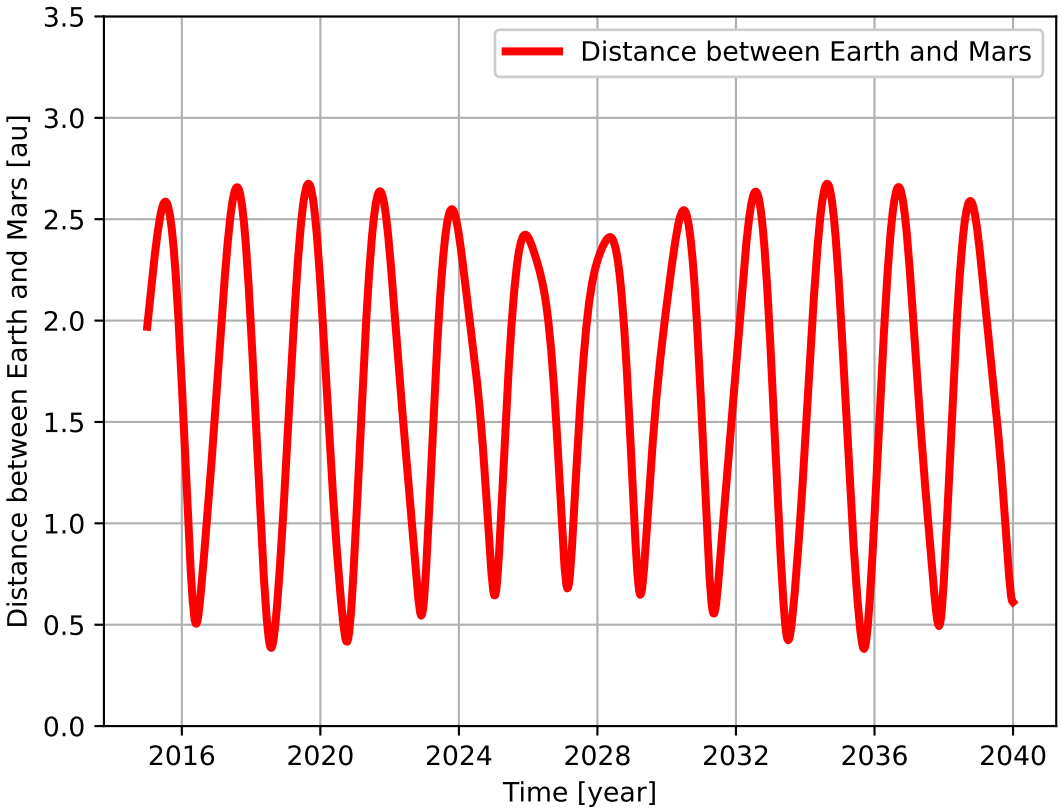

Figure 11: The distance between the Earth and the Mars.

Try following practice.

Practice 13-12

Use Astroquery to retrieve the state vectors of Earth and Venus from 2015 to 2030, calculate the distance between these two objects, and then visualise the change of distance.

4.3 Positions of the Sun and planets

Make a Python script to print positions of the Sun and planets at 00:00 (UT) on 12/Dec/2022.

Python Code 24: ai202209_s13_02_02.py

```
#!/usr/pkg/bin/python3.9
```

```
#
# Time-stamp: <2022/12/11 18:29:50 (CST) daisuke>
#

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# date/time
t_str = '2022-12-12T00:00:00'
t     = astropy.time.Time (t_str, scale='utc', format='isot')

# target list
# Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto
list_obj = ['10', '199', '299', '399', '499', '599', '699', '799', '899', '999']

# names
names = {
    '10':  'Sun',
    '199': 'Mercury',
    '299': 'Venus',
    '399': 'Earth',
    '499': 'Mars',
    '599': 'Jupiter',
    '699': 'Saturn',
    '799': 'Uranus',
    '899': 'Neptune',
    '999': 'Pluto'
}

# printing header
print (f"Positions of the Sun and planets on {t}:")

# getting positions of the Sun and planets
for i in range ( len (list_obj) ):
    # querying JPL Horizons
    obj = astroquery.jplhorizons.Horizons (id=list_obj[i], id_type=None, \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # printing position
    print (f"  {names[list_obj[i]]:10s}:", \
           f"x={vec['x'][0]:+13.8f},", \
           f"y={vec['y'][0]:+13.8f},", \
           f"z={vec['z'][0]:+13.8f}")
```

Execute above script to calculate the distance.

```
% chmod a+x ai202209_s13_02_02.py
% ./ai202209_s13_02_02.py
Positions of the Sun and planets on 2022-12-12T00:00:00.000:
  Sun       : x=  -0.00907140, y=  +0.00027535, z=  +0.00020905
  Mercury   : x=  +0.31860685, y=  -0.22787374, z=  -0.04849150
  Venus     : x=  +0.22639744, y=  -0.68804700, z=  -0.02282785
```

```
  Earth     : x=  +0.16858039, y=  +0.96876006, z=  +0.00015509
  Mars      : x=  +0.31849543, y=  +1.50405243, z=  +0.02369040
  Jupiter   : x=  +4.86045093, y=  +0.89025897, z=  -0.11243472
  Saturn    : x=  +8.08155417, y=  -5.59688730, z=  -0.22444804
  Uranus    : x= +13.41887615, y= +14.37726344, z=  -0.12044648
  Neptune   : x= +29.74597589, y=  -3.00556226, z=  -0.62363164
  Pluto     : x= +16.11062105, y= -30.65669989, z=  -1.37969362
```

Make a Python script to visualise positions of the Sun, Mercury, Venus, Earth, Mars, and Jupiter at 00:00 (UT) on 12/Dec/2022.

Python Code 25: ai202209_s13_02_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 18:43:13 (CST) daisuke>
#

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# file name
file_fig = 'ai202209_s13_02_03.png'

# date/time
t_str = '2022-12-12T00:00:00'
t     = astropy.time.Time (t_str, scale='utc', format='isot')

# target list
# Sun, Mercury, Venus, Earth, Mars, Jupiter
dic_target = {
    '10':  'Sun',
    '199': 'Mercury',
    '299': 'Venus',
    '399': 'Earth',
    '499': 'Mars',
    '599': 'Jupiter',
}

# marker size and colour
sizes   = [10, 2, 5, 5, 4, 8]
colours = ['yellow', 'blue', 'gold', 'green', 'red', 'orange']

# making objects "fig", "canvas", and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111)

# labels
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Y [au]')
```

```
# axes
ax.set_xlim (-5.5, 5.5)
ax.set_ylim (-5.5, 5.5)
ax.set_aspect('equal')
ax.grid ()

# getting positions of the Sun and planets
for i, n in enumerate (dic_target):
    # querying JPL Horizons
    obj = astroquery.jplhorizons.Horizons (id=n, id_type=None, \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # plotting data
    ax.plot (vec['x'], vec['y'], linestyle='None', \
             marker='o', markersize=sizes[i], color=colours[i], \
             label=dic_target[n])

# title of plot
ax.set_title (f"Positions of the Sun and planets on {t_str}")

# showing legend
ax.legend (bbox_to_anchor=(1.01, 1.00), loc='upper left', shadow=True)

# saving the plot into a file
fig.savefig (file_fig, dpi=225)
```

Execute above script to make a plot.

```
% chmod a+x ai202209_s13_02_03.py
% ./ai202209_s13_02_03.py
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  119960 Dec 11 16:32 ai202209_s13_02_01.png
-rw-r--r--  1 daisuke  taiwan   56950 Dec 11 18:39 ai202209_s13_02_03.png
```

Display PNG file. (Fig. 12)

```
% feh -dF ai202209_s13_02_03.png
```

Try following practice.

Practice 13-13

Use Astroquery to retrieve the state vectors of the Sun, Mercury, Venus, Earth, Mars, and Jupiter at 12:00:00 (UT) on 01/Sep/2023. Visualise positions of these objects.

Make a Python script to visualise positions of the Sun, planets and Pluto at 00:00 (UT) on 12/Dec/2022.

Python Code 26: ai202209_s13_02_04.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 18:48:47 (CST) daisuke>
```

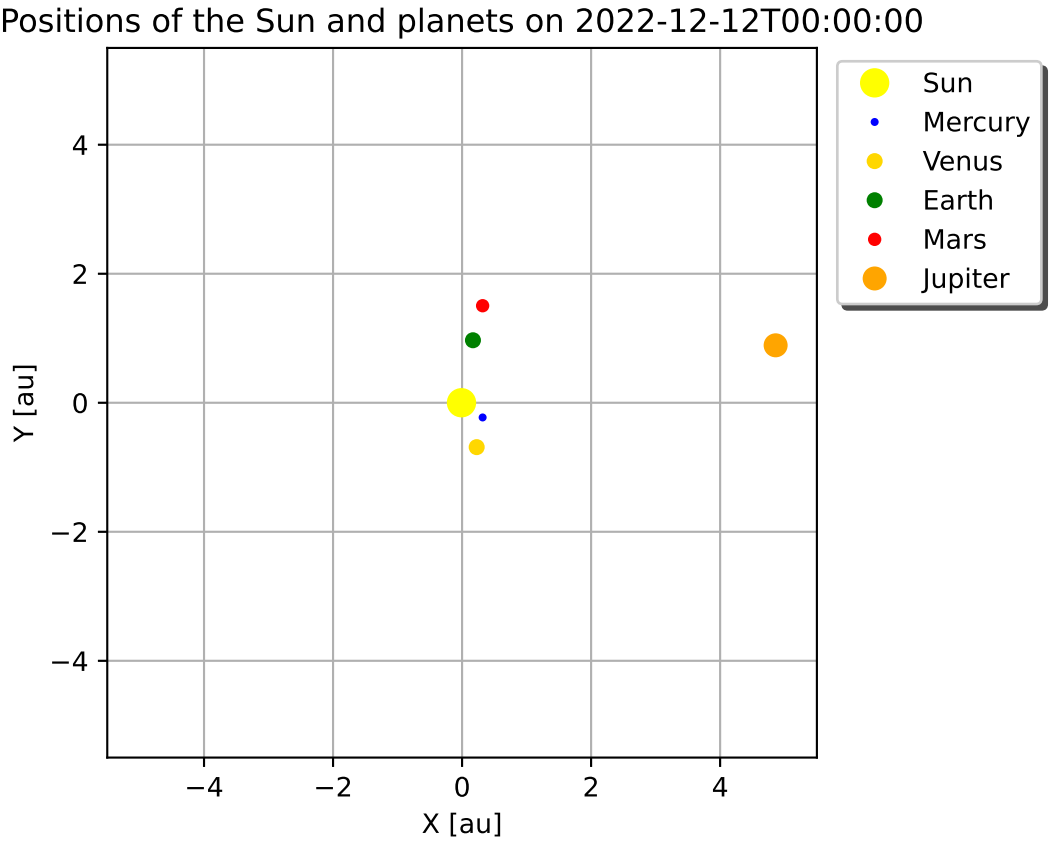

Figure 12: The positions of the Sun, Mercury, Venus, Earth, Mars, and Jupiter at 00:00 (UT) on 12/Dec/2022.

```
#

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# file name
file_fig = 'ai202209_s13_02_04.png'

# date/time
t_str = '2022-12-12T00:00:00'
t     = astropy.time.Time (t_str, scale='utc', format='isot')

# target list
# Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto
dic_target = {
    '10':  'Sun',
    '199': 'Mercury',
    '299': 'Venus',
    '399': 'Earth',
    '499': 'Mars',
    '599': 'Jupiter',
    '699': 'Saturn',
    '799': 'Uranus',
    '899': 'Neptune',
    '999': 'Pluto'
}

# marker size and colour
sizes   = [10, 2, 5, 5, 4, 8, 8, 6, 6, 2]
colours = ['yellow', 'blue', 'gold', 'green', 'red', \
           'orange', 'brown', 'lime', 'indigo', 'slategrey']

# making objects "fig" and "ax"
fig = matplotlib.figure.Figure ()
matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Y [au]')

# axes
ax.set_xlim (-35, 35)
ax.set_ylim (-35, 35)
ax.set_aspect('equal')
ax.grid ()

# getting positions of the Sun and planets
for i, n in enumerate (dic_target):
    # querying JPL Horizons
    obj = astroquery.jplhorizons.Horizons (id=n, id_type=None, \
                                           location='@ssb', \
```

```
                                            epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # plotting data
    ax.plot (vec['x'], vec['y'], linestyle='None', \
             marker='o', markersize=sizes[i], color=colours[i], \
             label=dic_target[n])

# title of plot
ax.set_title (f"Positions of the Sun and planets on {t_str}")

# showing legend
ax.legend (bbox_to_anchor=(1.01, 1.00), loc='upper left', shadow=True)

# saving the plot into a file
fig.savefig (file_fig, dpi=225)
```

Execute above script to make a plot.

```
% chmod a+x ai202209_s13_02_04.py
% ./ai202209_s13_02_04.py
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  119960 Dec 11 16:32 ai202209_s13_02_01.png
-rw-r--r--  1 daisuke  taiwan   56950 Dec 11 18:39 ai202209_s13_02_03.png
-rw-r--r--  1 daisuke  taiwan   77229 Dec 11 18:47 ai202209_s13_02_04.png
```

Display PNG file. (Fig. 13)

```
% feh -dF ai202209_s13_02_04.png
```

Try following practice.

Practice 13-14

Use Astroquery to retrieve the state vectors of the Sun, planets, and Pluto at 12:00:00 (UT) on 01/Oct/2023. Visualise positions of these objects.

4.4 Positions of 500 asteroids

Make a Python script to retrieve positions of 500 asteroids and visualise them.

Python Code 27: ai202209_s13_02_05.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 19:04:32 (CST) daisuke>
#

# importing numpy module
import numpy

# importing astropy module
import astropy.time

# importing astroquery module
```

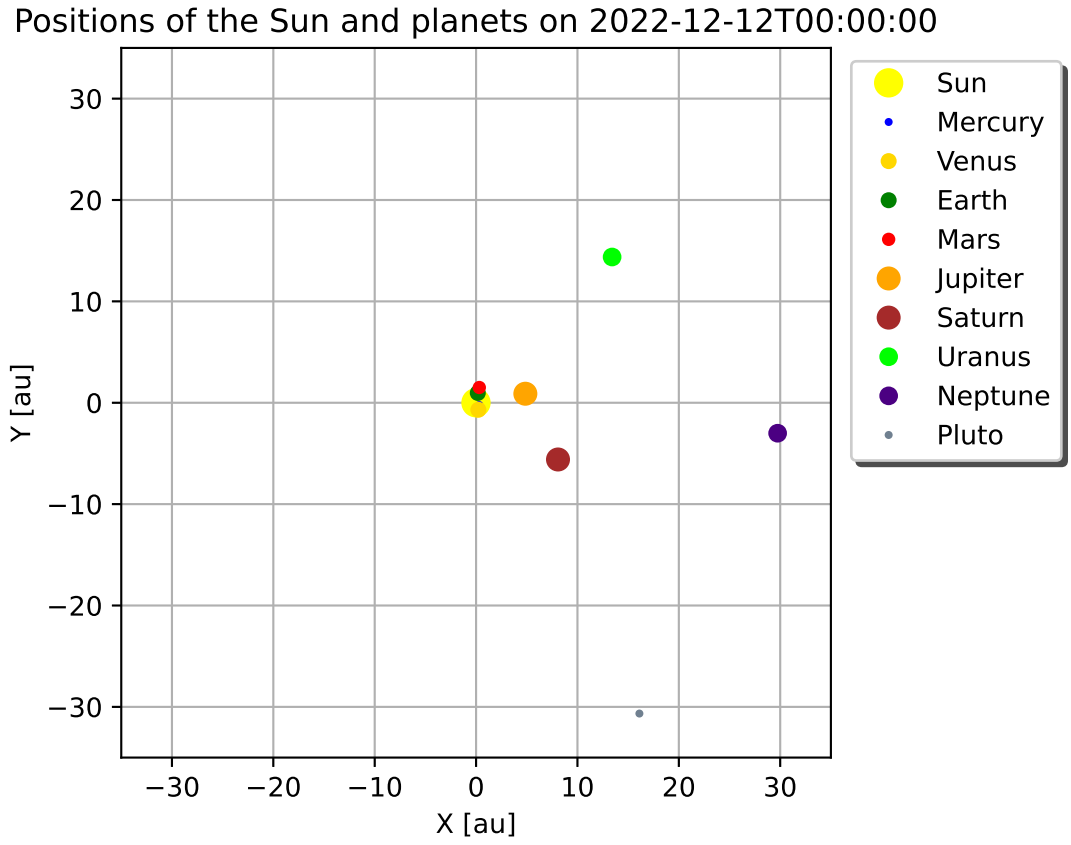

Figure 13: The positions of the Sun, planets, and Pluto at 00:00 (UT) on 12/Dec/2022.

```
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# file name
file_fig = 'ai202209_s13_02_05.png'

# date/time
t_str = '2022-12-12T00:00:00'
t     = astropy.time.Time (t_str, scale='utc', format='isot')

# target list
# Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto
dic_target = {
    '10':  'Sun',
    '199': 'Mercury',
    '299': 'Venus',
    '399': 'Earth',
    '499': 'Mars',
    '599': 'Jupiter',
    '699': 'Saturn',
    '799': 'Uranus',
    '899': 'Neptune',
    '999': 'Pluto'
}

# marker size and colour
sizes   = [10, 2, 5, 5, 4, 8, 8, 6, 6, 2]
colours = ['yellow', 'blue', 'gold', 'green', 'red', \
           'orange', 'brown', 'lime', 'indigo', 'slategrey']

# number of asteroids to be plotted
n_asteroids = 500

# making objects "fig", "canvas", and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111)

# labels
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Y [au]')

# axes
ax.set_xlim (-5.5, 5.5)
ax.set_ylim (-5.5, 5.5)
ax.set_aspect('equal')
ax.grid ()

# getting positions of the Sun and planets
for i,n in enumerate (dic_target):
    # querying JPL Horizons
    obj = astroquery.jplhorizons.Horizons (id=n, id_type=None, \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
```

```
    vec = obj.vectors ()

    # plotting data
    ax.plot (vec['x'], vec['y'], linestyle='None', \
             marker='o', markersize=sizes[i], color=colours[i], \
             label=dic_target[n])

# empty numpy array to store data
asteroid_x = numpy.array ([])
asteroid_y = numpy.array ([])

# getting positions of asteroids
for i in range (n_asteroids):
    # querying JPL Horizons
    i_str = str (i + 1)
    if ( (i + 1) % 10 == 0):
        print (f"progress: {i + 1:4d} / {n_asteroids:4d}")

    # query for Horizons System
    obj = astroquery.jplhorizons.Horizons (id=i_str, id_type='smallbody', \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # appending data to numpy arrays
    asteroid_x = numpy.append (asteroid_x, vec['x'][0])
    asteroid_y = numpy.append (asteroid_y, vec['y'][0])

# plotting asteroids
ax.plot (asteroid_x, asteroid_y, \
         linestyle='None', marker='.', markersize=1, color='purple', \
         label='asteroids')

# showing legend
ax.legend (bbox_to_anchor=(1.01, 1.00), loc='upper left', shadow=True)

# saving the plot into a file
fig.savefig (file_fig, dpi=225)
```

Execute above script to make a plot.

```
% chmod a+x ai202209_s13_02_05.py
% ./ai202209_s13_02_05.py
progress:   10 / 1000
progress:   20 / 1000
progress:   30 / 1000
progress:   40 / 1000
progress:   50 / 1000
progress:   60 / 1000
progress:   70 / 1000
progress:   80 / 1000
progress:   90 / 1000
progress:  100 / 1000

.....

progress:  410 / 1000
```

```
progress:  420 / 1000
progress:  430 / 1000
progress:  440 / 1000
progress:  450 / 1000
progress:  460 / 1000
progress:  470 / 1000
progress:  480 / 1000
progress:  490 / 1000
progress:  500 / 1000
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  119960 Dec 11 16:32 ai202209_s13_02_01.png
-rw-r--r--  1 daisuke  taiwan   56950 Dec 11 18:39 ai202209_s13_02_03.png
-rw-r--r--  1 daisuke  taiwan   77229 Dec 11 18:47 ai202209_s13_02_04.png
-rw-r--r--  1 daisuke  taiwan   73012 Dec 11 19:02 ai202209_s13_02_05.png
```

Display PNG file. (Fig. 14)

```
% feh -dF ai202209_s13_02_05.png
```

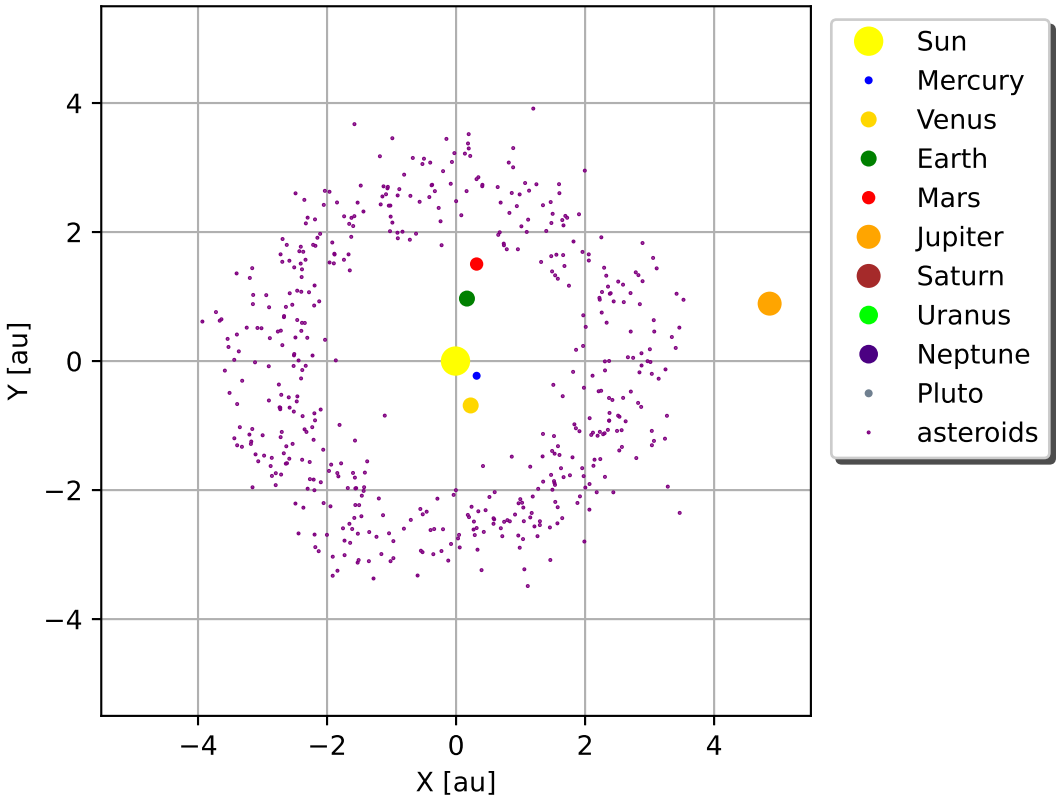

Figure 14: The positions of 500 asteroids at 00:00 (UT) on 12/Dec/2022.

Try following practice.

Practice 13-15

Use Astroquery to retrieve the state vectors of the Sun, planets, Pluto, and 500 asteroids at 12:00:00 (UT) on 01/Nov/2023. Visualise positions of these objects.

4.5 Edge-on view of the solar system

Make a Python script to make an edge-on view of the solar system.

Python Code 28: ai202209_s13_02_06.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 21:49:55 (CST) daisuke>
#

# importing numpy module
import numpy

# importing astropy module
import astropy.time

# importing astroquery module
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# file name
file_fig = 'ai202209_s13_02_06.eps'

# date/time
t_str = '2022-12-12T00:00:00'
t     = astropy.time.Time (t_str, scale='utc', format='isot')

# target list
# Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto
dic_target = {
    '10':  'Sun',
    '199': 'Mercury',
    '299': 'Venus',
    '399': 'Earth',
    '499': 'Mars',
    '599': 'Jupiter',
    '699': 'Saturn',
    '799': 'Uranus',
    '899': 'Neptune',
    '999': 'Pluto'
}

# marker size
sizes   = [10, 2, 5, 5, 4, 8, 8, 6, 6, 2]
colours = ['yellow', 'blue', 'gold', 'green', 'red', \
           'orange', 'brown', 'lime', 'indigo', 'slategrey']

# number of asteroids to be plotted
n_asteroids = 1000

# making objects "fig", "canvas", and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111)

# labels
```

```
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Z [au]')

# axes
ax.set_xlim (-5.5, 5.5)
ax.set_ylim (-2.5, 2.5)
ax.set_aspect ('equal')
ax.grid ()

# empty numpy array to store data
asteroid_x = numpy.array ([])
asteroid_y = numpy.array ([])

# getting positions of asteroids
for i in range (n_asteroids):
    # querying JPL Horizons
    i_str = str (i + 1)
    if ( (i + 1) % 10 == 0):
        print (f"progress: {i + 1:4d} / {n_asteroids:4d}")

    # query for Horizons System
    obj = astroquery.jplhorizons.Horizons (id=i_str, id_type='smallbody', \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # appending data to numpy arrays
    asteroid_x = numpy.append (asteroid_x, vec['x'][0])
    asteroid_y = numpy.append (asteroid_y, vec['z'][0])

ax.plot (asteroid_x, asteroid_y, '.', markersize=1, color='purple', \
         label='asteroids')

# getting positions of the Sun and planets
for i,n in enumerate (dic_target):
    # querying JPL Horizons
    obj = astroquery.jplhorizons.Horizons (id=n, id_type=None, \
                                           location='@ssb', \
                                           epochs=t.jd)

    # state vector of the target object
    vec = obj.vectors ()

    # plotting data
    if (i < 6):
        ax.plot (vec['x'], vec['z'], marker='o', markersize=sizes[i], \
                 color=colours[i], label=dic_target[n])

# title
ax.set_title (f"Edge-on View of Solar System on {t_str}")

# saving the plot into a file
fig.savefig (file_fig, dpi=225)
```

Execute above script to make a plot.

```
% chmod a+x ai202209_s13_02_06.py
```

```
% ./ai202209_s13_02_06.py
progress:   10 / 1000
progress:   20 / 1000
progress:   30 / 1000
progress:   40 / 1000
progress:   50 / 1000
progress:   60 / 1000
progress:   70 / 1000
progress:   80 / 1000
progress:   90 / 1000
progress:  100 / 1000

.....

progress:  910 / 1000
progress:  920 / 1000
progress:  930 / 1000
progress:  940 / 1000
progress:  950 / 1000
progress:  960 / 1000
progress:  970 / 1000
progress:  980 / 1000
progress:  990 / 1000
progress: 1000 / 1000
% ls -lF *.png
-rw-r--r--  1 daisuke  taiwan  119960 Dec 11 16:32 ai202209_s13_02_01.png
-rw-r--r--  1 daisuke  taiwan   56950 Dec 11 18:39 ai202209_s13_02_03.png
-rw-r--r--  1 daisuke  taiwan   77229 Dec 11 18:47 ai202209_s13_02_04.png
-rw-r--r--  1 daisuke  taiwan   73012 Dec 11 19:02 ai202209_s13_02_05.png
-rw-r--r--  1 daisuke  taiwan   69873 Dec 11 19:12 ai202209_s13_02_06.png
```

Display PNG file. (Fig. 15)

```
% feh -dF ai202209_s13_02_06.png
```

Try following practice.

Practice 13-16

Use Astroquery to retrieve the state vectors of the Sun, planets, Pluto, and 500 asteroids at 12:00:00 (UT) on 01/Dec/2023. Visualise the edge-on view of the solar system.

5 Using REBOUND package

Use REBOUND package to carry out orbital integration.

5.1 Making a simulation file

Make a Python script to make a new simulation with one star and one planet and save the simulation into a file.

Python Code 29: ai202209_s13_03_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 19:53:02 (CST) daisuke>
#
```

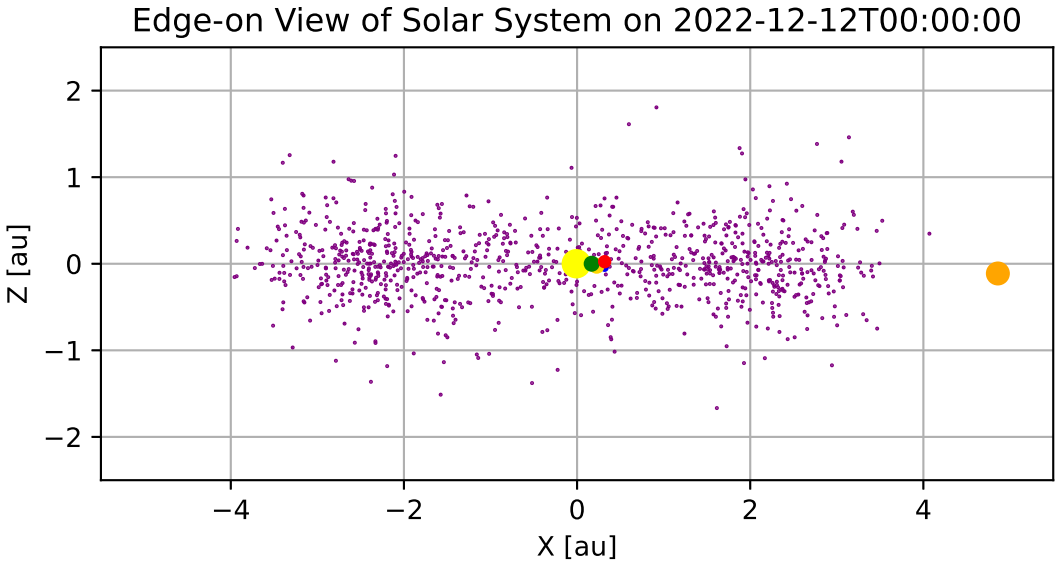

Figure 15: The edge-on view of the solar system at 00:00 (UT) on 12/Dec/2022.

```
# importing rebound module
import rebound

# name of simulation file
file_sim = 'simple0.bin'

# constructing a simulation object
sim = rebound.Simulation ()

# adding a solar mass star
sim.add (m=1.0)

# adding a planet mass object of a=1 and e=0.3
sim.add (m=10**-3, a=1.0, e=0.3)

# printing simulation object
print (sim)

# saving simulation into a file
sim.save (file_sim)
```

Execute above script to make a dynamical simulation for one star and one planet using REBOUND.

```
% chmod a+x ai202209_s13_03_00.py
% ./ai202209_s13_03_00.py
<rebound.simulation.Simulation object at 0x7bad1d5e6290, N=2, t=0.0>
% ls -lF *.bin
-rw-r--r--  1 daisuke  taiwan  2960 Dec 11 19:53 simple0.bin
```

5.2 Reading a simulation from a file

Make a Python script to open and read a simulation from a simulation file.

Python Code 30: ai202209_s13_03_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 20:01:30 (CST) daisuke>
#

# importing rebound module
import rebound

# name of simulation file
file_sim = 'simple0.bin'

# reading a simulation from a file
sim = rebound.Simulation (file_sim)

# printing simulation object
print (sim)
```

Execute above script to read a simulation from a file.

```
% chmod a+x ai202209_s13_03_01.py
% ./ai202209_s13_03_01.py
<rebound.simulation.Simulation object at 0x7a2e469c9990, N=2, t=0.0>
```

5.3 Carrying out orbital integration

Make a Python script to carry out orbital integration of a star and a planet.

Python Code 31: ai202209_s13_03_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 20:37:02 (CST) daisuke>
#

# importing numpy module
import numpy

# importing rebound module
import rebound

# name of simulation file
file_sim = 'simple0.bin'

# reading a simulation from a file
sim = rebound.Simulation (file_sim)

# moving to centre of momentum frame
sim.move_to_com ()

# particles in simulation
ps = sim.particles

# parameters for simulation
#  for G=1, one year is equal to 2 pi
#  0.01 time unit = 365.25 / (2 pi) * 0.01 = 0.58 day
year       = 2.0 * numpy.pi
t_interval = 0.01
n_output   = 1000

# settings for orbital integration
sim.integrator = 'ias15'

# orbital integration
print (f"# days from start of simulation, star's (x, y, z), planet's (x, y, z)")
for i in range (n_output):
    # time
    t     = t_interval * i
    t_day = t_interval * i * 365.25 / (2.0 * numpy.pi)
    # orbital integration
    sim.integrate (t)
    # position of star
    star_x = ps[0].x
    star_y = ps[0].y
    star_z = ps[0].z
    # position of planet
    planet_x = ps[1].x
    planet_y = ps[1].y
    planet_z = ps[1].z
    # printing position of star and planet
    print (f"{t_day:12.6f} {star_x:+10.6f} {star_y:+10.6f} {star_z:+10.6f}", \
           f"{planet_x:+10.6f} {planet_y:+10.6f} {planet_z:+10.6f}")
```

Execute above script to read a simulation from a file and carry out orbital integration.

```
% chmod a+x ai202209_s13_03_02.py
% ./ai202209_s13_03_02.py > simple0.data
% ls -lF *.data
-rw-r--r--  1 daisuke  taiwan  79070 Dec 11 20:40 simple0.data
% head simple0.data
# days from start of simulation, star's (x, y, z), planet's (x, y, z)
    0.000000  -0.000699  +0.000000  +0.000000  +0.699301  +0.000000  +0.000000
    0.581313  -0.000699  -0.000014  +0.000000  +0.699199  +0.013620  +0.000000
    1.162627  -0.000699  -0.000027  +0.000000  +0.698893  +0.027236  +0.000000
    1.743940  -0.000698  -0.000041  +0.000000  +0.698383  +0.040845  +0.000000
    2.325254  -0.000698  -0.000054  +0.000000  +0.697669  +0.054441  +0.000000
    2.906567  -0.000697  -0.000068  +0.000000  +0.696753  +0.068022  +0.000000
    3.487881  -0.000696  -0.000082  +0.000000  +0.695633  +0.081583  +0.000000
    4.069194  -0.000694  -0.000095  +0.000000  +0.694312  +0.095120  +0.000000
    4.650507  -0.000693  -0.000109  +0.000000  +0.692789  +0.108629  +0.000000
```

5.4 Making PNG files

Make a Python script to generate PNG files for location of star and planet.

Python Code 32: ai202209_s13_03_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 21:09:55 (CST) daisuke>
#

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# data file name
file_data = 'simple0.data'

# figure file name prefix
prefix_fig = 'simple0'

# counter
i = 0

# opening data file for reading
with open (file_data, 'r') as fh:
    # reading data file
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (time, star_x, star_y, star_z, planet_x, planet_y, planet_z) \
            = line.split ()
        # figure file name
        file_fig = f"{prefix_fig}_{i:08d}.png"

        # making objects "fig", "canvas", and "ax"
        fig    = matplotlib.figure.Figure ()
        canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
        ax     = fig.add_subplot (111)
```

```
        # labels
        ax.set_xlabel ('X [au]')
        ax.set_ylabel ('Y [au]')

        # axes
        ax.set_xlim (-2.0, +2.0)
        ax.set_ylim (-2.0, +2.0)
        ax.set_aspect ('equal')
        ax.grid ()

        # plotting location of star
        ax.plot (float (star_x), float (star_y), linestyle='None', \
                 marker='o', markersize=10, color='red', label='Star')

        # plotting location of planet
        ax.plot (float (planet_x), float (planet_y), linestyle='None', \
                 marker='o', markersize=3, color='blue', label='Planet')

        # title
        ax.set_title (f"Star and planet at {float (time):6.2f} day")

        # legend
        ax.legend (loc='upper right')

        # saving the plot into a file
        fig.savefig (file_fig, dpi=225)

        # incrementing counter
        i += 1
```

Execute above script to make PNG files.

```
% chmod a+x ai202209_s13_03_03.py
% ./ai202209_s13_03_03.py
% ls simple0_*.png
simple0_00000000.png      simple0_00000334.png      simple0_00000668.png
simple0_00000001.png      simple0_00000335.png      simple0_00000669.png
simple0_00000002.png      simple0_00000336.png      simple0_00000670.png
simple0_00000003.png      simple0_00000337.png      simple0_00000671.png
simple0_00000004.png      simple0_00000338.png      simple0_00000672.png
simple0_00000005.png      simple0_00000339.png      simple0_00000673.png
simple0_00000006.png      simple0_00000340.png      simple0_00000674.png
simple0_00000007.png      simple0_00000341.png      simple0_00000675.png
simple0_00000008.png      simple0_00000342.png      simple0_00000676.png
simple0_00000009.png      simple0_00000343.png      simple0_00000677.png

.....

simple0_00000324.png      simple0_00000658.png      simple0_00000992.png
simple0_00000325.png      simple0_00000659.png      simple0_00000993.png
simple0_00000326.png      simple0_00000660.png      simple0_00000994.png
simple0_00000327.png      simple0_00000661.png      simple0_00000995.png
simple0_00000328.png      simple0_00000662.png      simple0_00000996.png
simple0_00000329.png      simple0_00000663.png      simple0_00000997.png
simple0_00000330.png      simple0_00000664.png      simple0_00000998.png
simple0_00000331.png      simple0_00000665.png      simple0_00000999.png
simple0_00000332.png      simple0_00000666.png
simple0_00000333.png      simple0_00000667.png
% ls simple0_*.png | wc
```

```
    1000     1000    21000
```

Display one of PNG files. (Fig. 16)

```
% feh -dF simple0_00000000.png
```

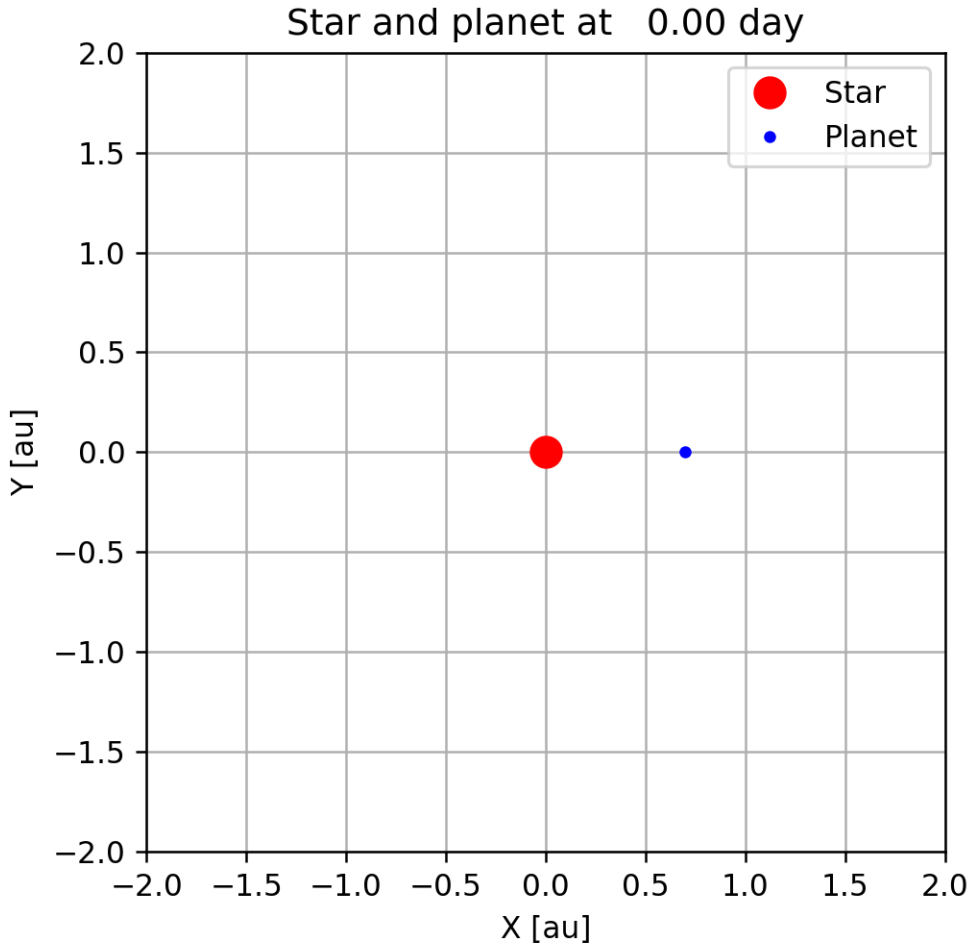

Figure 16: Positions of a star and a planet.

5.5 Making a movie

Use "`ffmpeg`" command to create a movie file.

```
% ffmpeg -f image2 -start_number 0 -framerate 30 -i simple0_%08d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 6 simple0.mp4
% ls -lF simple0.*
-rw-r--r--  1 daisuke  taiwan    2960 Dec 11 20:02 simple0.bin
-rw-r--r--  1 daisuke  taiwan   79070 Dec 11 20:40 simple0.data
-rw-r--r--  1 daisuke  taiwan  432927 Dec 11 21:14 simple0.mp4
```

Play the movie file.

```
% mplayer simple0.mp4
```

Try following practice.

Practice 13-17

Use REBOUND package to carry out orbital integration of a central star of 2 solar mass and a planet of Saturn mass with semimajor axis of 0.5 au and eccentricity of 0.4. Make a MEPG-4 movie to visualise the orbital motion of these objects.

6 A binary system

Simulate a binary system using REBOUND package.

6.1 Making a simulation

Make a Python script to simulate a binary system. The binary system consists of a star of 2 solar mass and a star of 1 solar mass. The semimajor axis of the orbit is 20 au, and the eccentricity is 0.6.

Python Code 33: ai202209_s13_04_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 21:25:34 (CST) daisuke>
#

# importing rebound module
import rebound

# name of simulation file
file_sim = 'simple1.bin'

# constructing a simulation object
sim = rebound.Simulation ()

# adding 2 solar mass star
sim.add (m=2.0)

# adding 1 solar mass star of a=20 au and e=0.6
sim.add (m=1.0, a=20.0, e=0.6)

# printing simulation object
print (sim)

# saving simulation into a file
sim.save (file_sim)
```

Execute above script to make a dynamical simulation for a binary system using REBOUND.

```
% chmod a+x ai202209_s13_04_00.py
% ./ai202209_s13_04_00.py
<rebound.simulation.Simulation object at 0x72fe65b18290, N=2, t=0.0>
% ls -lF *.bin
-rw-r--r--  1 daisuke  taiwan  2960 Dec 11 20:02 simple0.bin
-rw-r--r--  1 daisuke  taiwan  2960 Dec 11 21:27 simple1.bin
```

6.2 Carrying out orbital integration

Make a Python script to carry out orbital integration for a binary system.

Python Code 34: ai202209_s13_04_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 21:35:45 (CST) daisuke>
#

# importing numpy module
import numpy

# importing rebound module
import rebound

# name of simulation file
file_sim = 'simple1.bin'

# reading a simulation from a file
sim = rebound.Simulation (file_sim)

# moving to centre of momentum frame
sim.move_to_com ()

# particles in simulation
ps = sim.particles

# parameters for simulation
#  for G=1, one year is equal to 2 pi
#  1 time unit = 365.25 / (2 pi) = 58 day
year       = 2.0 * numpy.pi
t_interval = 1.0
n_output   = 1000
dt         = 0.01

# settings for orbital integration
sim.integrator = 'ias15'
sim.dt         = dt

# orbital integration
print (f"# year from start of simulation, star 1 (x, y, z), star 2 (x, y, z)")
for i in range (n_output):
    # time
    t    = t_interval * i
    t_yr = t_interval * i / (2.0 * numpy.pi)
    # orbital integration
    sim.integrate (t)
    # position of star 1
    star1_x = ps[0].x
    star1_y = ps[0].y
    star1_z = ps[0].z
    # position of star 2
    star2_x = ps[1].x
    star2_y = ps[1].y
    star2_z = ps[1].z
    # printing position of star and planet
    print (f"{t_yr:12.6f}", \
           f"{star1_x:+10.6f} {star1_y:+10.6f} {star1_z:+10.6f}", \
```

```
            f"{star2_x:+10.6f} {star2_y:+10.6f} {star2_z:+10.6f}")
```

Execute above script to read a simulation from a file and carry out orbital integration.

```
% chmod a+x ai202209_s13_01_01.py
% ./ai202209_s13_04_01.py > simple1.data
% ls -lF *.data
-rw-r--r--  1 daisuke  taiwan  79070 Dec 11 20:40 simple0.data
-rw-r--r--  1 daisuke  taiwan  79068 Dec 11 21:38 simple1.data
% head simple1.data
# year from start of simulation, star 1 (x, y, z), star 2 (x, y, z)
    0.000000  -2.666667  +0.000000  +0.000000  +5.333333  +0.000000  +0.000000
    0.159155  -2.658865  -0.257947  +0.000000  +5.317730  +0.515894  +0.000000
    0.318310  -2.635586  -0.514396  +0.000000  +5.271172  +1.028791  +0.000000
    0.477465  -2.597203  -0.767901  +0.000000  +5.194405  +1.535802  +0.000000
    0.636620  -2.544309  -1.017124  +0.000000  +5.088617  +2.034248  +0.000000
    0.795775  -2.477685  -1.260867  +0.000000  +4.955370  +2.521735  +0.000000
    0.954930  -2.398257  -1.498106  +0.000000  +4.796514  +2.996213  +0.000000
    1.114085  -2.307048  -1.727999  +0.000000  +4.614095  +3.455998  +0.000000
    1.273240  -2.205136  -1.949889  +0.000000  +4.410272  +3.899778  +0.000000
```

6.3 Making PNG files

Make a Python script to generate PNG files for location of two stars in a binary system.

Python Code 35: ai202209_s13_04_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 21:42:41 (CST) daisuke>
#

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# data file name
file_data = 'simple1.data'

# figure file name prefix
prefix_fig = 'simple1'

# counter
i = 0

# opening data file for reading
with open (file_data, 'r') as fh:
    # reading data file
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line
        (time, star1_x, star1_y, star1_z, star2_x, star2_y, star2_z) \
            = line.split ()
        # figure file name
        file_fig = f"{prefix_fig}_{i:08d}.png"
```

```
        # making objects "fig", "canvas", and "ax"
        fig    = matplotlib.figure.Figure ()
        canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
        ax     = fig.add_subplot (111)

        # labels
        ax.set_xlabel ('X [au]')
        ax.set_ylabel ('Y [au]')

        # axes
        ax.set_xlim (-25.0, +25.0)
        ax.set_ylim (-25.0, +25.0)
        ax.set_aspect ('equal')
        ax.grid ()

        # plotting location of star
        ax.plot (float (star1_x), float (star1_y), linestyle='None', \
                 marker='o', markersize=10, color='red', label='Star 1')

        # plotting location of planet
        ax.plot (float (star2_x), float (star2_y), linestyle='None', \
                 marker='o', markersize=5, color='blue', label='Star 2')

        # title
        ax.set_title (f"A binary system at {float (time):6.2f} yr")

        # legend
        ax.legend (loc='upper right')

        # saving the plot into a file
        fig.savefig (file_fig, dpi=225)

        # incrementing counter
        i += 1
```

Execute above script to make PNG files.

```
% chmod a+x ai202209_s13_04_02.py
% ./ai202209_s13_04_02.py
% ls simple1_*.png
simple1_00000000.png      simple1_00000334.png      simple1_00000668.png
simple1_00000001.png      simple1_00000335.png      simple1_00000669.png
simple1_00000002.png      simple1_00000336.png      simple1_00000670.png
simple1_00000003.png      simple1_00000337.png      simple1_00000671.png
simple1_00000004.png      simple1_00000338.png      simple1_00000672.png
simple1_00000005.png      simple1_00000339.png      simple1_00000673.png
simple1_00000006.png      simple1_00000340.png      simple1_00000674.png
simple1_00000007.png      simple1_00000341.png      simple1_00000675.png
simple1_00000008.png      simple1_00000342.png      simple1_00000676.png
simple1_00000009.png      simple1_00000343.png      simple1_00000677.png

.....

simple1_00000324.png      simple1_00000658.png      simple1_00000992.png
simple1_00000325.png      simple1_00000659.png      simple1_00000993.png
simple1_00000326.png      simple1_00000660.png      simple1_00000994.png
simple1_00000327.png      simple1_00000661.png      simple1_00000995.png
simple1_00000328.png      simple1_00000662.png      simple1_00000996.png
simple1_00000329.png      simple1_00000663.png      simple1_00000997.png
```

```
simple1_00000330.png      simple1_00000664.png      simple1_00000998.png
simple1_00000331.png      simple1_00000665.png      simple1_00000999.png
simple1_00000332.png      simple1_00000666.png
simple1_00000333.png      simple1_00000667.png
% ls simple1_*.png | wc
    1000    1000   21000
```

Display one of PNG files. (Fig. 17)

```
% feh -dF simple1_00000000.png
```

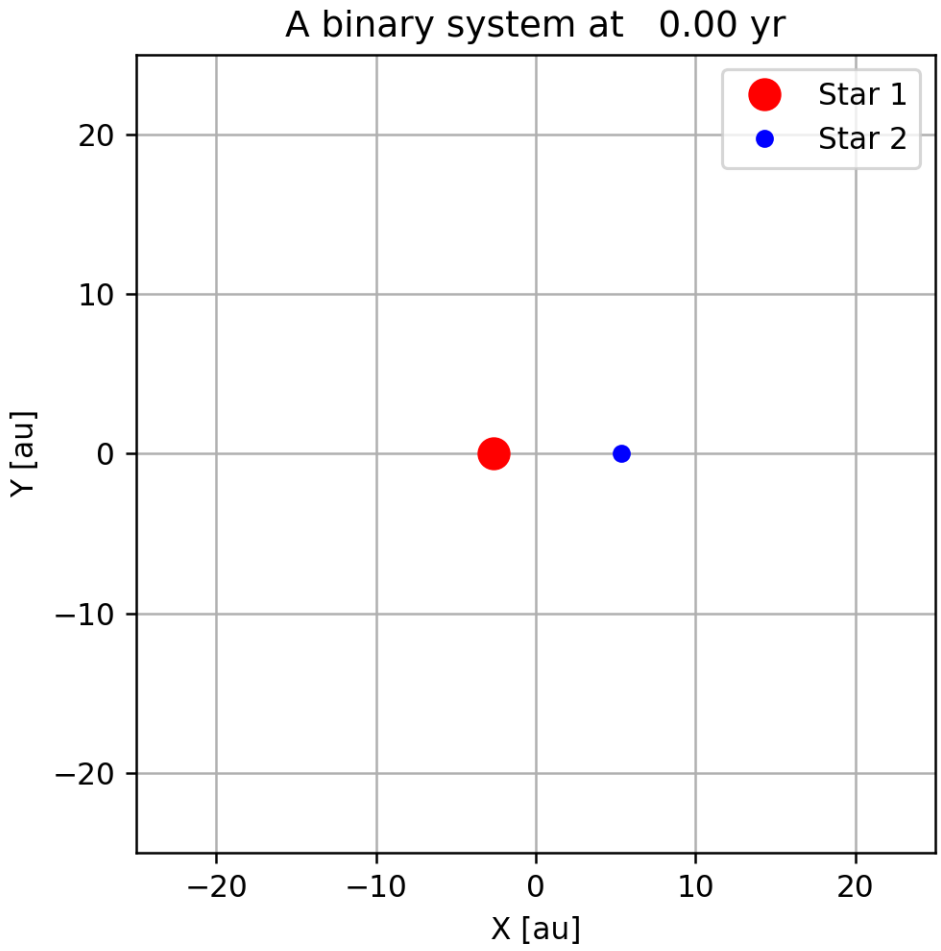

Figure 17: Positions of 2 stars in a binary system.

6.4 Making a movie

Use "`ffmpeg`" command to create a movie file.

```
% ffmpeg -f image2 -start_number 0 -framerate 30 -i simple1_%08d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 6 simple1.mp4
% ls -lF simple1.*
-rw-r--r--  1 daisuke  taiwan    2960 Dec 11 21:27 simple1.bin
```

```
-rw-r--r--  1 daisuke  taiwan   79068 Dec 11 21:38 simple1.data
-rw-r--r--  1 daisuke  taiwan  403375 Dec 11 21:47 simple1.mp4
```

Play the movie file.

```
% mplayer simple1.mp4
```

Try following practice.

Practice 13-18

Use REBOUND package to carry out orbital integration of a star of 5 solar mass and a star of 3 solar mass with semimajor axis of 10 au and eccentricity of 0.7. Make a MPEG-4 movie to visualise the orbital motion of these two stars.

7 Orbital motion of comets in solar system

Carry out orbital integration for some comets in solar system and visualise the orbital motion.

7.1 Making a simulation file

Make a Python script to create a simulation file for REBOUND. Download orbital elements of the Sun, planets, Pluto, and 7 comets.

Python Code 36: ai202209_s13_05_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 22:17:51 (CST) daisuke>
#

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing rebound module
import rebound

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# simulation file to be generated
file_sim = 'comets.bin'

# major bodies
majorbody = {
    'Sun':     '10',
    'Mercury': '1',
    'Venus':   '2',
    'Earth':   '3',
    'Mars':    '4',
    'Jupiter': '5',
```

```
    'Saturn':  '6',
    'Uranus':  '7',
    'Neptune': '8',
    'Pluto':   '9',
}

# minor bodies
minorbody = {
    '1P/Halley':                 'DES=1P; CAP',
    '2P/Encke':                  'DES=2P; CAP',
    '8P/Tuttle':                 'DES=8P; CAP',
    '21P/Giacobini-Zinner':      'DES=21P; CAP',
    '29P/Schwassmann-Wachmann':  'DES=29P; CAP',
    '55P/Tempel-Tuttle':         'DES=55P; CAP',
    '67P/Churyumov-Gerasimenko': 'DES=67P; CAP',
}

# epoch of orbital elements
t_epoch = '2022-12-12 00:00'

# construction of a simulation
sim = rebound.Simulation ()

# adding major bodies
for name in majorbody.keys ():
    sim.add (majorbody[name], date=t_epoch, hash=name)

# adding minor bodies
for name in minorbody.keys ():
    sim.add (minorbody[name], date=t_epoch, hash=name)

# printing simulation
print (sim)

# saving simulation to a file
sim.save (file_sim)
```

Execute above script to make a simulation file.

```
% chmod a+x ai202209_s13_05_00.py
% ./ai202209_s13_05_00.py
Searching NASA Horizons for '10'...
Found: Sun (10)
Searching NASA Horizons for '1'...
Found: Mercury Barycenter (199)
Searching NASA Horizons for '2'...
Found: Venus Barycenter (299)
Searching NASA Horizons for '3'...
Found: Earth-Moon Barycenter (3)
Searching NASA Horizons for '4'...
Found: Mars Barycenter (4)
Searching NASA Horizons for '5'...
Found: Jupiter Barycenter (5)
Searching NASA Horizons for '6'...
Found: Saturn Barycenter (6)
Searching NASA Horizons for '7'...
Found: Uranus Barycenter (7)
Searching NASA Horizons for '8'...
Found: Neptune Barycenter (8)
```

```
Searching NASA Horizons for '9'...
Found: Pluto Barycenter (9)
Searching NASA Horizons for 'DES=1P; CAP'...
Found: 1P/Halley
/usr/pkg/lib/python3.9/site-packages/rebound/horizons.py:168: RuntimeWarning: War
ning: Mass cannot be retrieved from NASA HORIZONS. Set to 0.
  warnings.warn("Warning: Mass cannot be retrieved from NASA HORIZONS. Set to 0."
, RuntimeWarning)
Searching NASA Horizons for 'DES=2P; CAP'...
Found: 2P/Encke
Searching NASA Horizons for 'DES=8P; CAP'...
Found: 8P/Tuttle
Searching NASA Horizons for 'DES=21P; CAP'...
Found: 21P/Giacobini-Zinner
Searching NASA Horizons for 'DES=29P; CAP'...
Found: 29P/Schwassmann-Wachmann 1
Searching NASA Horizons for 'DES=55P; CAP'...
Found: 55P/Tempel-Tuttle
Searching NASA Horizons for 'DES=67P; CAP'...
Found: 67P/Churyumov-Gerasimenko
<rebound.simulation.Simulation object at 0x760b3cb85010, N=17, t=0.0>
% ls -lF *.bin
-rw-r--r--  1 daisuke  taiwan  4880 Dec 11 22:18 comets.bin
-rw-r--r--  1 daisuke  taiwan  2960 Dec 11 20:02 simple0.bin
-rw-r--r--  1 daisuke  taiwan  2960 Dec 11 21:27 simple1.bin
```

7.2 Orbital integration

Make a Python script to carry out orbital integration of the Sun, planets, Pluto, and 7 comets.

Python Code 37: ai202209_s13_05_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 00:11:16 (CST) daisuke>
#

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.time
import astropy.units

# importing rebound module
import rebound

# date/time now
now = datetime.datetime.now ()

# units
u_day = astropy.units.day

# simulation file
file_sim = 'comets.bin'
```

```
# output file
file_output = 'comets.data'

# parameters
year       = 2.0 * numpy.pi
time_epoch = '2022-12-12T00:00:00'
t_interval = 0.1 # 0.1 ==> 365.25/(2.0*pi) * 0.1 = 5.81 days
n_output   = 3000

# objects
objects = [
    'Sun',
    'Mercury',
    'Venus',
    'Earth',
    'Mars',
    'Jupiter',
    'Saturn',
    'Uranus',
    'Neptune',
    'Pluto',
    '1P/Halley',
    '2P/Encke',
    '8P/Tuttle',
    '21P/Giacobini-Zinner',
    '29P/Schwassmann-Wachmann',
    '55P/Tempel-Tuttle',
    '67P/Churyumov-Gerasimenko',
]

# reading simulation from file
sim = rebound.Simulation (file_sim)
sim.integrator = 'mercurius'
sim.dt = +0.01
sim.move_to_com ()

# particles
ps = sim.particles

# opening file for writing
with open (file_output, 'w') as fh:
    # writing header
    fh.write (f"#\n")
    fh.write (f"# results of orbital integration using rebound\n")
    fh.write (f"#\n")
    fh.write (f"#   start of integration: {now}\n")
    fh.write (f"#\n")
    fh.write (f"#   list of objects:\n")
    for name in objects:
        fh.write (f"#     {name}\n")
    fh.write (f"#\n")
    fh.write (f"#   format of the data:\n")
    fh.write (f"#     JD, date/time, x,y,z,vx,vy,vz of obj1, ")
    fh.write (f"x,y,z,vx,vy,vz of obj2, ...\n")
    fh.write (f"#\n")

    # epoch
    t_epoch = astropy.time.Time (time_epoch, scale='utc', format='isot')
```

```
    # orbital integration
    for i in range (n_output):
        # target time
        time = t_interval * i
        # integration
        sim.integrate (time)
        # time after a step of integration
        t_current = t_epoch + 365.25 * sim.t / year * u_day
        jd_current = t_current.jd

        # writing data to file
        fh.write (f"{jd_current:.8f}|{t_current}")
        for j in range ( len (objects) ):
            fh.write (f"|{ps[j].x:+.15f},{ps[j].y:+.15f},{ps[j].z:+.15f},")
            fh.write (f"{ps[j].vx:+.15f},{ps[j].vy:+.15f},{ps[j].vz:+.15f}")
        fh.write (f"\n")

        # printing status
        if ( (i + 1) % 100 == 0 ):
            print (f"  status: {i + 1:08d} / {n_output:08d}")
```

Execute above script to carry out orbital integration.

```
% chmod a+x ai202209_s13_05_01.py
% ./ai202209_s13_05_01.py
  status: 00000100 / 00003000
  status: 00000200 / 00003000
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "taiutc" yielded 1 of "dubious year (Note 4)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
  status: 00000300 / 00003000
  status: 00000400 / 00003000
  status: 00000500 / 00003000
  status: 00000600 / 00003000
  status: 00000700 / 00003000
  status: 00000800 / 00003000
  status: 00000900 / 00003000
  status: 00001000 / 00003000
  status: 00001100 / 00003000
  status: 00001200 / 00003000
  status: 00001300 / 00003000
  status: 00001400 / 00003000
  status: 00001500 / 00003000
  status: 00001600 / 00003000
  status: 00001700 / 00003000
  status: 00001800 / 00003000
  status: 00001900 / 00003000
  status: 00002000 / 00003000
  status: 00002100 / 00003000
  status: 00002200 / 00003000
  status: 00002300 / 00003000
  status: 00002400 / 00003000
  status: 00002500 / 00003000
  status: 00002600 / 00003000
  status: 00002700 / 00003000
```

```
  status: 00002800 / 00003000
  status: 00002900 / 00003000
  status: 00003000 / 00003000
% ls -lF comets.*
-rw-r--r--  1 daisuke  taiwan     4880 Dec 11 22:18 comets.bin
-rw-r--r--  1 daisuke  taiwan  5958786 Dec 11 22:34 comets.data
% head -30 comets.data | cut -b 1-80
#
# results of orbital integration using rebound
#
#   start of integration: 2022-12-11 22:34:12.539711
#
#   list of objects:
#     Sun
#     Mercury
#     Venus
#     Earth
#     Mars
#     Jupiter
#     Saturn
#     Uranus
#     Neptune
#     Pluto
#     1P/Halley
#     2P/Encke
#     8P/Tuttle
#     21P/Giacobini-Zinner
#     29P/Schwassmann-Wachmann
#     55P/Tempel-Tuttle
#     67P/Churyumov-Gerasimenko
#
#   format of the data:
#     JD, date/time, x,y,z,vx,vy,vz of obj1, x,y,z,vx,vy,vz of obj2, ...
#
2459925.50000000|2022-12-12T00:00:00.000|-0.009070693121953,+0.000275654401594,+
2459931.31313430|2022-12-17T19:30:54.803|-0.009066512011633,+0.000223154250206,+
2459937.12626859|2022-12-23T15:01:49.606|-0.009061886781169,+0.000170701591204,+
```

7.3 Making PNG files

Make a Python script to generate PNG files to visualise positions of the Sun, planets, Pluto, and 7 comets.

Python Code 38: ai202209_s13_05_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 22:44:58 (CST) daisuke>
#

# importing datetime module
import datetime

# importing pathlib module
import pathlib

# importing numpy module
import numpy
```

```
# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# data file
file_data = 'comets.data'

# directory to store PNG files
dir_png = 'comets'

# making directory if it does not exist
p_png = pathlib.Path (dir_png)
p_png.mkdir (mode=0o755, exist_ok=True)

# counter
i = 0

# number of major bodies
n_major = 10
# number of minor bodies
n_minor = 7

# opening data file
with open (file_data, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # if the line starts with '#', then skip
        if (line[0] == '#'):
            continue
        # splitting data
        records = line.split ('|')
        # JD
        jd = float (records[0])
        # date/time
        t          = records[1]
        t_datetime = datetime.datetime.fromisoformat (t)
        YYYY       = t_datetime.year

        # PNG file name
        file_png = "%s/%04d.png" % (dir_png, i)

        # making objects "fig" and "canvas"
        fig    = matplotlib.figure.Figure ()
        canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

        # making object "ax"
        ax = fig.add_subplot (121)

        # labels
        ax.set_xlabel ('X [au]')
        ax.set_ylabel ('Y [au]')

        # axes
        ax.set_xlim (-35.0, 35.0)
        ax.set_ylim (-35.0, 35.0)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-30, 30, 7))
        ax.set_yticks (numpy.linspace (-30, 30, 7))
```

```
    for j in ( range (2, n_major + n_minor + 2) ):
        xyz = records[j].split (',')
        x = float (xyz[0])
        y = float (xyz[1])
        z = float (xyz[2])
        if (j == 2):
            sun, = ax.plot (x, y, color='yellow', linestyle='None', \
                            marker='o', markersize=10, label='Sun')
        elif (j == 3):
            mercury, = ax.plot (x, y, color='blue', linestyle='None', \
                                marker='o', markersize=2, label='Mercury')
        elif (j == 4):
            venus, = ax.plot (x, y, color='gold', linestyle='None', \
                              marker='o', markersize=3, label='Venus')
        elif (j == 5):
            earth, = ax.plot (x, y, color='green', linestyle='None', \
                              marker='o', markersize=3, label='Earth')
        elif (j == 6):
            mars, = ax.plot (x, y, color='red', linestyle='None', \
                             marker='o', markersize=2, label='Mars')
        elif (j == 7):
            jupiter, = ax.plot (x, y, color='orange', linestyle='None', \
                                marker='o', markersize=8, label='Jupiter')
        elif (j == 8):
            saturn, = ax.plot (x, y, color='brown', linestyle='None', \
                               marker='o', markersize=7, label='Saturn')
        elif (j == 9):
            uranus, = ax.plot (x, y, color='lime', linestyle='None', \
                               marker='o', markersize=6, label='Uranus')
        elif (j == 10):
            neptune, = ax.plot (x, y, color='indigo', linestyle='None', \
                                marker='o', markersize=6, label='Neptune')
        elif (j == 11):
            pluto, = ax.plot (x, y, color='slategrey', linestyle='None',
                              marker='o', markersize=2, label='Pluto')
        else:
            comets, = ax.plot (x, y, color='aqua', linestyle='None',
                               marker='^', markersize=2,
                               label='comets')

    # showing legend and title
    ax.legend (bbox_to_anchor=(0.0, -0.3), loc='upper left', \
               frameon=False, ncol=2, \
               handles=[sun, mercury, venus, earth, mars, jupiter])
    title_name = "Comets (Year %04d)" % (YYYY)
    ax.set_title (title_name)

    # making object "ax"
    ax = fig.add_subplot (222)

    # labels
    ax.set_xlabel ('X [au]')
    ax.set_ylabel ('Z [au]')

    # axes
    ax.set_xlim (-35.0, 35.0)
    ax.set_ylim (-15, 15)
    ax.set_aspect('equal')
```

```
        ax.grid ()
        ax.set_xticks (numpy.linspace (-30, 30, 7))
        ax.set_yticks (numpy.linspace (-15, 15, 7))

        for j in ( range (2, n_major + n_minor + 2) ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (x, z, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (x, z, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (x, z, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (x, z, color='green', linestyle='None', \
                                  marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (x, z, color='red', linestyle='None', \
                                 marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (x, z, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (x, z, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (x, z, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
                neptune, = ax.plot (x, z, color='indigo', linestyle='None', \
                                    marker='o', markersize=6, label='Neptune')
            elif (j == 11):
                pluto, = ax.plot (x, z, color='slategrey', linestyle='None',
                                  marker='o', markersize=2, label='Pluto')
            else:
                comets, = ax.plot (x, z, color='aqua', linestyle='None',
                                   marker='^', markersize=2,
                                   label='comets')

        # making object "ax"
        ax = fig.add_subplot (224)

        # labels
        ax.set_xlabel ('Y [au]')
        ax.set_ylabel ('Z [au]')

        # axes
        ax.set_xlim (-35.0, 35.0)
        ax.set_ylim (-15, 15)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-30, 30, 7))
        ax.set_yticks (numpy.linspace (-15, 15, 7))
```

```
        for j in ( range (2, n_major + n_minor + 2) ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (y, z, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (y, z, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (y, z, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (y, z, color='green', linestyle='None', \
                                  marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (y, z, color='red', linestyle='None', \
                                 marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (y, z, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (y, z, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (y, z, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
                neptune, = ax.plot (y, z, color='indigo', linestyle='None', \
                                    marker='o', markersize=6, label='Neptune')
            elif (j == 11):
                pluto, = ax.plot (y, z, color='slategrey', linestyle='None', \
                                  marker='o', markersize=2, label='Pluto')
            else:
                comets, = ax.plot (y, z, color='aqua', linestyle='None', \
                                   marker='^', markersize=2, \
                                   label='comets')

        ax.legend (bbox_to_anchor=(0.0, -0.4), loc='upper left', \
                   frameon=False, ncol=2, \
                   handles=[saturn, uranus, neptune, pluto, comets])
        # saving the plot into a file
        fig.tight_layout ()
        fig.savefig (file_png, dpi=225)

        # increment
        i += 1
        if (i % 100 == 0):
            print (f"progress: {i}")
```

Execute above script to make PNG files.

```
% chmod a+x ai202209_s13_05_02.py
% ./ai202209_s13_05_02.py
progress: 100
progress: 200
progress: 300
```

```
progress: 400
progress: 500
progress: 600
progress: 700
progress: 800
progress: 900
progress: 1000

.....

progress: 2100
progress: 2200
progress: 2300
progress: 2400
progress: 2500
progress: 2600
progress: 2700
progress: 2800
progress: 2900
progress: 3000
% ls comets
0000.png  0375.png  0750.png  1125.png  1500.png  1875.png  2250.png  2625.png
0001.png  0376.png  0751.png  1126.png  1501.png  1876.png  2251.png  2626.png
0002.png  0377.png  0752.png  1127.png  1502.png  1877.png  2252.png  2627.png
0003.png  0378.png  0753.png  1128.png  1503.png  1878.png  2253.png  2628.png
0004.png  0379.png  0754.png  1129.png  1504.png  1879.png  2254.png  2629.png
0005.png  0380.png  0755.png  1130.png  1505.png  1880.png  2255.png  2630.png
0006.png  0381.png  0756.png  1131.png  1506.png  1881.png  2256.png  2631.png
0007.png  0382.png  0757.png  1132.png  1507.png  1882.png  2257.png  2632.png
0008.png  0383.png  0758.png  1133.png  1508.png  1883.png  2258.png  2633.png
0009.png  0384.png  0759.png  1134.png  1509.png  1884.png  2259.png  2634.png

.....

0365.png  0740.png  1115.png  1490.png  1865.png  2240.png  2615.png  2990.png
0366.png  0741.png  1116.png  1491.png  1866.png  2241.png  2616.png  2991.png
0367.png  0742.png  1117.png  1492.png  1867.png  2242.png  2617.png  2992.png
0368.png  0743.png  1118.png  1493.png  1868.png  2243.png  2618.png  2993.png
0369.png  0744.png  1119.png  1494.png  1869.png  2244.png  2619.png  2994.png
0370.png  0745.png  1120.png  1495.png  1870.png  2245.png  2620.png  2995.png
0371.png  0746.png  1121.png  1496.png  1871.png  2246.png  2621.png  2996.png
0372.png  0747.png  1122.png  1497.png  1872.png  2247.png  2622.png  2997.png
0373.png  0748.png  1123.png  1498.png  1873.png  2248.png  2623.png  2998.png
0374.png  0749.png  1124.png  1499.png  1874.png  2249.png  2624.png  2999.png
% ls comets/*.png | wc
    3000    3000   48000
```

Display one of PNG files. (Fig. 18)

```
% feh -dF comets/0000.png
```

7.4 Making a MP4 movie

Use "`ffmpeg`" command to create a movie file.

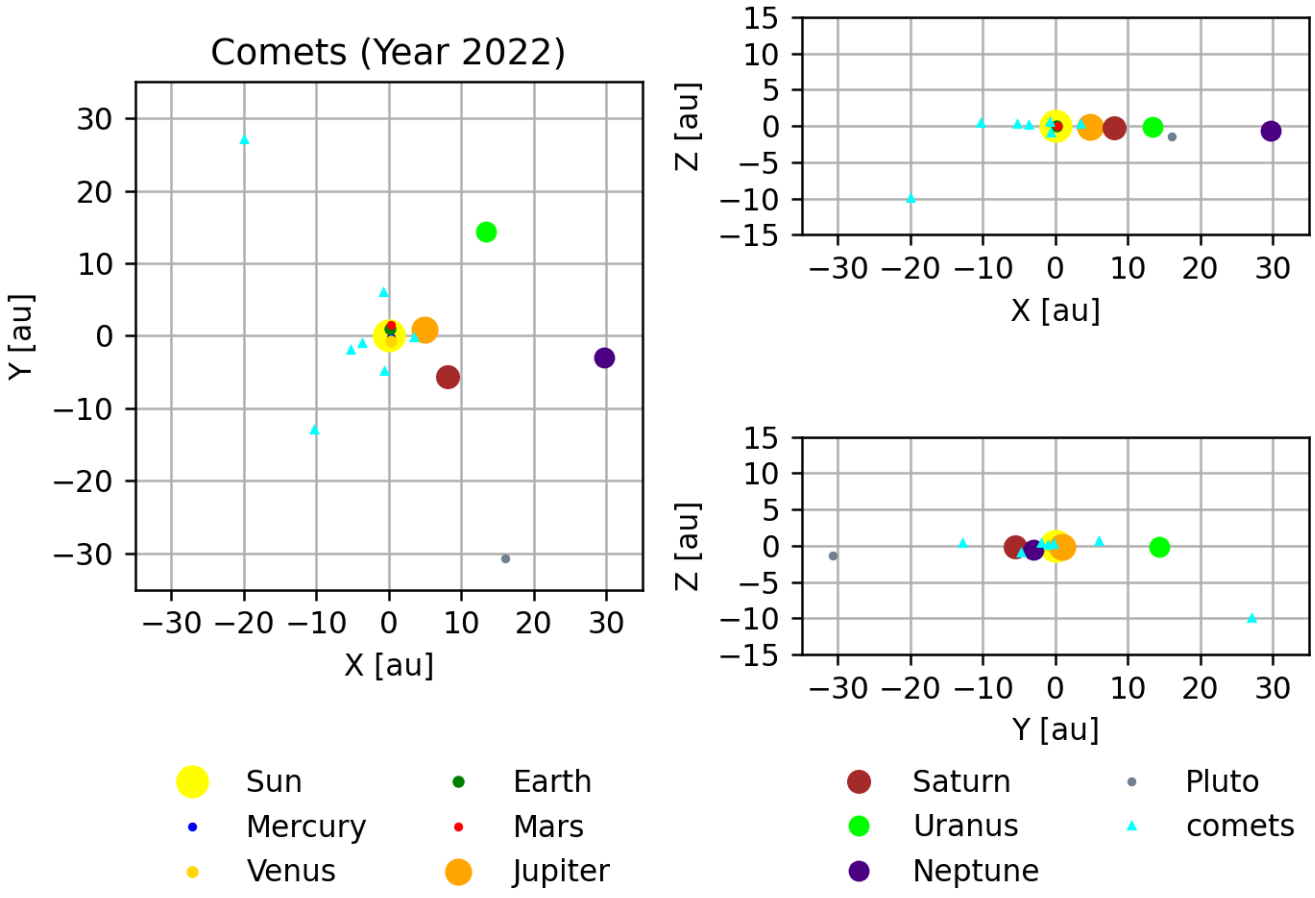

Figure 18: Positions of the Sun, planets, Pluto, and 7 comets.

```
% ffmpeg -f image2 -start_number 0 -framerate 30 -i comets/%04d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 6 comets.mp4
% ls -lF comets.*
-rw-r--r--  1 daisuke  taiwan     4880 Dec 11 22:18 comets.bin
-rw-r--r--  1 daisuke  taiwan  5958786 Dec 11 22:34 comets.data
-rw-r--r--  1 daisuke  taiwan  2213431 Dec 11 23:17 comets.mp4
```

Play the movie file.

```
% mplayer comets.mp4
```

Try following practice.

Practice 13-19

Use REBOUND package to carry out orbital integration of the Sun, planets, Pluto, and your favourite comet. Make a MPEG-4 movie to visualise the orbital motion of these objects.

8 Orbital motion of Jovian Trojan asteroids

Carry out orbital integration for Jovian Trojan asteroids in solar system and visualise the orbital motion.

8.1 Downloading asteroid orbit database file

Make a Python script to download the asteroid orbit database file from the Minor Planet Center of IAU.

Python Code 39: ai202209_s13_06_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 23:23:56 (CST) daisuke>
#

# importing urllib module
import urllib.request

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# URL of data file
url_data = 'https://www.minorplanetcenter.net/iau/MPCORB/MPCORB.DAT.gz'

# output file name
file_output = 'MPCORB.DAT.gz'

# printing status
print (f'Now, fetching {url_data}...')

# opening URL
with urllib.request.urlopen (url_data) as fh_read:
    # reading data
    data_byte = fh_read.read ()
```

```
# printing status
print (f'Finished fetching {url_data}!')

# printing status
print (f'Now, writing the data into file "{file_output}"...')

# opening file for writing
with open (file_output, 'wb') as fh_write:
    # writing data
    fh_write.write (data_byte)

# printing status
print (f'Finished writing the data into file "{file_output}"!')
```

Execute above script to download the data file.

```
% chmod a+x ai202209_s13_06_00.py
% ./ai202209_s13_06_00.py
Now, fetching https://www.minorplanetcenter.net/iau/MPCORB/MPCORB.DAT.gz...
Finished fetching https://www.minorplanetcenter.net/iau/MPCORB/MPCORB.DAT.gz!
Now, writing the data into file "MPCORB.DAT.gz"...
Finished writing the data into file "MPCORB.DAT.gz"!
% ls -lF MPCORB.DAT.gz
-rw-r--r--  1 daisuke  taiwan  74847748 Dec 11 23:26 MPCORB.DAT.gz
% gunzip -c MPCORB.DAT.gz | head -50 | cut -b 1-80
MINOR PLANET CENTER ORBIT DATABASE (MPCORB)

This file contains published orbital elements for all numbered and unnumbered
multi-opposition minor planets for which it is possible to make reasonable
predictions.  It also includes published elements for recent one-opposition
minor planets and is intended to be complete through the last issued Daily
Orbit Update MPEC.  As such it is intended to be of interest primarily
to astrometric observers.

   Software programs may include this datafile amongst their datasets, as
   long as this header is included (it is acceptable if it is contained
   in a file separate from the actual data) and that proper attribution
   to the Minor Planet Center is given.  Credit to the individual orbit
   computers is implicit by the inclusion of a reference and the name of
   the orbit computer on each orbit record.  Information on how to obtain
   updated copies of the datafile must also be included.

   The work of the individual astrometric observers, without whom none of
   the work of the Minor Planet Center would be possible, is gratefully
   acknowledged.  Credit to the individual observers is implicit by the
   inclusion of the reference to the publication of their observations in
   all data sets distributed by the Minor Planet Center.

New versions of this file, updated on a daily basis, will be available at:

          https://www.minorplanetcenter.org/iau/MPCORB/MPCORB.DAT

The elements contained within MPCORB are divided into three sections,
separated by blank lines.  The first section contains the numbered objects,
the second section contains the unnumbered objects with perturbed orbit
solutions and the third contains the recent 1-opposition objects with
unperturbed orbit solutions.  Each object's elements are stored on a single
line, the format of which is described at:
```

```
          http://www.minorplanetcenter.org/iau/info/MPOrbitFormat.html

If you find a problem with any data herein, please contact mpc@cfa.harvard.edu.

A brief header is given below:

Des'n     H     G   Epoch     M        Peri.      Node       Incl.       e

--------------------------------------------------------------------------------
00001     3.33  0.15 K232P  17.21569   73.47045   80.26013   10.58634  0.0788175
00002     4.12  0.15 K232P 357.84942  310.86480  172.91814   34.92703  0.2300844
00003     5.14  0.15 K232P 351.82413  247.73655  169.84298   12.99067  0.2564677
00004     3.21  0.15 K232P 115.13301  151.59911  103.75733    7.13926  0.0887575
00005     7.00  0.15 K232P 256.02913  358.88647  141.52285    5.36013  0.1879237
00006     5.60  0.15 K232P  91.86530  239.54004  138.63825   14.73757  0.2027524
00007     5.62  0.15 K232P 154.48051  145.41270  259.50440    5.51780  0.2296597
```

8.2 Finding Jovian Trojan asteroids

Make a Python script to open the MPCORB orbit database file, read the data, and find Jovian Trojan asteroids.

Python Code 40: ai202209_s13_06_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 23:45:41 (CST) daisuke>
#

# importing gzip module
import gzip

# importing numpy module
import numpy

# MPC's orbital elements file
file_mpcorb = 'MPCORB.DAT.gz'

# number of asteroids to process
n_asteroids = 3000

# dictionary to store orbital elements
dic_elements = {}

# counter
n_jt = 0

# opening file
with gzip.open (file_mpcorb, 'rb') as fh:
    # flag
    data_line = 'NO'
    # reading file
    for line in fh:
        # decoding byte data
        line = line.decode ('utf-8')
        if (data_line == 'YES'):
            # number (or provisional designation)
            number = line[0:7]
            # epoch
```

```
            epoch = line[20:25]
            # mean anomaly
            M = float (line[26:35])
            M_rad = numpy.deg2rad (M)
            # argument of perihelion
            peri = float (line[37:46])
            peri_rad = numpy.deg2rad (peri)
            # longitude of ascending node
            node = float (line[48:57])
            node_rad = numpy.deg2rad (node)
            # inclination
            i = float (line[59:68])
            i_rad = numpy.deg2rad (i)
            # eccentricity
            e = float (line[70:79])
            # semimajor axis
            a = float (line[92:103])
            # orbit type
            orbit_type = line[161:165]

            # adding data to the dictionary if Jovian Trojan asteroid
            if (orbit_type == '0009'):
                dic_elements[number] = {}
                dic_elements[number]['a'] = a
                dic_elements[number]['e'] = e
                dic_elements[number]['i'] = i_rad
                dic_elements[number]['peri'] = peri_rad
                dic_elements[number]['node'] = node_rad
                dic_elements[number]['M'] = M_rad
                # increment
                n_jt += 1

            # when finish reading expected number of asteroid data, then break
            if (n_jt >= n_asteroids):
                break
        # if the line starts with '----------'
        if (line[:10] == '----------'):
            # set the flag to 'YES'
            data_line = 'YES'
            continue

for number in sorted (dic_elements.keys ()):
    print (f"{number:8s}", \
           f"{dic_elements[number]['a']:10.8f}", \
           f"{dic_elements[number]['e']:10.8f}", \
           f"{dic_elements[number]['i']:10.8f}", \
           f"{dic_elements[number]['peri']:10.8f}", \
           f"{dic_elements[number]['node']:10.8f}", \
           f"{dic_elements[number]['M']:10.8f}")
```

Execute above script to find Jovian Trojan asteroids from the MPCORB orbit database file.

```
% chmod a+x ai202209_s13_06_01.py
% ./ai202209_s13_06_01.py > trojan.list
% ls -lF trojan.list
-rw-r--r--  1 daisuke  taiwan  225000 Dec 11 23:47 trojan.list
% head trojan.list
00588    5.20965500 0.14827780 0.18011466 2.33324009 5.52459435 6.18592579
00617    5.20886080 0.13988410 0.38506170 5.38095149 0.77423416 5.57644006
```

```
00624    5.27063610 0.02299890 0.31686086 3.14171221 5.98278866 5.03378857
00659    5.16818940 0.11669010 0.07893147 6.00508420 6.12161913 1.71685804
00884    5.19954580 0.12547680 0.15539591 5.87398833 5.26299097 0.82474679
00911    5.28234490 0.06756690 0.37981576 1.42234997 5.89935459 0.38305910
01143    5.23695230 0.09181310 0.05483004 4.14952937 3.86110377 5.77358983
01172    5.22920190 0.10666290 0.29064933 0.89454198 4.31600900 0.47137329
01173    5.27901960 0.13904550 0.12087330 0.72199245 4.95514595 5.75280401
01208    5.25215620 0.09274910 0.58544295 5.16148768 0.84740709 5.88468754
```

8.3 Making a simulation file for REBOUND

Make a Python script to create a simulation for REBOUND.

Python Code 41: ai202209_s13_06_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/11 23:54:08 (CST) daisuke>
#

# importing gzip module
import gzip

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing rebound module
import rebound

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# MPC's orbital elements file
file_mpcorb = 'MPCORB.DAT.gz'

# simulation file to be generated
file_sim = 'trojan.bin'

majorbody = {
    'Sun':     '10',
    'Mercury': '1',
    'Venus':   '2',
    'Earth':   '3',
    'Mars':    '4',
    'Jupiter': '5',
    'Saturn':  '6',
    'Uranus':  '7',
    'Neptune': '8',
    'Pluto':   '9',
}

# number of asteroids to process
```

```
n_asteroids = 3000

# counter
n_jt = 0

# epoch of orbital elements
# K232P => 2023-Feb-25
# if the epoch is not K232P, then change following line
t_epoch = '2023-02-25 00:00'

# construction of a simulation
sim = rebound.Simulation ()

# adding major bodies
for name in majorbody.keys ():
    sim.add (majorbody[name], date=t_epoch)

# dictionary to store orbital elements
dic_elements = {}

# opening file
with gzip.open (file_mpcorb, 'rb') as fh:
    # flag
    data_line = 'NO'
    # reading file
    for line in fh:
        # decoding byte data
        line = line.decode ('utf-8')
        if (data_line == 'YES'):
            # number (or provisional designation)
            number = line[0:7]
            # epoch
            epoch = line[20:25]
            # mean anomaly
            M = float (line[26:35])
            M_rad = numpy.deg2rad (M)
            # argument of perihelion
            peri = float (line[37:46])
            peri_rad = numpy.deg2rad (peri)
            # longitude of ascending node
            node = float (line[48:57])
            node_rad = numpy.deg2rad (node)
            # inclination
            i = float (line[59:68])
            i_rad = numpy.deg2rad (i)
            # eccentricity
            e = float (line[70:79])
            # semimajor axis
            a = float (line[92:103])
            # orbit type
            orbit_type = line[161:165]

            # adding data to the dictionary
            if (orbit_type == '0009'):
                dic_elements[number] = {}
                dic_elements[number]['a'] = a
                dic_elements[number]['e'] = e
                dic_elements[number]['i'] = i_rad
                dic_elements[number]['peri'] = peri_rad
```

```
                dic_elements[number]['node'] = node_rad
                dic_elements[number]['M'] = M_rad
                # increment
                n_jt += 1

            # when finish reading expected number of asteroid data, then break
            if (n_jt >= n_asteroids):
                break
        # if the line starts with '----------'
        if (line[:10] == '----------'):
            # set the flag to 'YES'
            data_line = 'YES'
            continue

# processing each asteroid orbit
for number in sorted (dic_elements.keys ()):
    # adding an asteroid
    sim.add (m=0.0, \
             a=dic_elements[number]['a'], \
             e=dic_elements[number]['e'], \
             inc=dic_elements[number]['i'], \
             omega=dic_elements[number]['peri'], \
             Omega=dic_elements[number]['node'], \
             M=dic_elements[number]['M'], \
             date=t_epoch)

# printing simulation object
print (sim)

# saving simulation to a file
sim.save (file_sim)
```

Execute above script to make a simulation for REBOUND.

```
% chmod a+x ai202209_s13_06_02.py
% ./ai202209_s13_06_02.py
Searching NASA Horizons for '10'...
Found: Sun (10)
Searching NASA Horizons for '1'...
Found: Mercury Barycenter (199)
Searching NASA Horizons for '2'...
Found: Venus Barycenter (299)
Searching NASA Horizons for '3'...
Found: Earth-Moon Barycenter (3)
Searching NASA Horizons for '4'...
Found: Mars Barycenter (4)
Searching NASA Horizons for '5'...
Found: Jupiter Barycenter (5)
Searching NASA Horizons for '6'...
Found: Saturn Barycenter (6)
Searching NASA Horizons for '7'...
Found: Uranus Barycenter (7)
Searching NASA Horizons for '8'...
Found: Neptune Barycenter (8)
Searching NASA Horizons for '9'...
Found: Pluto Barycenter (9)
<rebound.simulation.Simulation object at 0x7637e9625290, N=3010, t=0.0>
% ls -lF trojan.*
-rw-r--r--  1 daisuke  taiwan  387984 Dec 11 23:54 trojan.bin
```

```
-rw-r--r--  1 daisuke  taiwan  225000 Dec 11 23:47 trojan.list
```

8.4 Carrying out orbital integration

Make a Python script to carry out orbital integration of Jovian Trojan asteroids.

Python Code 42: ai202209_s13_06_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 00:11:23 (CST) daisuke>
#

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.time
import astropy.units

# importing rebound module
import rebound

# date/time now
now = datetime.datetime.now ()

# units
u_day = astropy.units.day

# simulation file
file_sim = 'trojan.bin'

# output file
file_output = 'trojan.data'

# parameters
year       = 2.0 * numpy.pi
time_epoch = '2023-02-25T00:00:00'
t_interval = 0.1 # 0.1 ==> 365.25/(2.0*pi) * 0.1 = 5.81 days
n_output   = 1000

# major bodies
majorbody = [
    'Sun',
    'Mercury',
    'Venus',
    'Earth',
    'Mars',
    'Jupiter',
    'Saturn',
    'Uranus',
    'Neptune',
    'Pluto',
]
```

```
# number of minor bodies
n_minorbody = 3000

# reading simulation from file
sim = rebound.Simulation (file_sim)
sim.integrator = 'mercurius'
sim.dt = +0.01
sim.move_to_com ()

# particles
ps = sim.particles

# opening file for writing
with open (file_output, 'w') as fh:
    # writing header
    fh.write (f"#\n")
    fh.write (f"# results of orbital integration using rebound\n")
    fh.write (f"#\n")
    fh.write (f"#   start of integration: {now}\n")
    fh.write (f"#\n")
    fh.write (f"#   list of objects:\n")
    for name in majorbody:
        fh.write (f"#     {name}\n")
    fh.write (f"#     and {n_minorbody} minor bodies\n")
    fh.write (f"#\n")
    fh.write (f"#   format of the data:\n")
    fh.write (f"#     JD, date/time, x,y,z,vx,vy,vz of obj1, ")
    fh.write (f"x,y,z,vx,vy,vz of obj2, ...\n")
    fh.write (f"#\n")

    # epoch
    t_epoch = astropy.time.Time (time_epoch, scale='utc', format='isot')

    # orbital integration
    for i in range (n_output):
        # target time
        time = t_interval * i
        # integration
        sim.integrate (time)
        # time after a step of integration
        t_current = t_epoch + 365.25 * sim.t / year * u_day
        jd_current = t_current.jd

        # writing data to file
        fh.write (f"{jd_current:.8f}|{t_current}")
        for j in range ( len (majorbody) + n_minorbody):
            fh.write (f"|{ps[j].x:+.15f},{ps[j].y:+.15f},{ps[j].z:+.15f},")
            fh.write (f"{ps[j].vx:+.15f},{ps[j].vy:+.15f},{ps[j].vz:+.15f}")
        fh.write (f"\n")

        # printing status
        if ( (i + 1) % 100 == 0 ):
            print (f"  status: {i + 1:08d} / {n_output:08d}")
```

Execute above script to carry out orbital integration.

```
% chmod a+x ai202209_s13_06_03.py
% ./ai202209_s13_06_03.py
  status: 00000100 / 00001000
```

```
  status: 00000200 / 00001000
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "taiutc" yielded 1 of "dubious year (Note 4)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
  status: 00000300 / 00001000
  status: 00000400 / 00001000
  status: 00000500 / 00001000
  status: 00000600 / 00001000
  status: 00000700 / 00001000
  status: 00000800 / 00001000
  status: 00000900 / 00001000
  status: 00001000 / 00001000
% ls -lF trojan.*
-rw-r--r--  1 daisuke  taiwan     387984 Dec 11 23:54 trojan.bin
-rw-r--r--  1 daisuke  taiwan  343185847 Dec 12 01:08 trojan.data
-rw-r--r--  1 daisuke  taiwan     225000 Dec 11 23:47 trojan.list
```

8.5 Making PNG files

Make a Python script to generate PNG files to visualise positions of Jovian Trojan asteroids.

Python Code 43: ai202209_s13_06_04.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 01:51:59 (CST) daisuke>
#

# importing datetime module
import datetime

# importing pathlib module
import pathlib

# importing numpy module
import numpy

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# data file
file_data = 'trojan.data'

# directory to store PNG files
dir_png = 'trojan'

# making directory if it does not exist
p_png = pathlib.Path (dir_png)
p_png.mkdir (mode=0o755, exist_ok=True)

# counter
i = 0

# number of major bodies
```

```
n_major = 10
# number of minor bodies
n_minor = 3000

# opening data file
with open (file_data, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # if the line starts with '#', then skip
        if (line[0] == '#'):
            continue
        # splitting data
        records = line.split ('|')
        # JD
        jd = float (records[0])
        # date/time
        t = records[1]
        t_datetime = datetime.datetime.fromisoformat (t)
        YYYY = t_datetime.year

        # PNG file name
        file_png = f"{dir_png}/{i:04d}.png"

        # making objects "fig" and "canvas"
        fig    = matplotlib.figure.Figure ()
        canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

        # making objects "ax"
        ax = fig.add_subplot (121)

        # labels
        label_x = 'X [au]'
        label_y = 'Y [au]'
        label_z = 'Z [au]'
        ax.set_xlabel (label_x)
        ax.set_ylabel (label_y)

        # axes
        ax.set_xlim (-7.0, 7.0)
        ax.set_ylim (-7.0, 7.0)
        ax.set_aspect ('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-6, 6, 7))
        ax.set_yticks (numpy.linspace (-6, 6, 7))

        for j in ( range (2, n_major + n_minor + 2) ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (x, y, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (x, y, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (x, y, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
```

```
        elif (j == 5):
            earth, = ax.plot (x, y, color='green', linestyle='None', \
                              marker='o', markersize=3, label='Earth')
        elif (j == 6):
            mars, = ax.plot (x, y, color='red', linestyle='None', \
                             marker='o', markersize=2, label='Mars')
        elif (j == 7):
            jupiter, = ax.plot (x, y, color='orange', linestyle='None', \
                                marker='o', markersize=8, label='Jupiter')
        elif (j == 8):
            saturn, = ax.plot (x, y, color='brown', linestyle='None', \
                               marker='o', markersize=7, label='Saturn')
        elif (j == 9):
            uranus, = ax.plot (x, y, color='lime', linestyle='None', \
                               marker='o', markersize=6, label='Uranus')
        elif (j == 10):
            neptune, = ax.plot (x, y, color='indigo', linestyle='None', \
                                marker='o', markersize=6, label='Neptune')
        elif (j == 11):
            pluto, = ax.plot (x, y, color='slategrey', linestyle='None', \
                              marker='o', markersize=2, label='Pluto')
        else:
            asteroids, = ax.plot (x, y, color='purple', linestyle='None', \
                                  marker='.', markersize=1, \
                                  label='asteroids')

    # showing legend and title
    ax.legend (bbox_to_anchor=(0.0, -0.36), loc='upper left', \
               frameon=False, ncol=2, \
               handles=[sun, mercury, venus, earth, mars, jupiter])
    title_name = "Jovian Trojan Asteroids (Year %04d)" % (YYYY)
    ax.set_title (title_name)

    # making objects "ax"
    ax = fig.add_subplot (222)

    # labels
    label_x = 'X [au]'
    label_y = 'Y [au]'
    label_z = 'Z [au]'
    ax.set_xlabel (label_x)
    ax.set_ylabel (label_z)

    # axes
    ax.set_xlim (-7.0, 7.0)
    ax.set_ylim (-5, 5)
    ax.set_aspect('equal')
    ax.grid ()
    ax.set_xticks (numpy.linspace (-6, 6, 7))
    ax.set_yticks (numpy.linspace (-4, 4, 5))

    for j in ( range (2, n_major + n_minor + 2) ):
        xyz = records[j].split (',')
        x = float (xyz[0])
        y = float (xyz[1])
        z = float (xyz[2])
        if (j == 2):
            sun, = ax.plot (x, z, color='yellow', linestyle='None', \
                            marker='o', markersize=10, label='Sun')
```

```
            elif (j == 3):
                mercury, = ax.plot (x, z, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (x, z, color='gold', linestyle='None', \
                                   marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (x, z, color='green', linestyle='None', \
                                   marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (x, z, color='red', linestyle='None', \
                                  marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (x, z, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (x, z, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (x, z, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
                neptune, = ax.plot (x, z, color='indigo', linestyle='None', \
                                    marker='o', markersize=6, label='Neptune')
            elif (j == 11):
                pluto, = ax.plot (x, z, color='slategrey', linestyle='None',
                                  marker='o', markersize=2, label='Pluto')
            else:
                asteroids, = ax.plot (x, z, color='purple', linestyle='None',
                                      marker='.', markersize=1,
                                      label='asteroids')

        # making objects "ax"
        ax = fig.add_subplot (224)

        # labels
        label_x = 'X [au]'
        label_y = 'Y [au]'
        label_z = 'Z [au]'
        ax.set_xlabel (label_y)
        ax.set_ylabel (label_z)

        # axes
        ax.set_xlim (-7.0, 7.0)
        ax.set_ylim (-5, 5)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-6, 6, 7))
        ax.set_yticks (numpy.linspace (-4, 4, 5))

        for j in ( range (2, n_major + n_minor + 2) ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (y, z, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
```

```
            mercury, = ax.plot (y, z, color='blue', linestyle='None', \
                                marker='o', markersize=2, label='Mercury')
        elif (j == 4):
            venus, = ax.plot (y, z, color='gold', linestyle='None', \
                              marker='o', markersize=3, label='Venus')
        elif (j == 5):
            earth, = ax.plot (y, z, color='green', linestyle='None', \
                              marker='o', markersize=3, label='Earth')
        elif (j == 6):
            mars, = ax.plot (y, z, color='red', linestyle='None', \
                             marker='o', markersize=2, label='Mars')
        elif (j == 7):
            jupiter, = ax.plot (y, z, color='orange', linestyle='None', \
                                marker='o', markersize=8, label='Jupiter')
        elif (j == 8):
            saturn, = ax.plot (y, z, color='brown', linestyle='None', \
                               marker='o', markersize=7, label='Saturn')
        elif (j == 9):
            uranus, = ax.plot (y, z, color='lime', linestyle='None', \
                               marker='o', markersize=6, label='Uranus')
        elif (j == 10):
            neptune, = ax.plot (y, z, color='indigo', linestyle='None', \
                                marker='o', markersize=6, label='Neptune')
        elif (j == 11):
            pluto, = ax.plot (y, z, color='slategrey', linestyle='None', \
                              marker='o', markersize=2, label='Pluto')
        else:
            asteroids, = ax.plot (y, z, color='purple', linestyle='None', \
                                  marker='.', markersize=1, \
                                  label='asteroids')

    ax.legend (bbox_to_anchor=(-0.3, -0.4), loc='upper left', \
               frameon=False, ncol=2, \
               handles=[saturn, uranus, neptune, pluto, asteroids])
    # saving the plot into a file
    fig.tight_layout ()
    fig.savefig (file_png, dpi=225)

    # increment
    i += 1
    if (i % 100 == 0):
        print (f"status: {i}")
```

Execute above script to make PNG files.

```
% chmod a+x ai202209_s13_06_04.py
% ./ai202209_s13_06_04.py
```

Display one of PNG files. (Fig. 19)

```
% feh -dF trojan/0000.png
```

8.6 Making a MP4 movie

Use "`ffmpeg`" command to create a movie file.

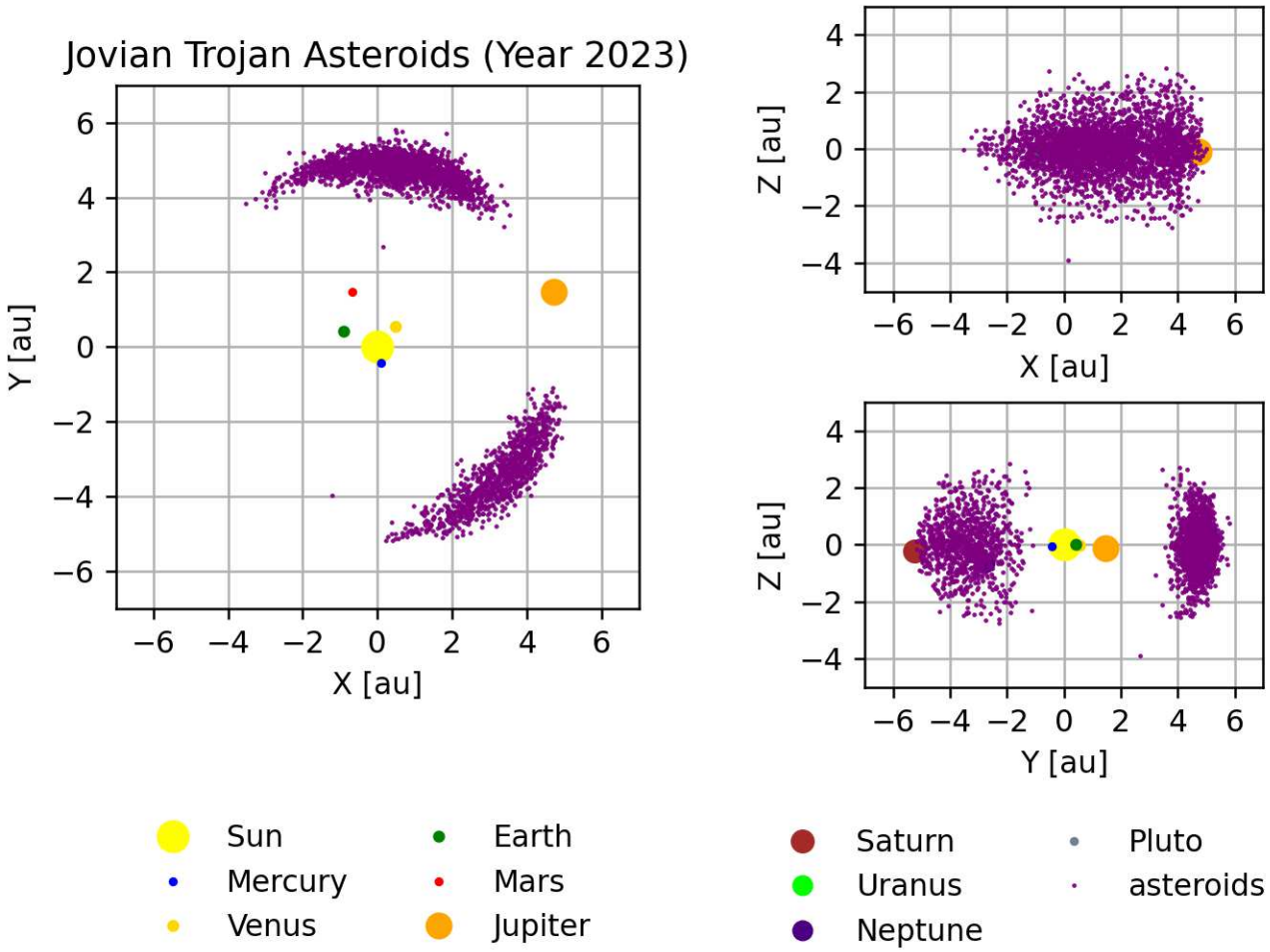

Figure 19: Positions of 3000 Jovian Trojan asteroids.

```
% ffmpeg -f image2 -start_number 0 -framerate 30 -i trojan/%04d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 6 trojan.mp4
```

Play the movie file.

```
% mplayer trojan.mp4
```

Try following practice.

Practice 13-20

Use REBOUND package to carry out orbital integration of the Sun, planets, Pluto, and Hilda asteroids. Make a MPEG-4 movie to visualise the orbital motion of these objects.

9 Main asteroid belt

Visualise orbital motion of asteroids in the mainbelt.

9.1 Reading orbital elements

Make a Python script to read orbital elements from MPCORB orbit database file.

Python Code 44: ai202209_s13_07_00.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 01:36:55 (CST) daisuke>
#

# importing gzip module
import gzip

# importing numpy module
import numpy

# MPC's orbital elements file
file_mpcorb = 'MPCORB.DAT.gz'

# number of asteroids to process
n_asteroids = 3000

# dictionary to store orbital elements
dic_elements = {}

# opening file
with gzip.open (file_mpcorb, 'rb') as fh:
    # flag
    data_line = 'NO'
    # reading file
    for line in fh:
        # decoding byte data
        line = line.decode ('utf-8')
        if (data_line == 'YES'):
            # number (or provisional designation)
            number = line[0:7]
            # epoch
```

```
            epoch = line[20:25]
            # mean anomaly
            M = float (line[26:35])
            M_rad = numpy.deg2rad (M)
            # argument of perihelion
            peri = float (line[37:46])
            peri_rad = numpy.deg2rad (peri)
            # longitude of ascending node
            node = float (line[48:57])
            node_rad = numpy.deg2rad (node)
            # inclination
            i = float (line[59:68])
            i_rad = numpy.deg2rad (i)
            # eccentricity
            e = float (line[70:79])
            # semimajor axis
            a = float (line[92:103])

            # adding data to the dictionary
            dic_elements[number] = {}
            dic_elements[number]['a'] = a
            dic_elements[number]['e'] = e
            dic_elements[number]['i'] = i_rad
            dic_elements[number]['peri'] = peri_rad
            dic_elements[number]['node'] = node_rad
            dic_elements[number]['M'] = M_rad

            # when finish reading expected number of asteroid data, then break
            if (int (number) == n_asteroids):
                break
        # if the line starts with '----------'
        if (line[:10] == '----------'):
            # set the flag to 'YES'
            data_line = 'YES'
            continue

for number in sorted (dic_elements.keys ()):
    print (f"{number:8s}", \
           f"{dic_elements[number]['a']:10.8f}", \
           f"{dic_elements[number]['e']:10.8f}", \
           f"{dic_elements[number]['i']:10.8f}", \
           f"{dic_elements[number]['peri']:10.8f}", \
           f"{dic_elements[number]['node']:10.8f}", \
           f"{dic_elements[number]['M']:10.8f}")
```

Execute above script to read orbital elements.

```
% chmod a+x ai202209_s13_07_00.py
% ./ai202209_s13_07_00.py
00001    2.76718170 0.07881750 0.18476649 1.28230126 1.40080353 0.30047047
00002    2.76963170 0.23008440 0.60959167 5.42561429 3.01799088 6.24565061
00003    2.67013680 0.25646770 0.22672996 4.32381848 2.96431921 6.14048946
00004    2.36303820 0.08875750 0.12460359 2.64590361 1.81090703 2.00945010
00005    2.57869790 0.18792370 0.09355192 6.26375054 2.47003970 4.46855130
00006    2.42524050 0.20275240 0.25721912 4.18076239 2.41969393 1.60335195
00007    2.38695030 0.22965970 0.09630378 2.53793039 4.52920620 2.69619353
00008    2.20103750 0.15639050 0.10278610 4.98482074 1.93494269 4.48382258
00009    2.38656960 0.12342860 0.09732986 0.10298123 1.20225016 5.09365319
00010    3.14018110 0.11115900 0.06687630 5.45392161 4.94230382 0.69351496
```

```
.....

02991     2.33724810 0.22135650 0.08991657 3.45854933 1.68537475 1.68837096
02992     2.74478300 0.18991800 0.12302337 4.67133273 3.22753755 3.39407425
02993     2.58754760 0.19358000 0.21468841 1.30742195 5.46738595 3.06637629
02994     2.41950430 0.22726890 0.04337824 5.61767486 6.25371036 4.02410163
02995     2.61555810 0.13761020 0.25917214 5.75941200 2.96049031 0.96823571
02996     2.78153610 0.03081910 0.06390994 5.26844355 5.83428121 6.09221592
02997     2.55590400 0.19667320 0.12534885 6.11199731 6.19604294 4.96237354
02998     2.42323850 0.19618460 0.05374829 3.57144177 2.74715365 3.76545222
02999     2.27058070 0.10585860 0.11805878 0.69080149 1.30729385 2.25041462
03000     2.35123210 0.18135350 0.04805153 3.02825970 3.50733425 0.37203386
```

9.2 Making a simulation file for REBOUND

Make a Python script to create a simulation file for REBOUND.

Python Code 45: ai202209_s13_07_01.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 01:43:47 (CST) daisuke>
#

# importing gzip module
import gzip

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing rebound module
import rebound

# importing ssl module
import ssl

# allow insecure downloading
ssl._create_default_https_context = ssl._create_unverified_context

# MPC's orbital elements file
file_mpcorb = 'MPCORB.DAT.gz'

# simulation file to be generated
file_sim = 'iss.bin'

majorbody = {
    'Sun':     '10',
    'Mercury': '1',
    'Venus':   '2',
    'Earth':   '3',
    'Mars':    '4',
    'Jupiter': '5',
    'Saturn':  '6',
    'Uranus':  '7',
```

```
    'Neptune': '8',
    'Pluto':   '9',
}

# number of asteroids to process
n_asteroids = 3000

# epoch of orbital elements
# K232P => 2023-Feb-25
# if the epoch is not K232P, then change following line
t_epoch = '2023-02-25 00:00'

# construction of a simulation
sim = rebound.Simulation ()

# adding major bodies
for name in majorbody.keys ():
    sim.add (majorbody[name], date=t_epoch)

# dictionary to store orbital elements
dic_elements = {}

# opening file
with gzip.open (file_mpcorb, 'rb') as fh:
    # flag
    data_line = 'NO'
    # reading file
    for line in fh:
        # decoding byte data
        line = line.decode ('utf-8')
        if (data_line == 'YES'):
            # number (or provisional designation)
            number = line[0:7]
            # epoch
            epoch = line[20:25]
            # mean anomaly
            M = float (line[26:35])
            M_rad = numpy.deg2rad (M)
            # argument of perihelion
            peri = float (line[37:46])
            peri_rad = numpy.deg2rad (peri)
            # longitude of ascending node
            node = float (line[48:57])
            node_rad = numpy.deg2rad (node)
            # inclination
            i = float (line[59:68])
            i_rad = numpy.deg2rad (i)
            # eccentricity
            e = float (line[70:79])
            # semimajor axis
            a = float (line[92:103])

            # adding data to the dictionary
            dic_elements[number] = {}
            dic_elements[number]['a'] = a
            dic_elements[number]['e'] = e
            dic_elements[number]['i'] = i_rad
            dic_elements[number]['peri'] = peri_rad
            dic_elements[number]['node'] = node_rad
```

```
            dic_elements[number]['M'] = M_rad

            # when finish reading expected number of asteroid data, then break
            if (int (number) == n_asteroids):
                break
        # if the line starts with '----------'
        if (line[:10] == '----------'):
            # set the flag to 'YES'
            data_line = 'YES'
            continue

# processing each asteroid orbit
for number in sorted (dic_elements.keys ()):
    # adding an asteroid
    sim.add (m=0.0, \
             a=dic_elements[number]['a'], \
             e=dic_elements[number]['e'], \
             inc=dic_elements[number]['i'], \
             omega=dic_elements[number]['peri'], \
             Omega=dic_elements[number]['node'], \
             M=dic_elements[number]['M'], \
             date=t_epoch)

# printing simulation object
print (sim)

# saving simulation to a file
sim.save (file_sim)
```

Execute above script to make a simulation file.

```
% chmod a+x ai202209_s13_07_01.py
% ./ai202209_s13_07_01.py
Searching NASA Horizons for '10'...
Found: Sun (10)
Searching NASA Horizons for '1'...
Found: Mercury Barycenter (199)
Searching NASA Horizons for '2'...
Found: Venus Barycenter (299)
Searching NASA Horizons for '3'...
Found: Earth-Moon Barycenter (3)
Searching NASA Horizons for '4'...
Found: Mars Barycenter (4)
Searching NASA Horizons for '5'...
Found: Jupiter Barycenter (5)
Searching NASA Horizons for '6'...
Found: Saturn Barycenter (6)
Searching NASA Horizons for '7'...
Found: Uranus Barycenter (7)
Searching NASA Horizons for '8'...
Found: Neptune Barycenter (8)
Searching NASA Horizons for '9'...
Found: Pluto Barycenter (9)
<rebound.simulation.Simulation object at 0x79038aafa290, N=3010, t=0.0>
% ls -lF *.bin
-rw-r--r--  1 daisuke  taiwan    4880 Dec 11 22:18 comets.bin
-rw-r--r--  1 daisuke  taiwan  387984 Dec 12 01:44 iss.bin
-rw-r--r--  1 daisuke  taiwan    2960 Dec 11 20:02 simple0.bin
-rw-r--r--  1 daisuke  taiwan    2960 Dec 11 21:27 simple1.bin
```

```
-rw-r--r--  1 daisuke  taiwan  387984 Dec 11 23:54 trojan.bin
```

9.3 Orbital integration

Make a Python script to carry out orbital integration of mainbelt asteroids.

Python Code 46: ai202209_s13_07_02.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 01:52:29 (CST) daisuke>
#

# importing datetime module
import datetime

# importing numpy module
import numpy

# importing astropy module
import astropy.time

# importing rebound module
import rebound

# date/time now
now = datetime.datetime.now ()

# units
u_day = astropy.units.day

# simulation file
file_sim = 'iss.bin'

# output file
file_output = 'iss.data'

# parameters
year       = 2.0 * numpy.pi
time_epoch = '2023-02-25T00:00:00'
t_interval = 0.1 # 0.1 ==> 365.25/(2.0*pi) * 0.1 = 5.81 days
n_output   = 1000

# objects
objects = [
    'Sun',
    'Mercury',
    'Venus',
    'Earth',
    'Mars',
    'Jupiter',
    'Saturn',
    'Uranus',
    'Neptune',
    'Pluto',
    ]

# number of asteroids
```

```
n_asteroids = 3000

# reading simulation from file
sim = rebound.Simulation (file_sim)
sim.integrator = 'mercurius'
sim.dt = +0.01
sim.move_to_com ()

# particles
ps = sim.particles

# opening file for writing
with open (file_output, 'w') as fh:
    # writing header
    fh.write (f"#\n")
    fh.write (f"# results of orbital integration using rebound\n")
    fh.write (f"#\n")
    fh.write (f"#   start of integration: {now}\n")
    fh.write (f"#\n")
    fh.write (f"#   list of objects:\n")
    for name in objects:
        fh.write (f"#     {name}\n")
    fh.write (f"#     and {n_asteroids} minor bodies\n")
    fh.write (f"#\n")
    fh.write (f"#   format of the data:\n")
    fh.write (f"#     JD, date/time, x,y,z,vx,vy,vz of obj1, ")
    fh.write (f"x,y,z,vx,vy,vz of obj2, ...\n")
    fh.write (f"#\n")

    # epoch
    t_epoch = astropy.time.Time (time_epoch, scale='utc', format='isot')

    # orbital integration
    for i in range (n_output):
        # target time
        time = t_interval * i
        # integration
        sim.integrate (time)
        # time after a step of integration
        t_current = t_epoch + 365.25 * sim.t / year * u_day
        jd_current = t_current.jd

        # writing data to file
        fh.write (f"{jd_current:.8f}|{t_current}")
        for j in range ( len (objects) + n_asteroids):
            fh.write (f"|{ps[j].x:+.15f},{ps[j].y:+.15f},{ps[j].z:+.15f},")
            fh.write (f"{ps[j].vx:+.15f},{ps[j].vy:+.15f},{ps[j].vz:+.15f}")
        fh.write (f"\n")

        # printing status
        if ( (i + 1) % 100 == 0 ):
            print (f"  status: {i + 1:08d} / {n_output:08d}")
```

Execute above script to carry out orbital integration.

```
% chmod a+x ai202209_s13_07_02.py
% ./ai202209_s13_07_02.py
  status: 00000100 / 00001000
  status: 00000200 / 00001000
```

```
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "taiutc" yielded 1 of "dubious year (Note 4)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/usr/pkg/lib/python3.9/site-packages/erfa/core.py:154: ErfaWarning: ERFA function
 "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
  status: 00000300 / 00001000
  status: 00000400 / 00001000
  status: 00000500 / 00001000
  status: 00000600 / 00001000
  status: 00000700 / 00001000
  status: 00000800 / 00001000
  status: 00000900 / 00001000
  status: 00001000 / 00001000
% ls -lF iss.*
-rw-r--r--  1 daisuke  taiwan     387984 Dec 12 01:44 iss.bin
-rw-r--r--  1 daisuke  taiwan  343187112 Dec 12 03:01 iss.data
% head -30 iss.data | cut -b 1-80
#
# results of orbital integration using rebound
#
#   start of integration: 2022-12-12 01:52:32.915370
#
#   list of objects:
#     Sun
#     Mercury
#     Venus
#     Earth
#     Mars
#     Jupiter
#     Saturn
#     Uranus
#     Neptune
#     Pluto
#     and 3000 minor bodies
#
#   format of the data:
#     JD, date/time, x,y,z,vx,vy,vz of obj1, x,y,z,vx,vy,vz of obj2, ...
#
2460000.50000000|2023-02-25T00:00:00.000|-0.008983414248625,-0.000395111524420,+
2460006.31313430|2023-03-02T19:30:54.803|-0.008973743033079,-0.000446297824952,+
2460012.12626859|2023-03-08T15:01:49.606|-0.008963674819858,-0.000497337397762,+
2460017.93940289|2023-03-14T10:32:44.410|-0.008953215055873,-0.000548224900502,+
2460023.75253719|2023-03-20T06:03:39.213|-0.008942369140748,-0.000598954433558,+
2460029.56567148|2023-03-26T01:34:34.016|-0.008931143567704,-0.000649518517813,+
2460035.37880578|2023-03-31T21:05:28.819|-0.008919548892696,-0.000699908626156,+
2460041.19194008|2023-04-06T16:36:23.622|-0.008907601570201,-0.000750120011792,+
2460047.00507437|2023-04-12T12:07:18.426|-0.008895319089160,-0.000800156570092,+
```

9.4 Making PNG files

Make a Python script to generate PNG files to visualise positions of mainbelt asteroids.

Python Code 47: ai202209_s13_07_03.py

```
#!/usr/pkg/bin/python3.9

#
# Time-stamp: <2022/12/12 03:15:13 (CST) daisuke>
```

```
#

# importing datetime module
import datetime

# importing pathlib module
import pathlib

# importing numpy module
import numpy

# importing matplotlib module
import matplotlib.figure
import matplotlib.backends.backend_agg

# data file
file_data = 'iss.data'

# directory to store PNG files
dir_png = 'iss'

# making directory if it does not exist
p_png = pathlib.Path (dir_png)
p_png.mkdir (mode=0o755, exist_ok=True)

# counter
i = 0

# number of major bodies
n_major = 10
# number of minor bodies
n_minor = 3000

# opening data file
with open (file_data, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # if the line starts with '#', then skip
        if (line[0] == '#'):
            continue
        # splitting data
        records = line.split ('|')
        # JD
        jd = float (records[0])
        # date/time
        t = records[1]
        t_datetime = datetime.datetime.fromisoformat (t)
        YYYY = t_datetime.year

        # PNG file name
        file_png = "%s/%04d.png" % (dir_png, i)

        # making objects "fig" and "canvas"
        fig    = matplotlib.figure.Figure ()
        canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

        # making object "ax"
        ax = fig.add_subplot (121)
```

```
        # labels
        label_x = 'X [au]'
        label_y = 'Y [au]'
        label_z = 'Z [au]'
        ax.set_xlabel (label_x)
        ax.set_ylabel (label_y)

        # axes
        ax.set_xlim (-6.0, 6.0)
        ax.set_ylim (-6.0, 6.0)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-5, 5, 11))
        ax.set_yticks (numpy.linspace (-5, 5, 11))

        for j in (
                list ( range (2 + n_major, len (records) ) )
                + list ( range (2, n_major + 2) )
        ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (x, y, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (x, y, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (x, y, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (x, y, color='green', linestyle='None', \
                                  marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (x, y, color='red', linestyle='None', \
                                 marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (x, y, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (x, y, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (x, y, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
                neptune, = ax.plot (x, y, color='indigo', linestyle='None', \
                                    marker='o', markersize=6, label='Neptune')
            elif (j == 11):
                pluto, = ax.plot (x, y, color='slategrey', linestyle='None',
                                  marker='o', markersize=2, label='Pluto')
            else:
                asteroids, = ax.plot (x, y, color='purple', linestyle='None',
                                      marker='.', markersize=1,
                                      label='asteroids')

        # showing legend and title
```

```
        ax.legend (bbox_to_anchor=(0.0, -0.27), loc='upper left', \
                   frameon=False, ncol=2, \
                   handles=[sun, mercury, venus, earth, mars, jupiter])
        title_name = "Inner Solar System (Year %04d)" % (YYYY)
        ax.set_title (title_name)

        # making object "ax"
        ax = fig.add_subplot (222)

        # labels
        label_x = 'X [au]'
        label_y = 'Y [au]'
        label_z = 'Z [au]'
        ax.set_xlabel (label_x)
        ax.set_ylabel (label_z)

        # axes
        ax.set_xlim (-6.0, 6.0)
        ax.set_ylim (-2.5, 2.5)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-5, 5, 11))
        ax.set_yticks (numpy.linspace (-2, 2, 5))

        for j in (
                list ( range (2 + n_major, len (records) ) )
                + list ( range (2, n_major + 2) )
        ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (x, z, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (x, z, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (x, z, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (x, z, color='green', linestyle='None', \
                                  marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (x, z, color='red', linestyle='None', \
                                 marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (x, z, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (x, z, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (x, z, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
                neptune, = ax.plot (x, z, color='indigo', linestyle='None', \
                                    marker='o', markersize=6, label='Neptune')
```

```
            elif (j == 11):
                pluto, = ax.plot (x, z, color='slategrey', linestyle='None',
                                  marker='o', markersize=2, label='Pluto')
            else:
                asteroids, = ax.plot (x, z, color='purple', linestyle='None',
                                      marker='.', markersize=1,
                                      label='asteroids')

        # making object "ax"
        ax = fig.add_subplot (224)

        # labels
        label_x = 'X [au]'
        label_y = 'Y [au]'
        label_z = 'Z [au]'
        ax.set_xlabel (label_y)
        ax.set_ylabel (label_z)

        # axes
        ax.set_xlim (-6.0, 6.0)
        ax.set_ylim (-2.5, 2.5)
        ax.set_aspect('equal')
        ax.grid ()
        ax.set_xticks (numpy.linspace (-5, 5, 11))
        ax.set_yticks (numpy.linspace (-2, 2, 5))

        for j in (
                list ( range (2 + n_major, len (records) ) )
                + list ( range (2, n_major + 2) )
        ):
            xyz = records[j].split (',')
            x = float (xyz[0])
            y = float (xyz[1])
            z = float (xyz[2])
            if (j == 2):
                sun, = ax.plot (y, z, color='yellow', linestyle='None', \
                                marker='o', markersize=10, label='Sun')
            elif (j == 3):
                mercury, = ax.plot (y, z, color='blue', linestyle='None', \
                                    marker='o', markersize=2, label='Mercury')
            elif (j == 4):
                venus, = ax.plot (y, z, color='gold', linestyle='None', \
                                  marker='o', markersize=3, label='Venus')
            elif (j == 5):
                earth, = ax.plot (y, z, color='green', linestyle='None', \
                                  marker='o', markersize=3, label='Earth')
            elif (j == 6):
                mars, = ax.plot (y, z, color='red', linestyle='None', \
                                 marker='o', markersize=2, label='Mars')
            elif (j == 7):
                jupiter, = ax.plot (y, z, color='orange', linestyle='None', \
                                    marker='o', markersize=8, label='Jupiter')
            elif (j == 8):
                saturn, = ax.plot (y, z, color='brown', linestyle='None', \
                                   marker='o', markersize=7, label='Saturn')
            elif (j == 9):
                uranus, = ax.plot (y, z, color='lime', linestyle='None', \
                                   marker='o', markersize=6, label='Uranus')
            elif (j == 10):
```

```
            neptune, = ax.plot (y, z, color='indigo', linestyle='None', \
                                marker='o', markersize=6, label='Neptune')
        elif (j == 11):
            pluto, = ax.plot (y, z, color='slategrey', linestyle='None',
                              marker='o', markersize=2, label='Pluto')
        else:
            asteroids, = ax.plot (y, z, color='purple', linestyle='None',
                                  marker='.', markersize=1,
                                  label='asteroids')

    ax.legend (bbox_to_anchor=(-0.2, -0.4), loc='upper left', \
               frameon=False, ncol=2, \
               handles=[saturn, uranus, neptune, pluto, asteroids])
    # saving the plot into a file
    fig.tight_layout ()
    fig.savefig (file_png, dpi=225)

    # increment
    i += 1
    print (f"status: {i}")
```

Execute above script to make PNG files.

```
% chmod a+x ai202209_s13_07_03.py
% ./ai202209_s13_07_03.py
```

Display one of PNG files. (Fig. 20)

```
% feh -dF iss/0000.png
```

9.5 Making a MP4 movie

Use "`ffmpeg`" command to create a movie file.

```
% ffmpeg -f image2 -start_number 0 -framerate 30 -i iss/%04d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 6 iss.mp4
```

Play the movie file.

```
% mplayer iss.mp4
```

Try following practice.

Practice 13-21

Use REBOUND package to carry out orbital integration of the Sun, planets, Pluto, and 10,000 asteroids. Make a MPEG-4 movie to visualise the orbital motion of these objects.

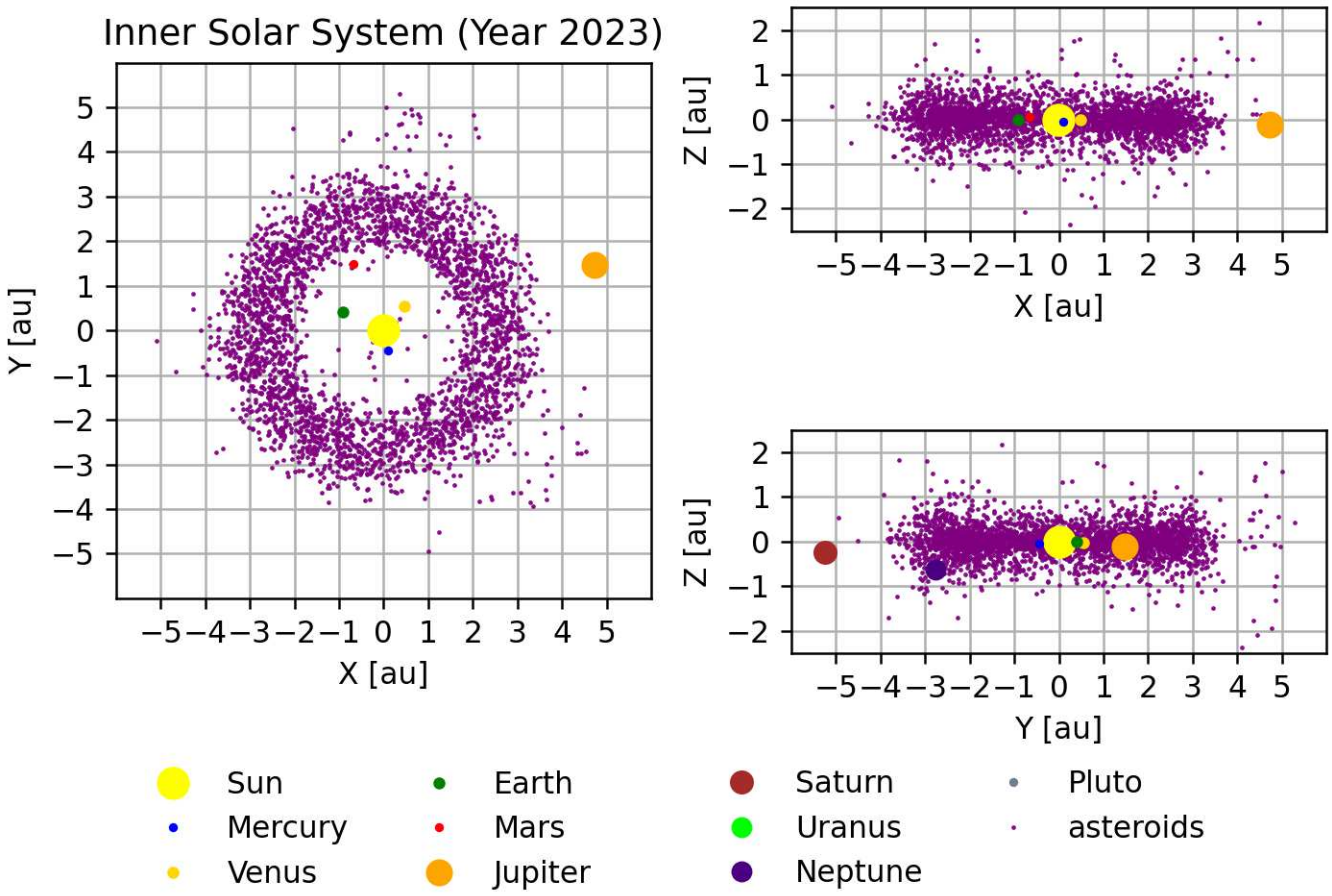

Figure 20: Positions of 3000 mainbelt asteroids.

10 For your further reading

Read following document to learn more about NASA/JPL Horizons System, Astroquery, and REBOUND package.

- NASA/JPL Horizons System: `https://ssd.jpl.nasa.gov/horizons/`
  - Manual: `https://ssd.jpl.nasa.gov/horizons/manual.html`
  - Tutorial: `https://ssd.jpl.nasa.gov/horizons/tutorial.html`
- Astroquery: `https://astroquery.readthedocs.io/`
  - JPL Horizons Queries: `https://astroquery.readthedocs.io/en/latest/jplhorizons/jplhorizons.html`
- REBOUND: `https://rebound.readthedocs.io/`
  - Quick-start guide: `https://rebound.readthedocs.io/en/latest/quickstart_installation/`
  - REBOUND API: `https://rebound.readthedocs.io/en/latest/api/`
  - Examples in REBOUND: `https://rebound.readthedocs.io/en/latest/examples/`

11 Assignment

1. What is the distance between Earth and Jupiter at 12:00:00 (UT) on 01/Jan/2023? Obtain state vectors of Earth and Jupiter from JPL Horizons and find the distance. Show your Python code.
2. Which planet has the greatest distance from the Ecliptic plane at 12:00:00 (UT) on 01/Jul/2023? What is the distance from the Ecliptic plane? Obtain state vectors of the Sun and planets from JPL Horizons, and find the distance. Show your Python code.
3. Obtain state vectors of the Sun, planets, and your favourite asteroid from 2000 to 2030 from JPL Horizons System. Make a MPEG-4 movie of motions of those bodies on X-Y plane. Show your Python code. Give a short description about your favourite asteroid.
4. Obtain state vectors of the Sun, planets, and your favourite comet from 2000 to 2030 from JPL Horizons System. Make a MPEG-4 movie of motions of those bodies on X-Y plane. Show your Python code. Give a short description about your favourite comet.
5. Show the trajectory of an interstellar comet 2I/Borisov. Show the Python code you made.
6. Carry out orbital integration of the Sun, 8 planets, and Pluto for 1000 years from year 2000 using REBOUND. Visualise the distance between Neptune and Pluto as a function of time.