Astroinformatics 2022
Session 04: Using Matplotlib

Kinoshita Daisuke

03 October 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 make various types of plots using Matplotlib.

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) "`s04`". (Fig. 3) Choose the file "`ai202209_s04.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

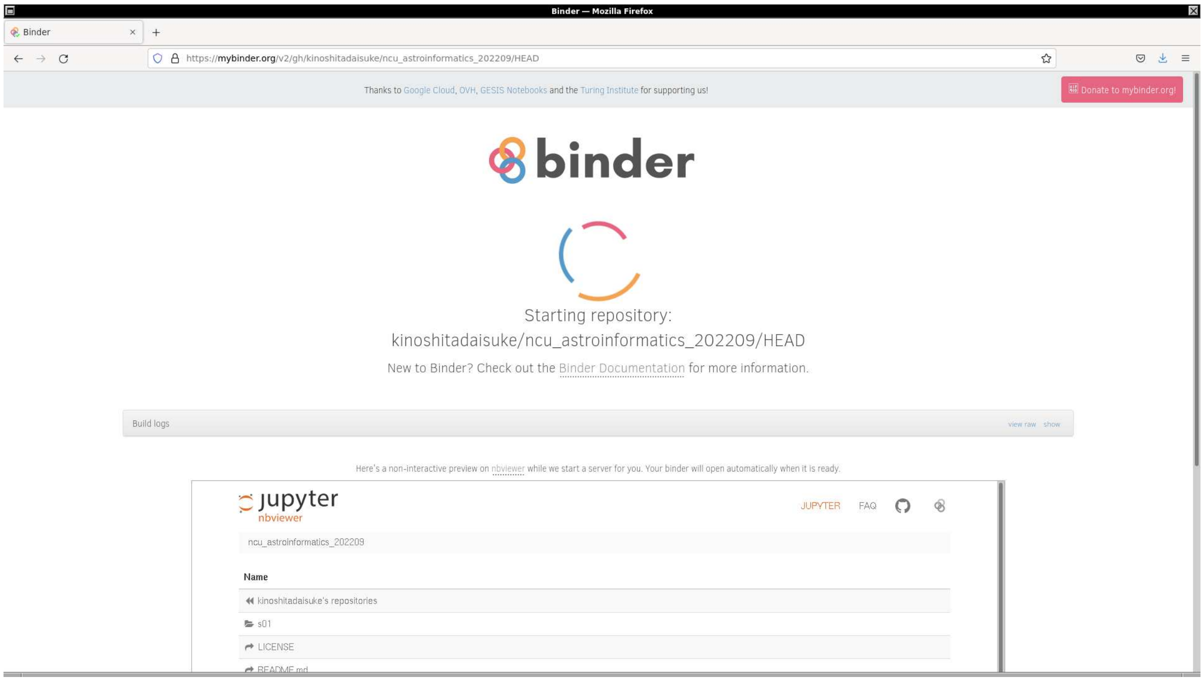


Figure 1: Using Binder to execute sample Python scripts for this session.

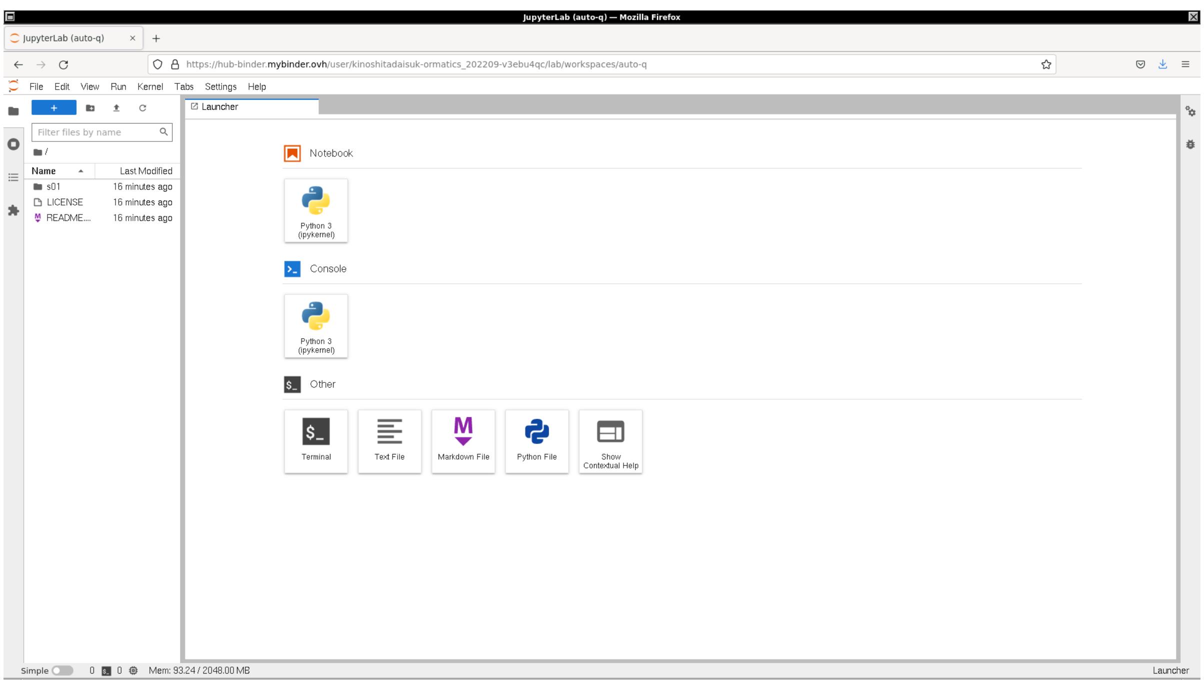


Figure 2: Using Binder to execute sample Python scripts for this session.

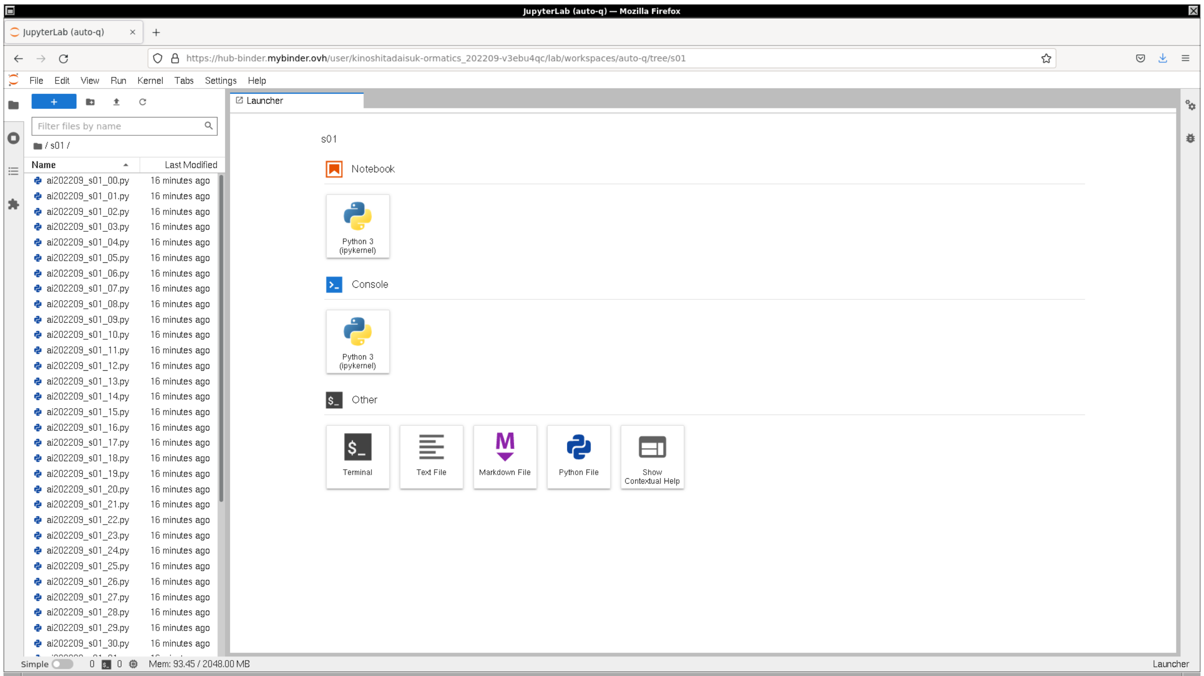


Figure 3: Using Binder to execute sample Python scripts for this session.

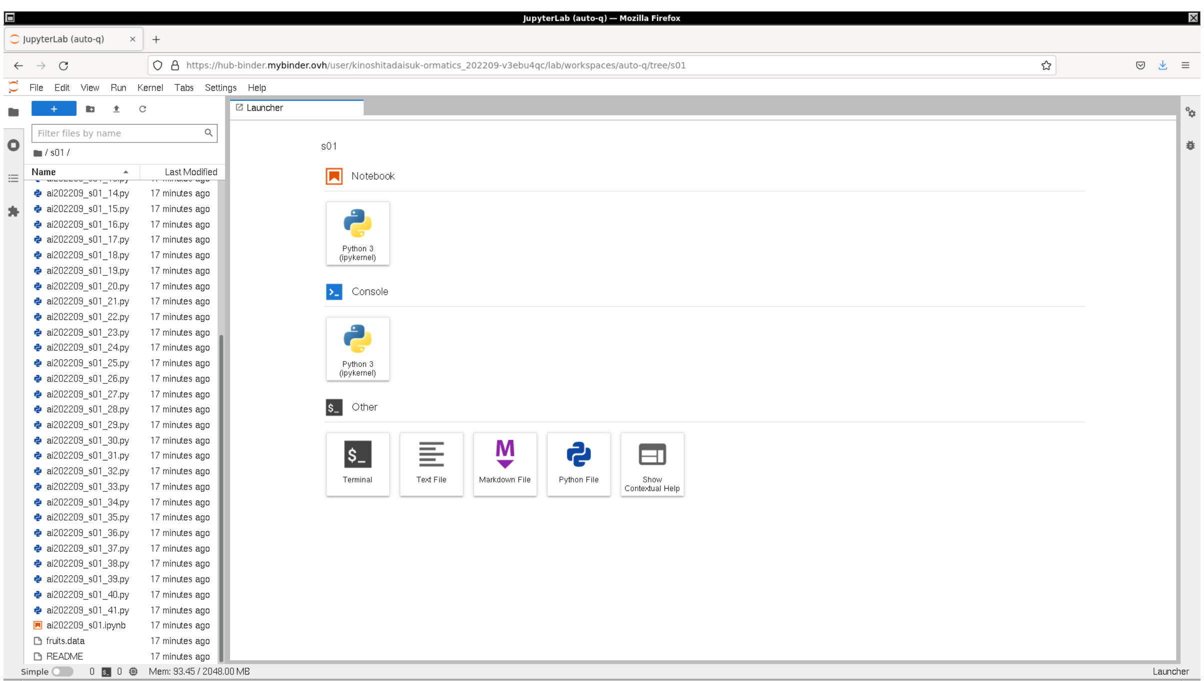


Figure 4: Using Binder to execute sample Python scripts for this session.

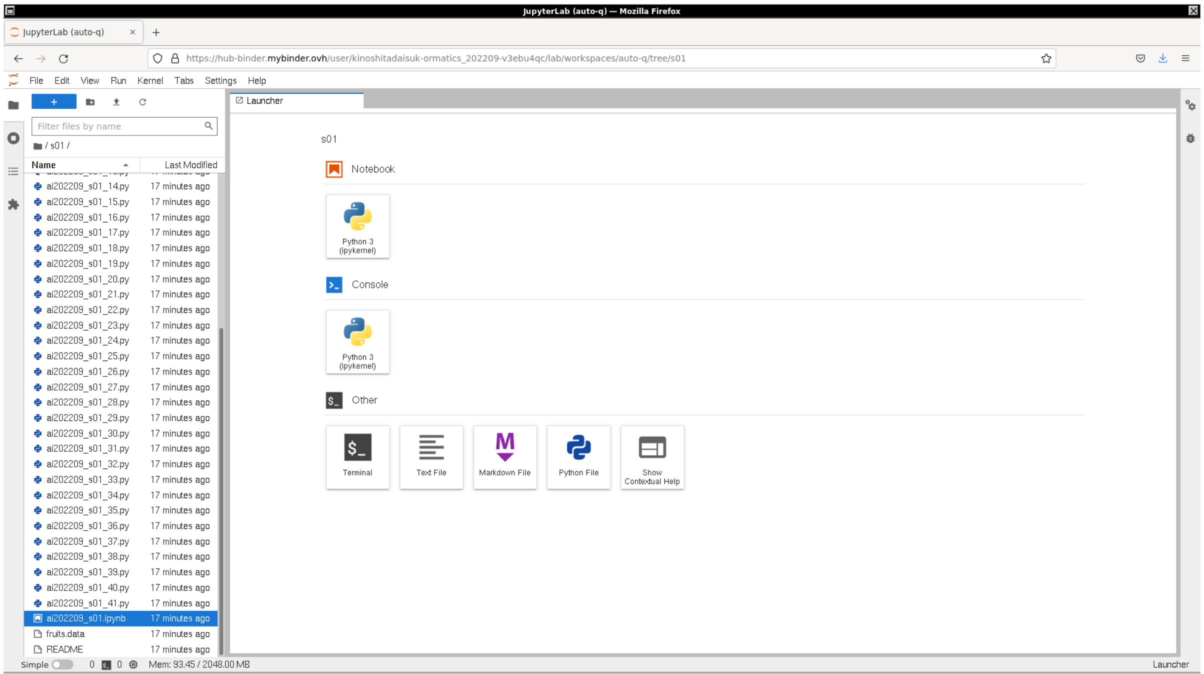


Figure 5: Using Binder to execute sample Python scripts for this session.

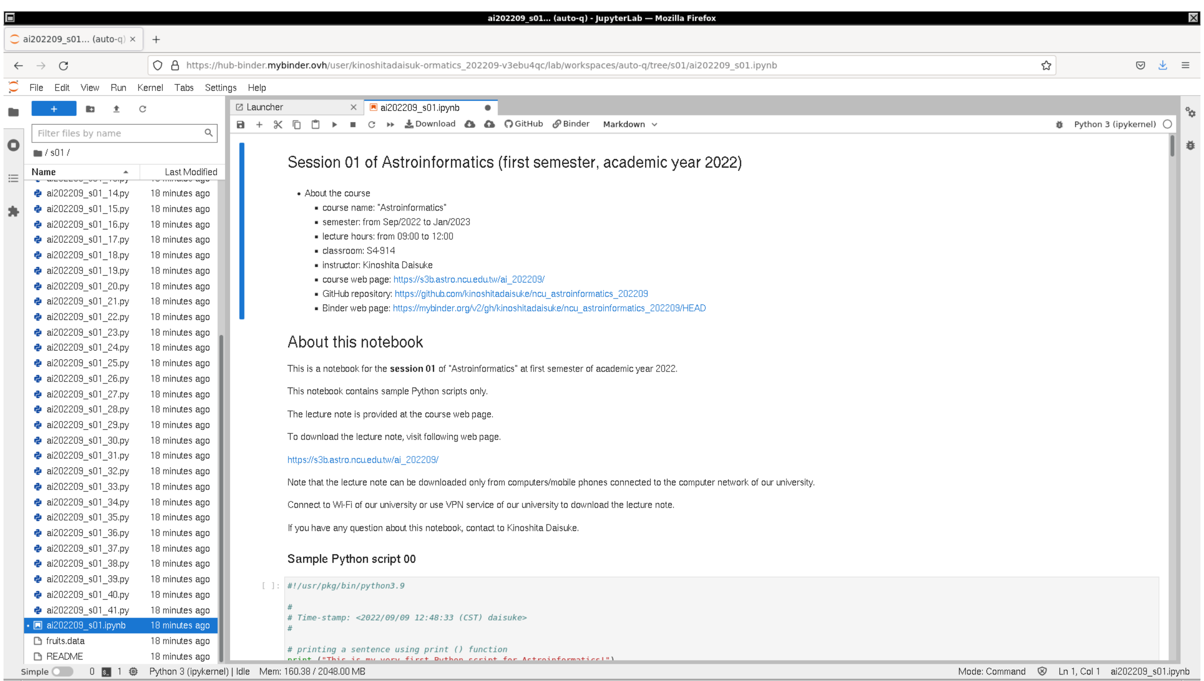


Figure 6: Using Binder to execute sample Python scripts for this session.

2 About Matplotlib

Matplotlib is the de facto standard library for high quality scientific visualisation. Various types of publication quality plots can be created by using Matplotlib. In addition to static plots, animations and interactive plots can also be created by Matplotlib. Visit the official website of Matplotlib to learn about it. (Fig. 7, 8, 9, 10, 11, 12, 13, and 14)

- Matplotlib: `https://matplotlib.org/`
  - Documentation for version 3.6: `https://matplotlib.org/stable/`
  - Cheatsheets and handouts: `https://matplotlib.org/cheatsheets/`
  - Examples: `https://matplotlib.org/stable/gallery/`
  - Tutorials: `https://matplotlib.org/stable/tutorials/`
  - Users guide: `https://matplotlib.org/stable/users/`

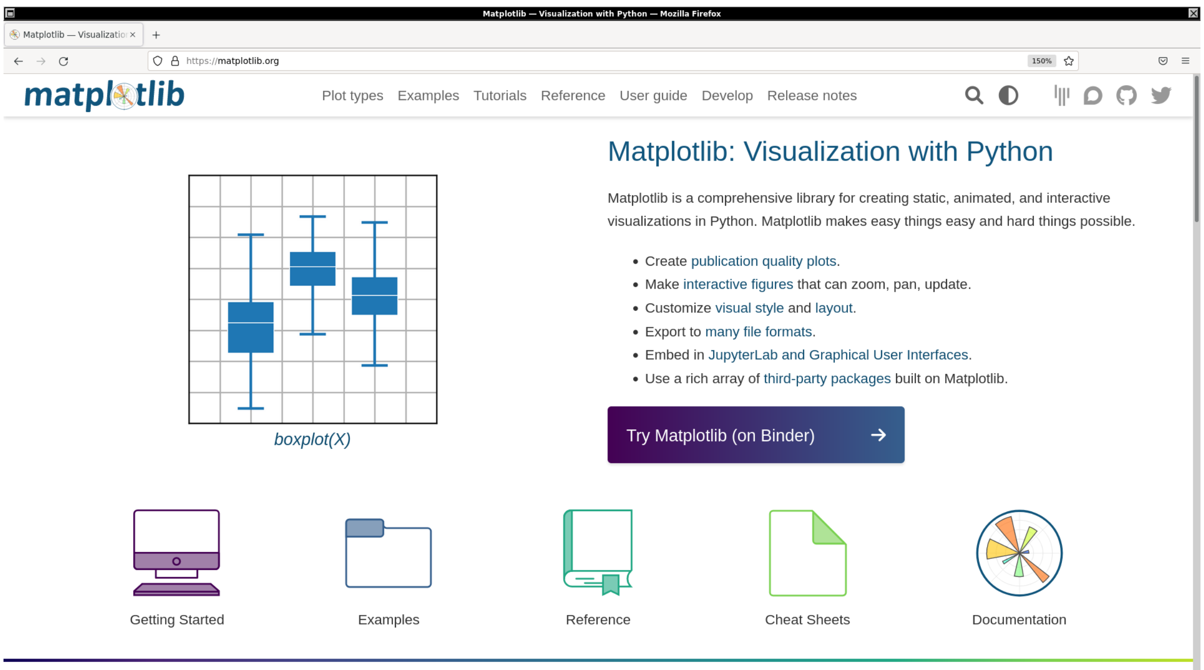


Figure 7: The official website of Matplotlib.

3 Matplotlib APIs

Matplotlib offers several different APIs (Application Programming Interfaces) for making plots. Two major APIs are "implicit `pyplot` interface" and "explicit Axes interface" (or "object-oriented interface").

3.1 Making a plot using implicit pyplot interface

Here is an example of making a plot using implicit pyplot interface.

Python Code 1: ai202209_s04_00.py

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

#
# Time-stamp: <2022/10/01 13:09:20 (CST) daisuke>
#

# importing matplotlib module
```

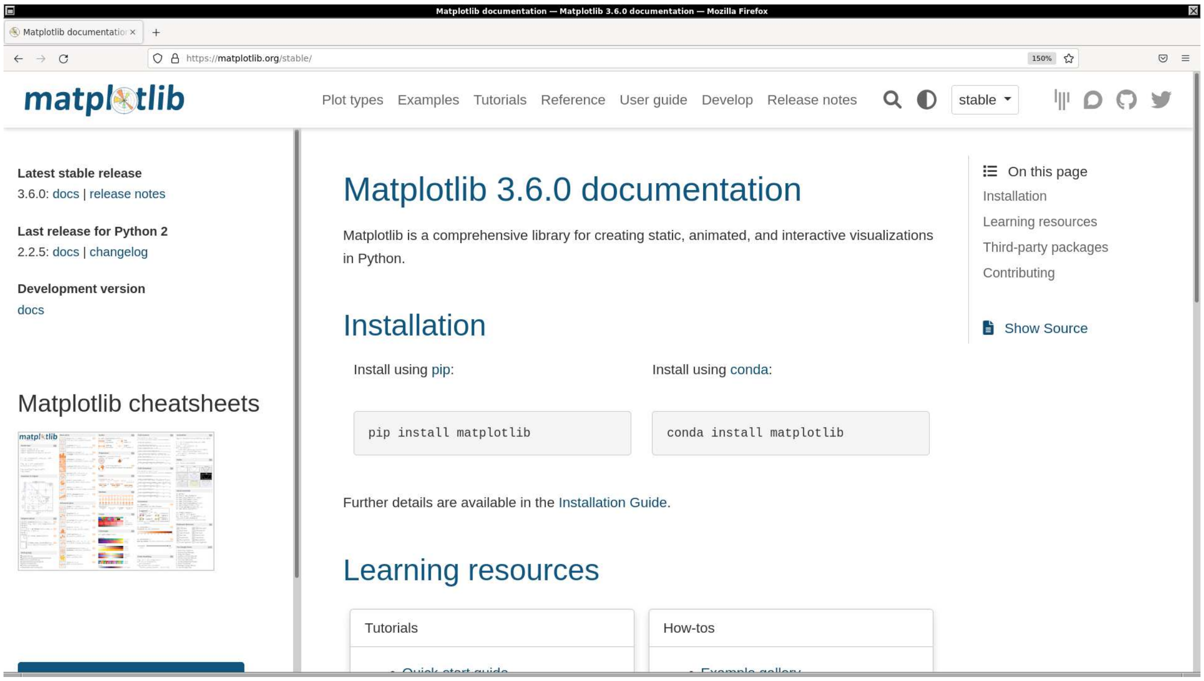


Figure 8: The documentation page of the official website of Matplotlib.

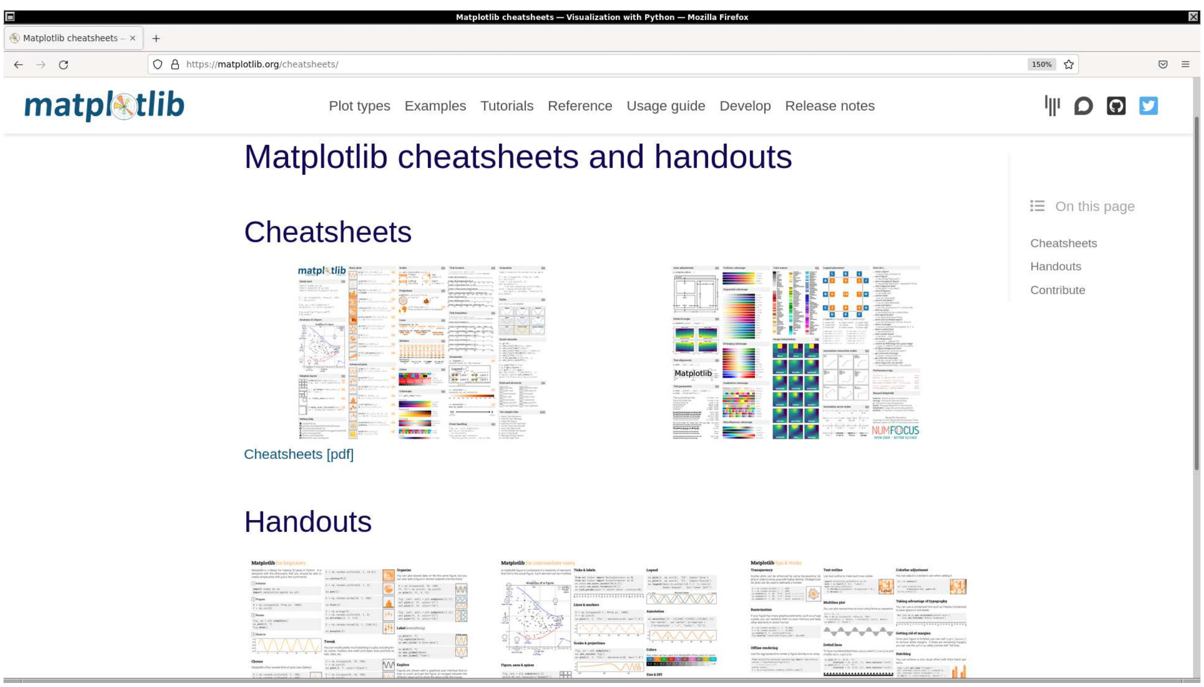


Figure 9: The cheatsheets and handouts of Matplotlib.

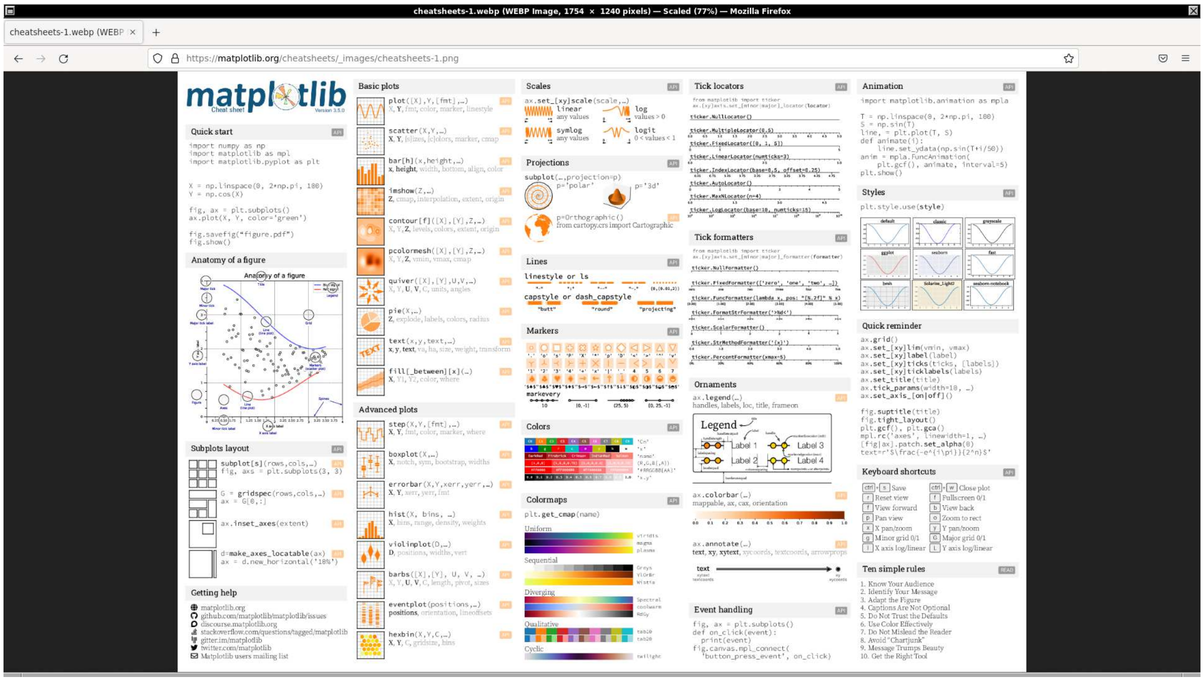


Figure 10: The cheatsheet of Matplotlib.

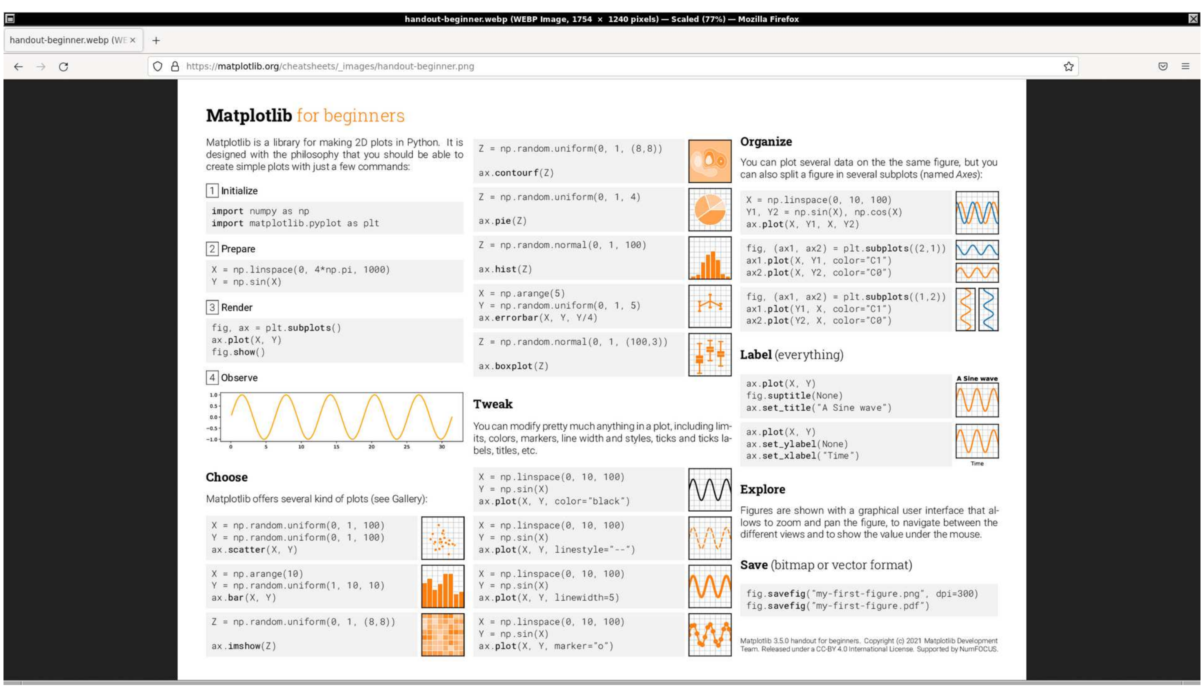


Figure 11: The handout for beginners of Matplotlib.

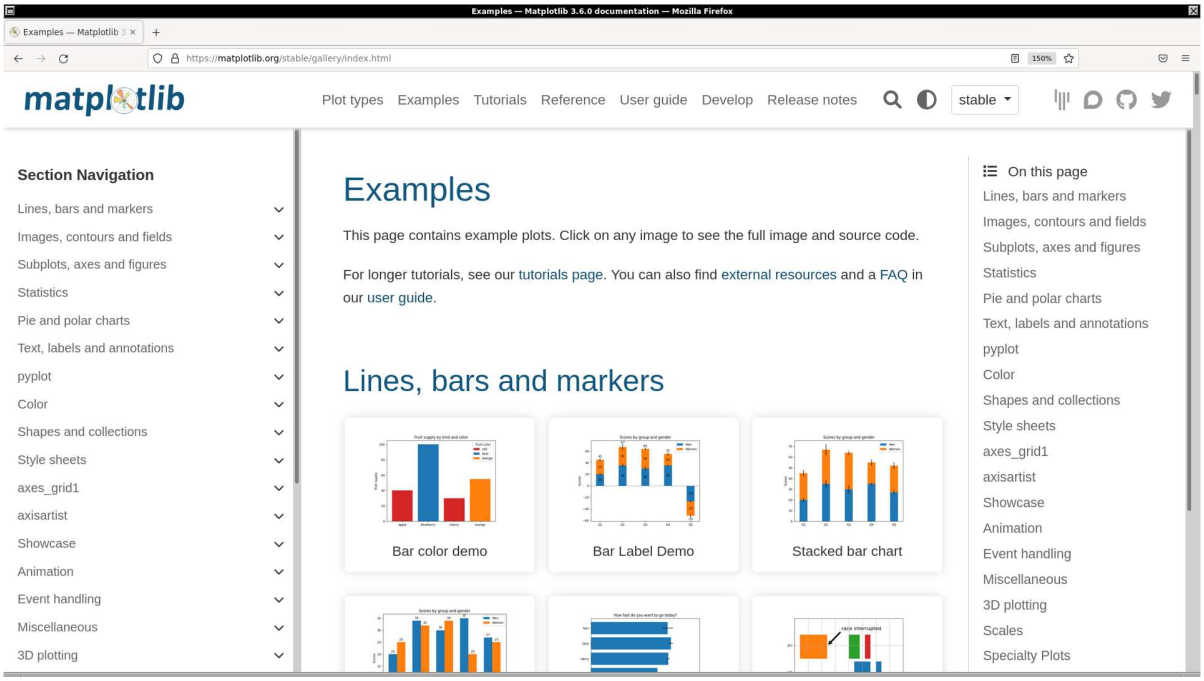


Figure 12: The examples page of the official website of Matplotlib.

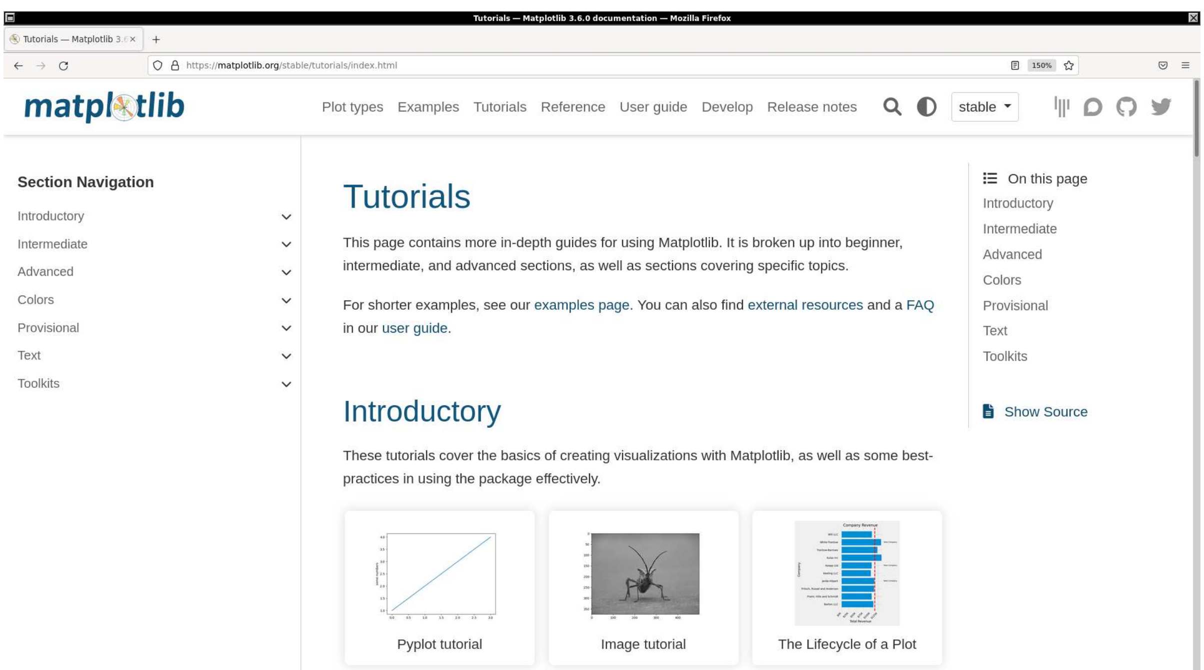


Figure 13: The tutorials page of the official website of Matplotlib.

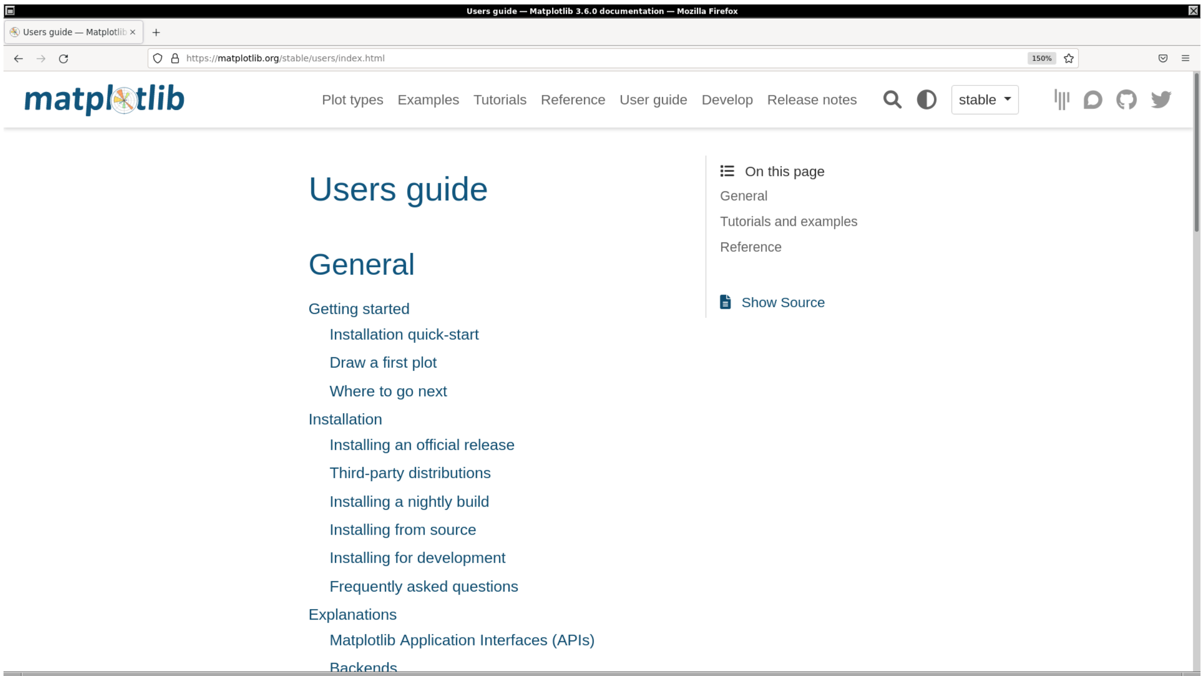


Figure 14: The users guide of the official website of Matplotlib.

```
import matplotlib.pyplot

# data to be plotted
data_x = [1.0, 2.0, 3.0, 4.0, 5.0]
data_y = [3.0, 2.0, 5.0, 1.0, 4.0]

# output file name
file_output = 'ai202209_s04_00.png'

#
# for making a plot using implicit pyplot interface, we call some functions
#

# making a plot using procedural pyplot interface
matplotlib.pyplot.plot (data_x, data_y, label="Sample data")

# adding legend to the plot
matplotlib.pyplot.legend ()

# saving a plot as a file
matplotlib.pyplot.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_00.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 12:27 ai202209_s04_00.png
```

Display the PNG image file. (15)

```
% feh -dF ai202209_s04_00.png
```

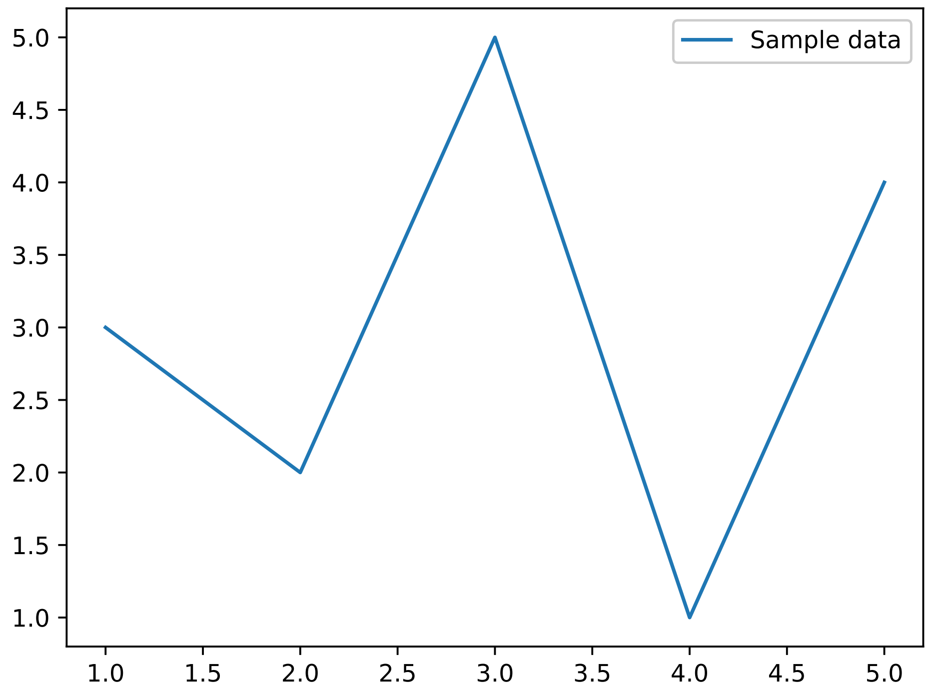


Figure 15: Plot created by the script `ai202209_s04_00.py`.

Try following practice.

Practice 04-01

Make your own Python script to create a plot using implicit pyplot interface of Matplotlib.

3.2 Making a plot using explicit Axes interface

Here is an example of making a plot using explicit Axes interface.

Python Code 2: ai202209_s04_01.py

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

#
# Time-stamp: <2022/10/01 13:09:37 (CST) daisuke>
#

# importing matplotlib module
import matplotlib.pyplot

# data to be plotted
data_x = [1.0, 2.0, 3.0, 4.0, 5.0]
data_y = [3.0, 2.0, 5.0, 1.0, 4.0]

# output file name
file_output = 'ai202209_s04_01.png'

#
# for making a plot using object-oriented interface,
```

```
# we first construct "fig" and "axes" objects,
# and then use methods for these "fig" and "axes".
#

# making a fig object using matplot.pyplot.figure function
fig = matplotlib.pyplot.figure ()

# constructing an axes object using object-oriented interface
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='Sample data')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_01.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 12:50 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:01 ai202209_s04_01.png
```

Display the PNG image file. (16)

```
% feh -dF ai202209_s04_01.png
```

Try following practice.

Practice 04-02

Make your own Python script to create a plot using explicit Axes interface of Matplotlib. (Note: Construct "`fig`" object using the function "`matplotlib.pyplot.figure ()`", and then construct "`axes`" object using "`.add_subplot ()`" method.)

Here is one more example of making a plot using explicit Axes interface.

Python Code 3: ai202209_s04_02.py

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

#
# Time-stamp: <2022/10/02 23:36:26 (CST) daisuke>
#

# importing matplotlib module
import matplotlib.pyplot

# data to be plotted
data_x = [1.0, 2.0, 3.0, 4.0, 5.0]
data_y = [3.0, 2.0, 5.0, 1.0, 4.0]

# output file name
file_output = 'ai202209_s04_02.png'
```

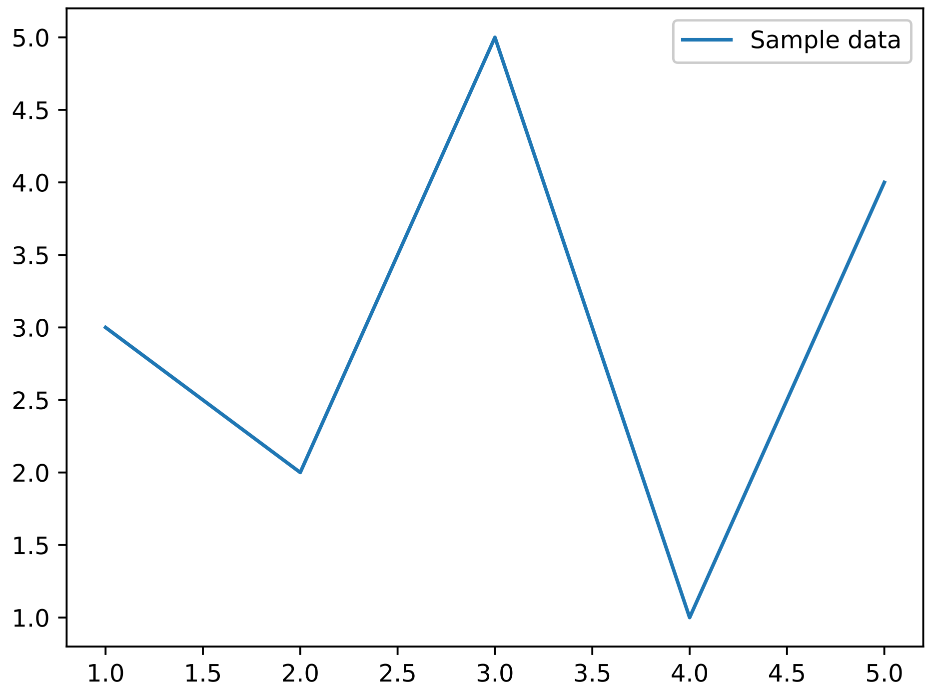


Figure 16: Plot created by the script `ai202209_s04_01.py`.

```
#
# for making a plot using object-oriented interface,
# we first construct "fig" and "axes" objects,
# and then use methods for these "fig" and "axes".
#

# making a fig and an axes objects using matplot.pyplot.subplots function
fig, ax = matplotlib.pyplot.subplots ()

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='Sample data')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_02.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 12:50 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:01 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:02 ai202209_s04_02.png
```

Display the PNG image file. (17)

```
% feh -dF ai202209_s04_02.png
```

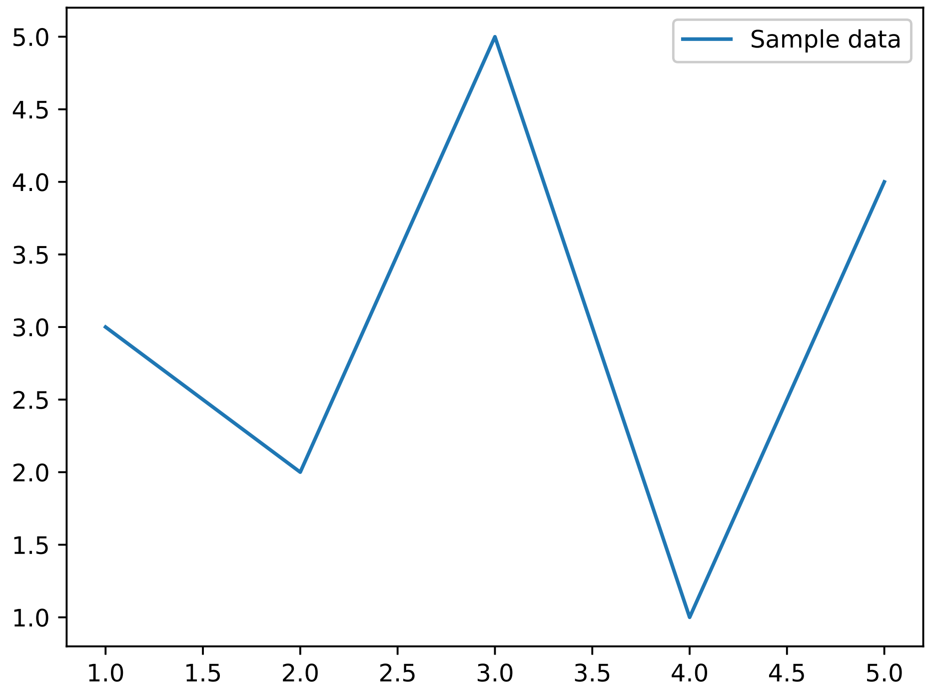


Figure 17: Plot created by the script `ai202209_s04_02.py`.

Try following practice.

Practice 04-03

Make your own Python script to create a plot using explicit Axes interface of Matplotlib. (Note: Use "`matplotlib.pyplot.subplots ()`" function to construct both "`fig`" object and "`axes`" object at once.)

3.3 Making a plot using pure object-oriented interface

Here is an example of making a plot using pure object-oriented interface.

Python Code 4: ai202209_s04_03.py

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

#
# Time-stamp: <2022/10/01 13:10:13 (CST) daisuke>
#

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

# data to be plotted
data_x = [1.0, 2.0, 3.0, 4.0, 5.0]
data_y = [3.0, 2.0, 5.0, 1.0, 4.0]
```

```
# output file name
file_output = 'ai202209_s04_03.png'

#
# for making a plot using object-oriented interface,
# we first construct "fig" and "axes" objects,
# and then use methods for these "fig" and "axes".
#

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='Sample data')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_03.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 12:50 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:01 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:02 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:07 ai202209_s04_03.png
```

Display the PNG image file. (18)

```
% feh -dF ai202209_s04_03.png
```

Try following practice.

Practice 04-04

Make your own Python script to create a plot using pure object-oriented interface of Matplotlib. (Note: Construct "`fig`" object using "`matplotlib.figure.Figure ()`", and then construct "`axes`" object using "`.add_subplot ()`" method.)

3.4 A note on Matplotlib APIs for this course

There are several different Matplotlib APIs. Explicit Axes interface enables us to customise a plot and give fine adjustments. For this course, we use explicit Axes interface for making plots.

For more information about Matplotlib APIs, read following documents.

- `https://matplotlib.org/stable/api/`
- `https://matplotlib.org/stable/users/explain/api_interfaces.html`

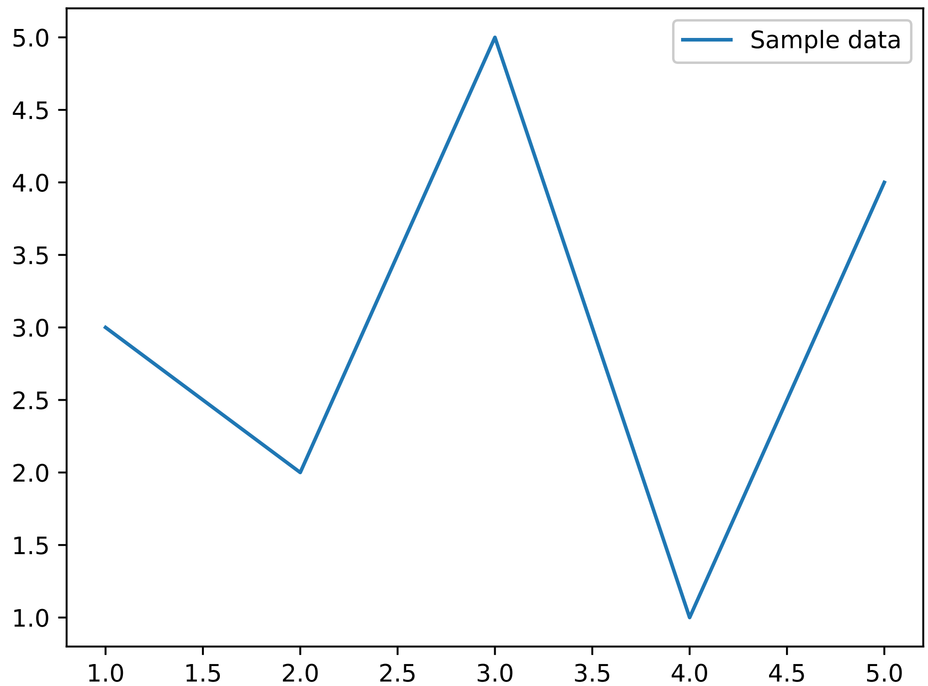


Figure 18: Plot created by the script `ai202209_s04_03.py`.

- `https://matplotlib.org/matplotblog/posts/pyplot-vs-object-oriented-interface/`
- `https://matplotlib.org/2.2.5/gallery/api/agg_oo_sgskip.html`
- `https://matplotlib.org/stable/gallery/user_interfaces/canvasagg.html`

4 Plotting a line

Make a plot of a straight line. Here is an example.

Python Code 5: ai202209_s04_04.py

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

#
# Time-stamp: <2022/10/01 21:55:57 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = 3.0 * data_x + 5.0

# output file name
```

```
file_output = 'ai202209_s04_04.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='$f(x) = 3x + 5$')

# setting ranges of x-axis and y-axis
ax.set_xlim (-1.0, +11.0)
ax.set_ylim (0.0, +40.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$ [arbitrary unit]')
ax.set_ylabel ('$y$ [arbitrary unit]')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_04.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
```

Display the PNG image file. (19)

```
% feh -dF ai202209_s04_04.png
```

Try following practice.

Practice 04-05

Make your own Python script to create a plot of a straight line $f(x) = -2x + 25$.

5 Plotting a curve

Make a plot of a curve. Here is an example.

Python Code 6: ai202209_s04_05.py

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

#
```

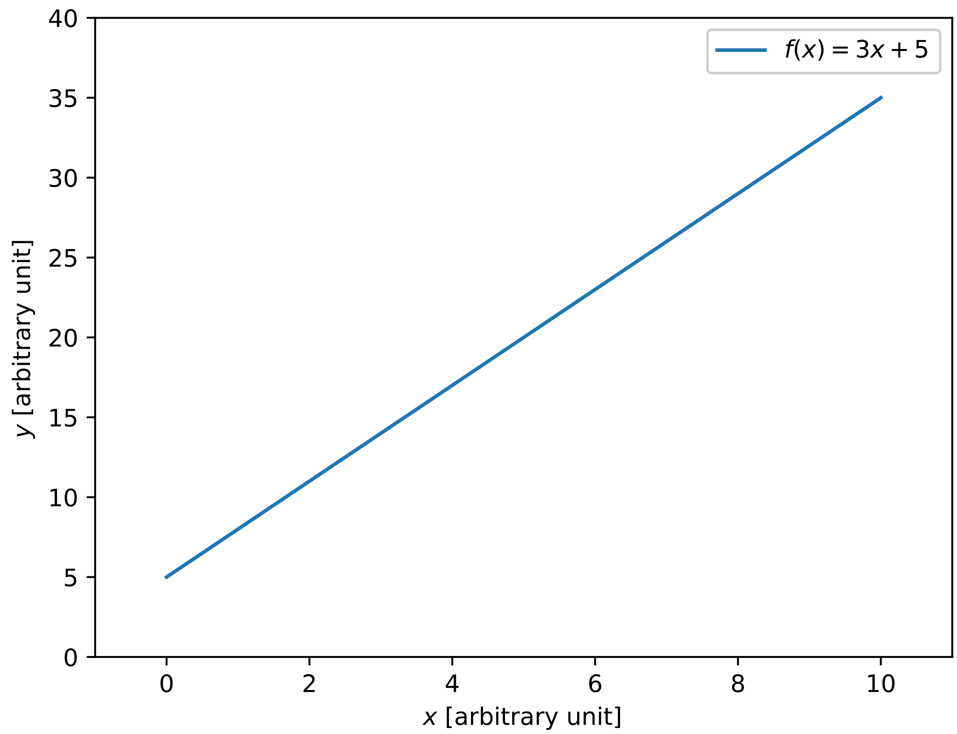


Figure 19: Plot created by the script `ai202209_s04_04.py`.

```
# Time-stamp: <2022/10/01 22:03:58 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = 2.0 * (data_x - 5.0)**2 + 3.0

# output file name
file_output = 'ai202209_s04_05.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='$f(x) = 2 (x-5)^2 + 3$')
```

```
# setting ranges of x-axis and y-axis
ax.set_xlim (-1.0, +11.0)
ax.set_ylim (0.0, +70.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$ [arbitrary unit]')
ax.set_ylabel ('$y$ [arbitrary unit]')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_05.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
-rw-r--r--  1 daisuke  taiwan  25102 Oct  1 22:02 ai202209_s04_05.png
```

Display the PNG image file. (20)

```
% feh -dF ai202209_s04_05.png
```

Try following practice.

Practice 04-06

Make your own Python script to create a plot of a curve $f(x) = -3(x+4)^2 + 20$.

6 Plotting a sine curve

Make a plot of a sine curve. Here is an example. Following example sets ticks for x and y axes. It also shows grid on the plot.

Python Code 7: ai202209_s04_06.py

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

#
# Time-stamp: <2022/10/01 22:15:28 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
```

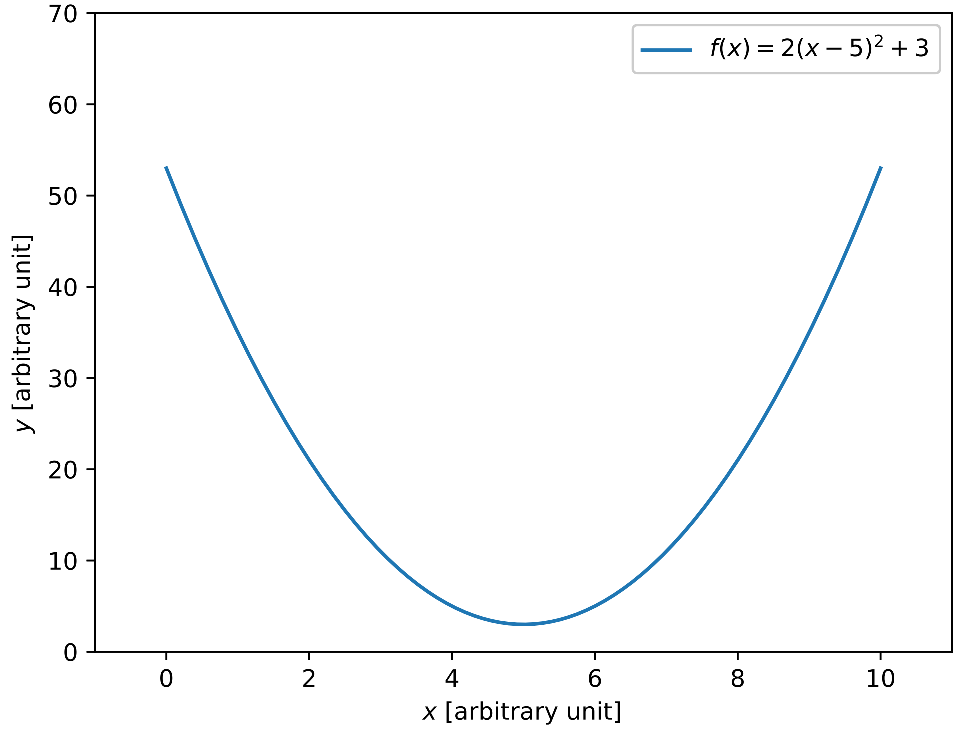


Figure 20: Plot created by the script `ai202209_s04_05.py`.

```
data_x = numpy.linspace (0.0, 720.0, 7201)
data_y = numpy.sin ( numpy.deg2rad (data_x) )

# output file name
file_output = 'ai202209_s04_06.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='$f(x) = \sin (x)$')

# setting ranges of x-axis and y-axis
ax.set_xlim (0.0, +720.0)
ax.set_ylim (-1.2, +1.2)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$ [deg]')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, 720.0, 9))
```

```
ax.set_yticks (numpy.linspace (-1.0, +1.0, 11))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_06.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
-rw-r--r--  1 daisuke  taiwan  25102 Oct  1 22:02 ai202209_s04_05.png
-rw-r--r--  1 daisuke  taiwan  30436 Oct  1 22:15 ai202209_s04_06.png
```

Display the PNG image file. (21)

```
% feh -dF ai202209_s04_06.png
```

Try following practice.

Practice 04-07

Make your own Python script to create a plot of a cosine curve $f(x) = 2\cos(x - 90°)$.

7 Plotting a circle

Make a plot of a circle. Here is an example.

Python Code 8: ai202209_s04_07.py

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

#
# Time-stamp: <2022/10/01 22:29:43 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
radius = 3.0
theta  = numpy.linspace (0.0, 2.0 * numpy.pi, 1001)
data_x = radius * numpy.cos (theta)
data_y = radius * numpy.sin (theta)
```

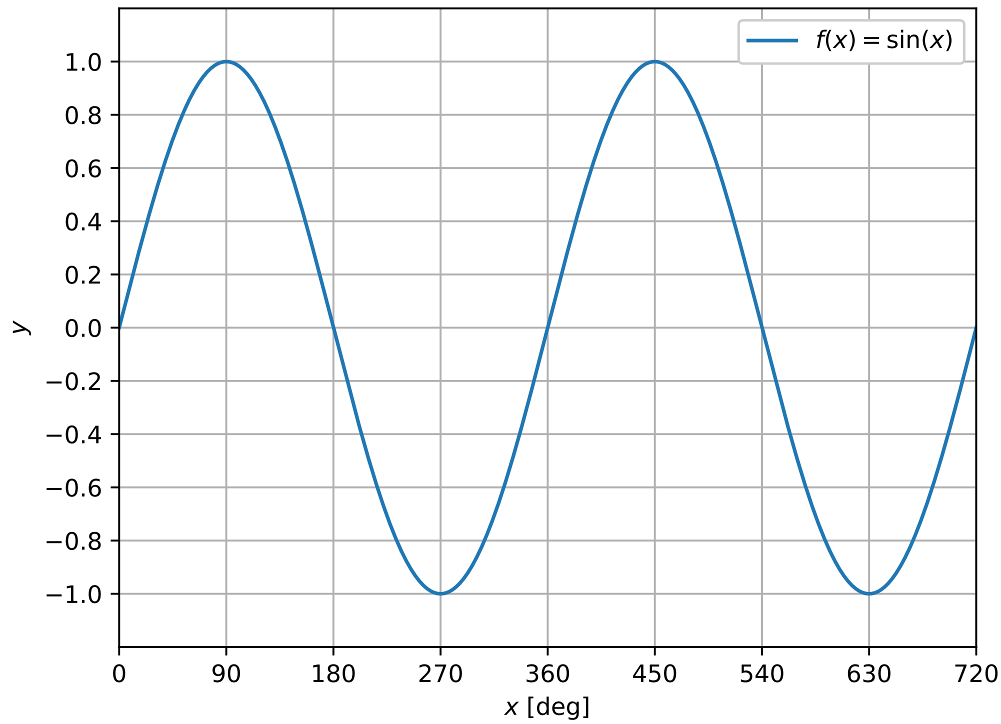


Figure 21: Plot created by the script `ai202209_s04_06.py`.

```
# output file name
file_output = 'ai202209_s04_07.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, label='circle of radius 3')

# setting ranges of x-axis and y-axis
ax.set_xlim (-4.0, +4.0)
ax.set_ylim (-4.0, +4.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-4.0, +4.0, 9))
ax.set_yticks (numpy.linspace (-4.0, +4.0, 9))
```

```
# showing grid
ax.grid ()

# setting aspect ratio
ax.set_aspect ('equal', 'box')

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_07.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
-rw-r--r--  1 daisuke  taiwan  25102 Oct  1 22:02 ai202209_s04_05.png
-rw-r--r--  1 daisuke  taiwan  30436 Oct  1 22:15 ai202209_s04_06.png
-rw-r--r--  1 daisuke  taiwan  23172 Oct  1 22:29 ai202209_s04_07.png
```

Display the PNG image file. (22)

```
% feh -dF ai202209_s04_07.png
```

Try following practice.

Practice 04-08

Make your own Python script to create a plot of an ellipse.

8 Plotting multiple lines/curves

Make a plot of multiple lines/curves. Here is an example.

Python Code 9: ai202209_s04_08.py

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

#
# Time-stamp: <2022/10/01 22:49:30 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x   = numpy.linspace (-360.0, +360.0, 10**4)
data_sin = numpy.sin ( numpy.deg2rad (data_x) )
```

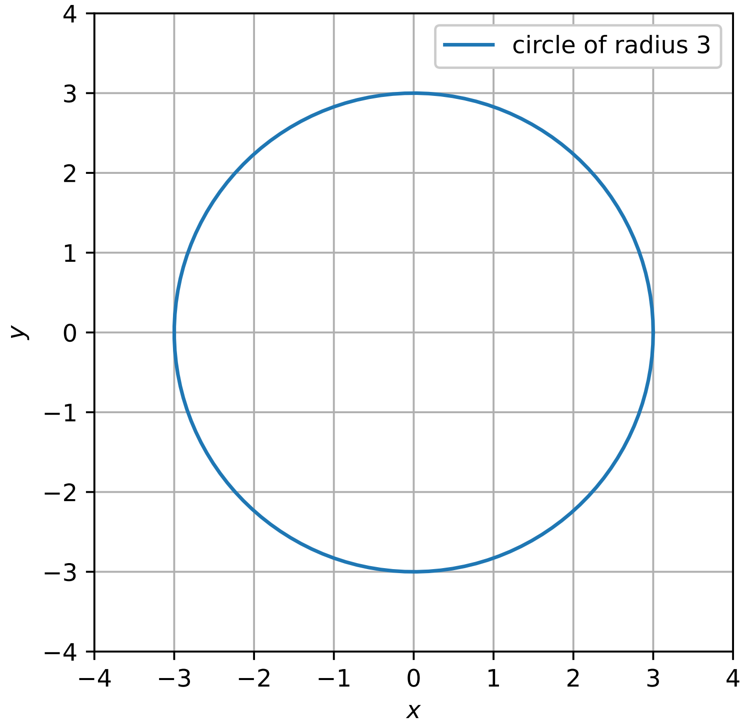


Figure 22: Plot created by the script ai202209_s04_07.py.

```
data_cos = numpy.cos ( numpy.deg2rad (data_x) )

# output file name
file_output = 'ai202209_s04_08.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_sin, label='sine curve')
ax.plot (data_x, data_cos, label='cosine curve')

# setting ranges of x-axis and y-axis
ax.set_xlim (-360.0, +360.0)
ax.set_ylim (-1.2, +1.2)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$ [deg]')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-360.0, +360.0, 9))
```

```
ax.set_yticks (numpy.linspace (-1.0, +1.0, 5))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_08.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
-rw-r--r--  1 daisuke  taiwan  25102 Oct  1 22:02 ai202209_s04_05.png
-rw-r--r--  1 daisuke  taiwan  30436 Oct  1 22:15 ai202209_s04_06.png
-rw-r--r--  1 daisuke  taiwan  23172 Oct  1 22:29 ai202209_s04_07.png
-rw-r--r--  1 daisuke  taiwan  39369 Oct  1 22:46 ai202209_s04_08.png
```

Display the PNG image file. (23)

```
% feh -dF ai202209_s04_08.png
```

Try following practice.

Practice 04-09

Make your own Python script to create a plot of two different curves.

Here is an example of plotting five lines.

Python Code 10: ai202209_s04_09.py

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

#
# Time-stamp: <2022/10/02 07:39:05 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x  = numpy.linspace (-10.0, +10.0, 10**3)
data_l1 = 2.0 * data_x - 12.0
data_l2 = 2.0 * data_x - 6.0
data_l3 = 2.0 * data_x - 0.0
data_l4 = 2.0 * data_x + 6.0
```

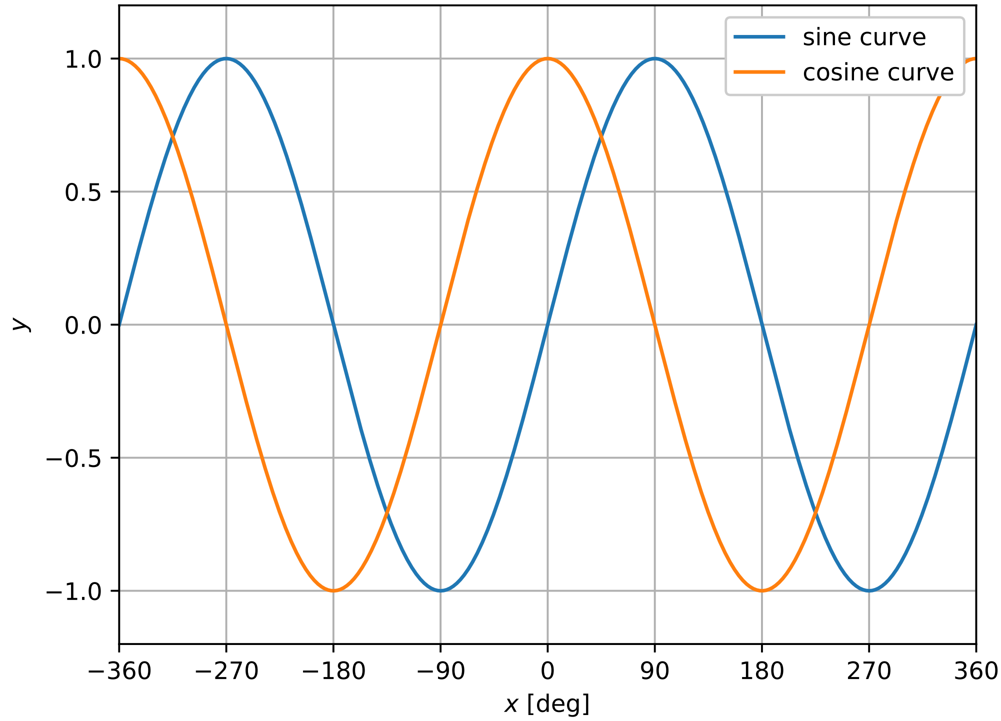


Figure 23: Plot created by the script `ai202209_s04_08.py`.

```
data_l5 = 2.0 * data_x + 12.0

# output file name
file_output = 'ai202209_s04_09.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_l1, label='$f(x) = 2x - 12$')
ax.plot (data_x, data_l2, label='$f(x) = 2x - 6$')
ax.plot (data_x, data_l3, label='$f(x) = 2x$')
ax.plot (data_x, data_l4, label='$f(x) = 2x + 6$')
ax.plot (data_x, data_l5, label='$f(x) = 2x + 12$')

# setting ranges of x-axis and y-axis
ax.set_xlim (-10.0, +10.0)
ax.set_ylim (-30.0, +30.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')
```

```
# setting ticks
ax.set_xticks (numpy.linspace (-10.0, +10.0, 11))
ax.set_yticks (numpy.linspace (-30.0, +30.0, 11))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_09.py
% ls -l *.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_00.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_01.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:09 ai202209_s04_02.png
-rw-r--r--  1 daisuke  taiwan  25675 Oct  1 13:10 ai202209_s04_03.png
-rw-r--r--  1 daisuke  taiwan  21634 Oct  1 21:55 ai202209_s04_04.png
-rw-r--r--  1 daisuke  taiwan  25102 Oct  1 22:02 ai202209_s04_05.png
-rw-r--r--  1 daisuke  taiwan  30436 Oct  1 22:15 ai202209_s04_06.png
-rw-r--r--  1 daisuke  taiwan  23172 Oct  1 22:29 ai202209_s04_07.png
-rw-r--r--  1 daisuke  taiwan  39369 Oct  1 22:49 ai202209_s04_08.png
-rw-r--r--  1 daisuke  taiwan  53290 Oct  1 22:55 ai202209_s04_09.png
```

Display the PNG image file. (24)

```
% feh -dF ai202209_s04_09.png
```

Try following practice.

Practice 04-10

Make your own Python script to create a plot of three different curves.

9 Changing properties of lines/curves

Try to change properties of lines/curves.

9.1 Changing colours of line/curves

Change the colours of lines/curves. Here is an example.

Python Code 11: ai202209_s04_10.py

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

#
# Time-stamp: <2022/10/02 07:39:27 (CST) daisuke>
#

# importing numpy module
import numpy
```

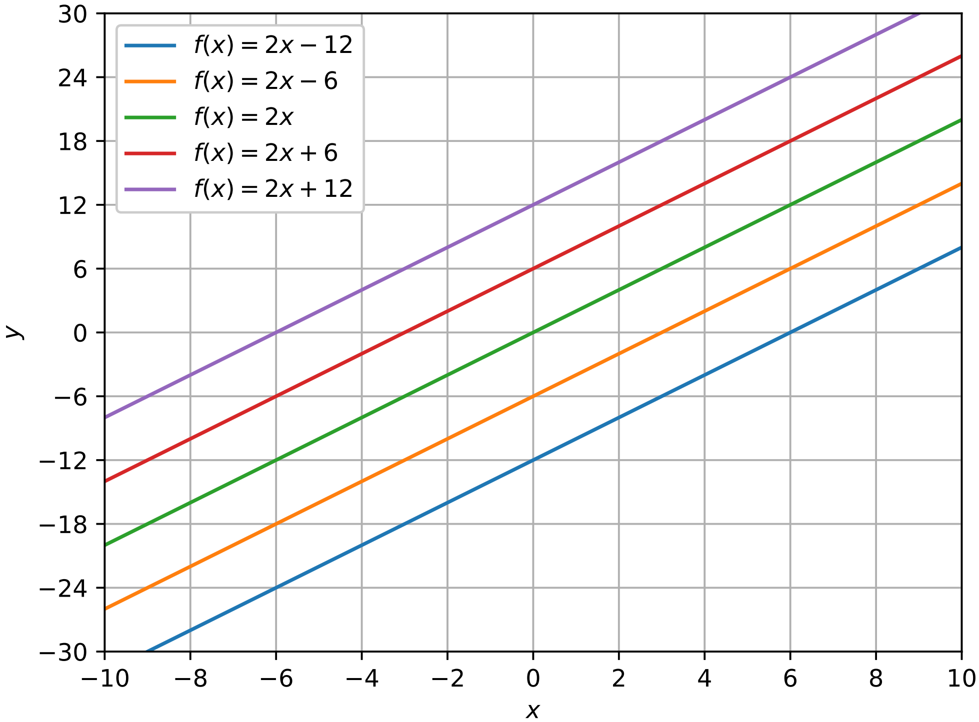


Figure 24: Plot created by the script `ai202209_s04_09.py`.

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

# data to be plotted
data_x  = numpy.linspace (-10.0, +10.0, 10**3)
data_l0 = 2.0 * data_x - 30.0
data_l1 = 2.0 * data_x - 20.0
data_l2 = 2.0 * data_x - 10.0
data_l3 = 2.0 * data_x - 0.0
data_l4 = 2.0 * data_x + 10.0
data_l5 = 2.0 * data_x + 20.0
data_l6 = 2.0 * data_x + 30.0

# output file name
file_output = 'ai202209_s04_10.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
```

```
ax.plot (data_x, data_l0, color='red',     label='$f(x) = 2x - 30$')
ax.plot (data_x, data_l1, color='green',   label='$f(x) = 2x - 20$')
ax.plot (data_x, data_l2, color='blue',    label='$f(x) = 2x - 10$')
ax.plot (data_x, data_l3, color='cyan',    label='$f(x) = 2x$')
ax.plot (data_x, data_l4, color='magenta', label='$f(x) = 2x + 10$')
ax.plot (data_x, data_l5, color='yellow',  label='$f(x) = 2x + 20$')
ax.plot (data_x, data_l6, color='grey',    label='$f(x) = 2x + 30$')

# setting ranges of x-axis and y-axis
ax.set_xlim (-10.0, +10.0)
ax.set_ylim (-60.0, +100.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-10.0, +10.0, 11))
ax.set_yticks (numpy.linspace (-60.0, +100.0, 17))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_10.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_04.png       ai202209_s04_08.png
ai202209_s04_01.png       ai202209_s04_05.png       ai202209_s04_09.png
ai202209_s04_02.png       ai202209_s04_06.png       ai202209_s04_10.png
ai202209_s04_03.png       ai202209_s04_07.png
```

Display the PNG image file. (25)

```
% feh -dF ai202209_s04_10.png
```

Names of colours you can use for Matplotlib can be found at following web page. (Fig. 26)

- List of named colors: https://matplotlib.org/stable/gallery/color/named_colors.html

Try following practice.

Practice 04-11

Make your own Python script to create a plot of a line (or a curve) using your favourite colour.

9.2 Changing line styles

Change the line style of lines/curves. Here is an example.

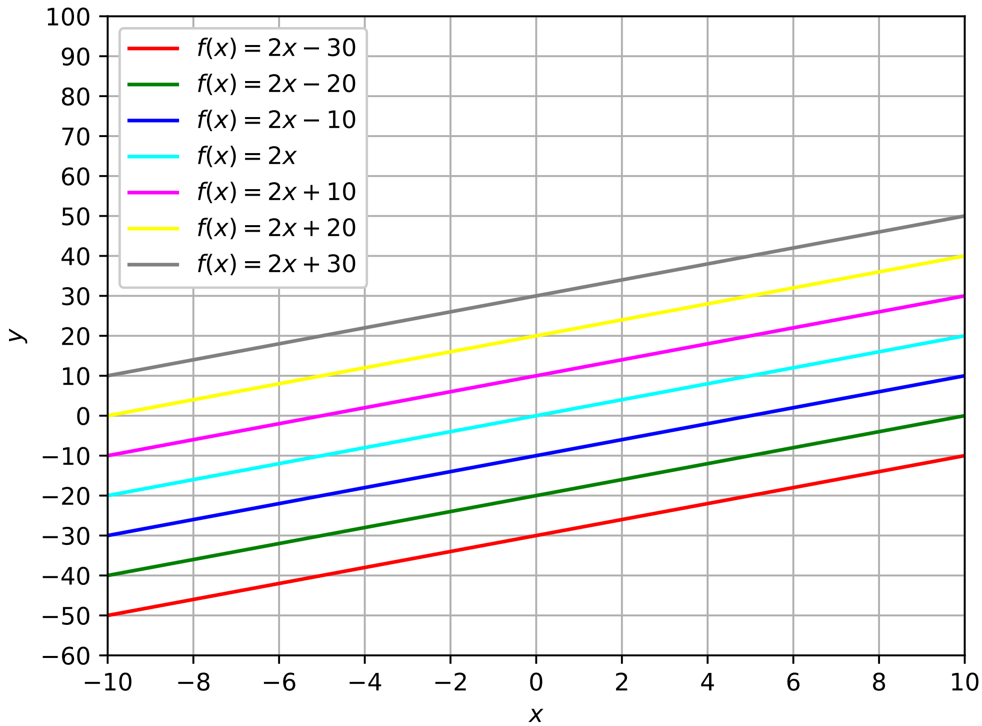


Figure 25: Plot created by the script `ai202209_s04_10.py`.

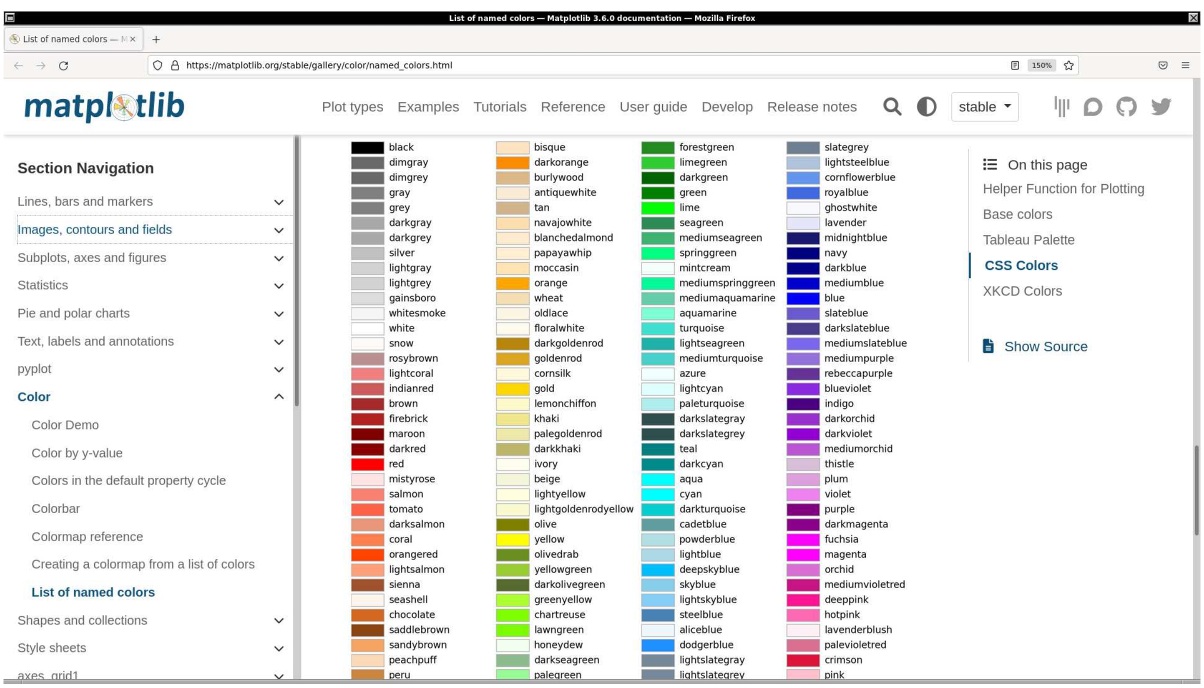


Figure 26: The list of available colour names for Matplotlib.

Python Code 12: ai202209_s04_11.py

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

#
# Time-stamp: <2022/10/02 09:40:13 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x  = numpy.linspace (-10.0, +10.0, 10**3)
data_c0 = numpy.sin (data_x)
data_c1 = numpy.sin (data_x - numpy.pi / 4.0)
data_c2 = numpy.sin (data_x - numpy.pi / 2.0)
data_c3 = numpy.sin (data_x - numpy.pi * 3.0 / 4.0)

# output file name
file_output = 'ai202209_s04_11.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_c0, color='k', linestyle='-',  label='solid line')
ax.plot (data_x, data_c1, color='k', linestyle='--', label='dashed line')
ax.plot (data_x, data_c2, color='k', linestyle='-.', label='dash-dotted line')
ax.plot (data_x, data_c3, color='k', linestyle=':',  label='dotted line')

# setting ranges of x-axis and y-axis
ax.set_xlim (-10.0, +10.0)
ax.set_ylim (-1.2, +1.9)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-10.0, +10.0, 11))
ax.set_yticks (numpy.linspace (-1.0, +1.0, 5))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_11.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_04.png        ai202209_s04_08.png
ai202209_s04_01.png        ai202209_s04_05.png        ai202209_s04_09.png
ai202209_s04_02.png        ai202209_s04_06.png        ai202209_s04_10.png
ai202209_s04_03.png        ai202209_s04_07.png        ai202209_s04_11.png
```

Display the PNG image file. (27)

```
% feh -dF ai202209_s04_11.png
```

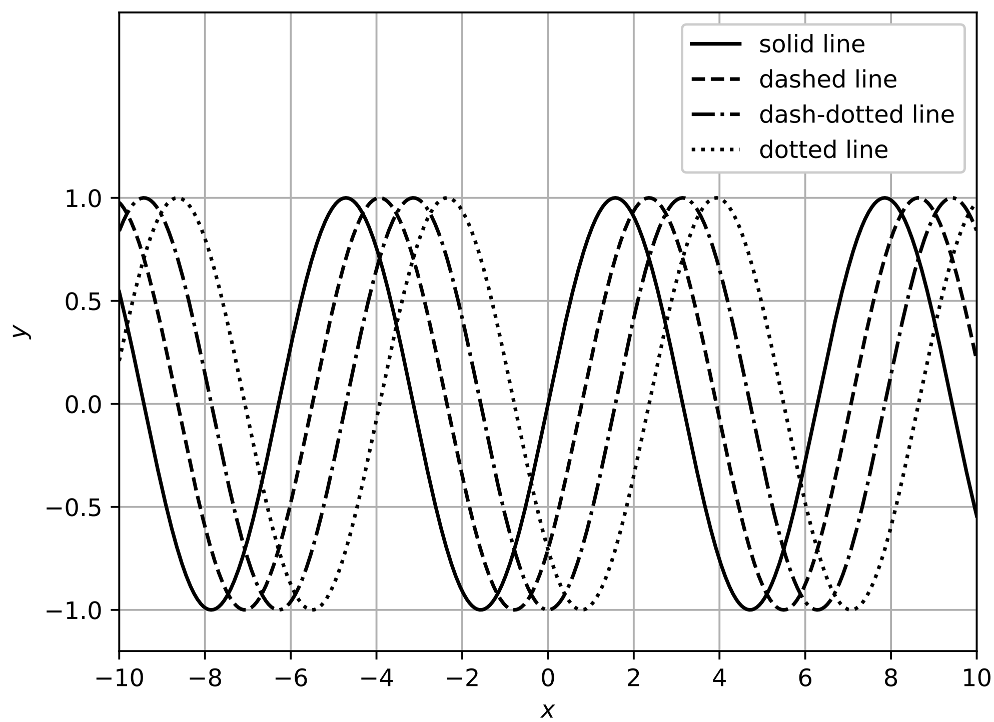


Figure 27: Plot created by the script `ai202209_s04_11.py`.

Try following practice.

Practice 04-12

Make your own Python script to create a plot of a line (or a curve) using dash-dotted line style.

9.3 Changing line width

Change the line width of lines/curves. Here is an example.

Python Code 13: ai202209_s04_12.py

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

#
# Time-stamp: <2022/10/02 09:50:43 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x  = numpy.linspace (-10.0, +10.0, 10**3)
data_c0 = numpy.cos (data_x)
data_c1 = numpy.cos (data_x - numpy.pi / 4.0)
data_c2 = numpy.cos (data_x - numpy.pi / 2.0)
data_c3 = numpy.cos (data_x - numpy.pi * 3.0 / 4.0)

# output file name
file_output = 'ai202209_s04_12.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_c0, c='k', ls='-', linewidth=1, label='linewidth=1')
ax.plot (data_x, data_c1, c='k', ls='-', linewidth=2, label='linewidth=2')
ax.plot (data_x, data_c2, c='k', ls='-', linewidth=3, label='linewidth=3')
ax.plot (data_x, data_c3, c='k', ls='-', linewidth=4, label='linewidth=4')

# setting ranges of x-axis and y-axis
ax.set_xlim (-10.0, +10.0)
ax.set_ylim (-1.2, +1.9)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-10.0, +10.0, 11))
ax.set_yticks (numpy.linspace (-1.0, +1.0, 5))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_12.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_05.png        ai202209_s04_10.png
ai202209_s04_01.png        ai202209_s04_06.png        ai202209_s04_11.png
ai202209_s04_02.png        ai202209_s04_07.png        ai202209_s04_12.png
ai202209_s04_03.png        ai202209_s04_08.png
ai202209_s04_04.png        ai202209_s04_09.png
```

Display the PNG image file. (28)

```
% feh -dF ai202209_s04_12.png
```

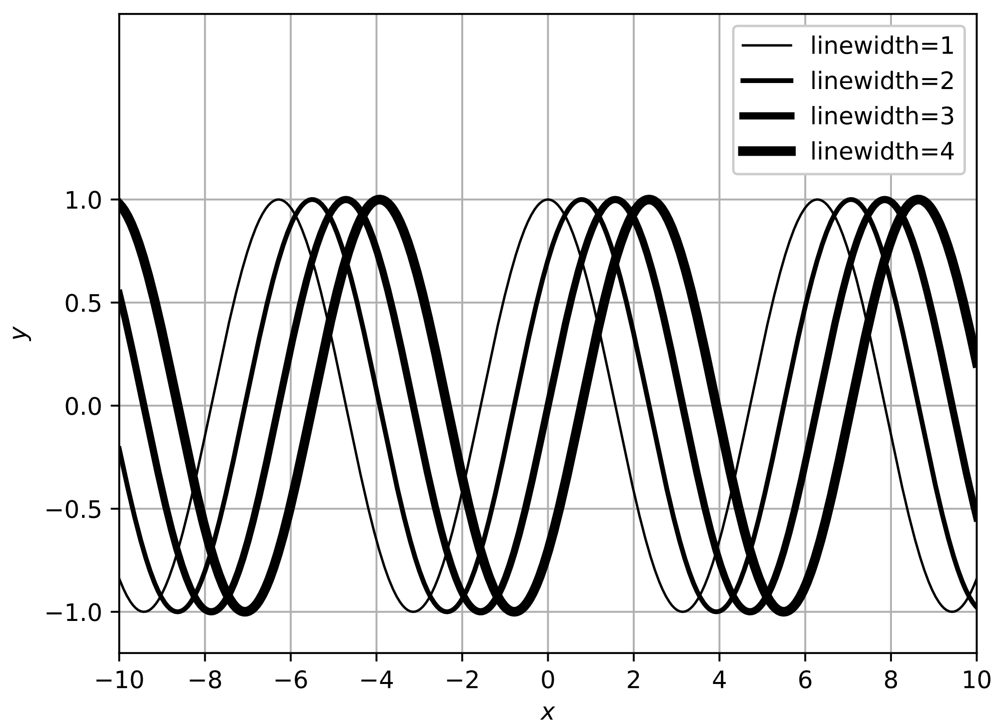


Figure 28: Plot created by the script ai202209_s04_12.py.

Try following practice.

Practice 04-13

Make your own Python script to create a plot of three lines (or three curves) using different line width.

9.4 Combinations of colour, line style, and line width

Try following example.

Python Code 14: ai202209_s04_13.py

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

#
# Time-stamp: <2022/10/02 09:58:37 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x  = numpy.linspace (-10.0, +10.0, 10**3)
data_c0 = numpy.cos (data_x)
data_c1 = numpy.cos (data_x - numpy.pi / 4.0)
data_c2 = numpy.cos (data_x - numpy.pi / 2.0)
data_c3 = numpy.cos (data_x - numpy.pi * 3.0 / 4.0)

# output file name
file_output = 'ai202209_s04_13.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_c0, c='k', ls='-',  lw=1, label='c=k, ls=-,  lw=1')
ax.plot (data_x, data_c1, c='r', ls='--', lw=2, label='c=r, ls=--, lw=2')
ax.plot (data_x, data_c2, c='g', ls='-.', lw=3, label='c=g, ls=-., lw=3')
ax.plot (data_x, data_c3, c='b', ls=':',  lw=4, label='c=b, ls=:,  lw=4')

# setting ranges of x-axis and y-axis
ax.set_xlim (-10.0, +10.0)
ax.set_ylim (-1.2, +1.9)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (-10.0, +10.0, 11))
ax.set_yticks (numpy.linspace (-1.0, +1.0, 5))

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_13.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_05.png       ai202209_s04_10.png
ai202209_s04_01.png       ai202209_s04_06.png       ai202209_s04_11.png
ai202209_s04_02.png       ai202209_s04_07.png       ai202209_s04_12.png
ai202209_s04_03.png       ai202209_s04_08.png       ai202209_s04_13.png
ai202209_s04_04.png       ai202209_s04_09.png
```

Display the PNG image file. (29)

```
% feh -dF ai202209_s04_13.png
```

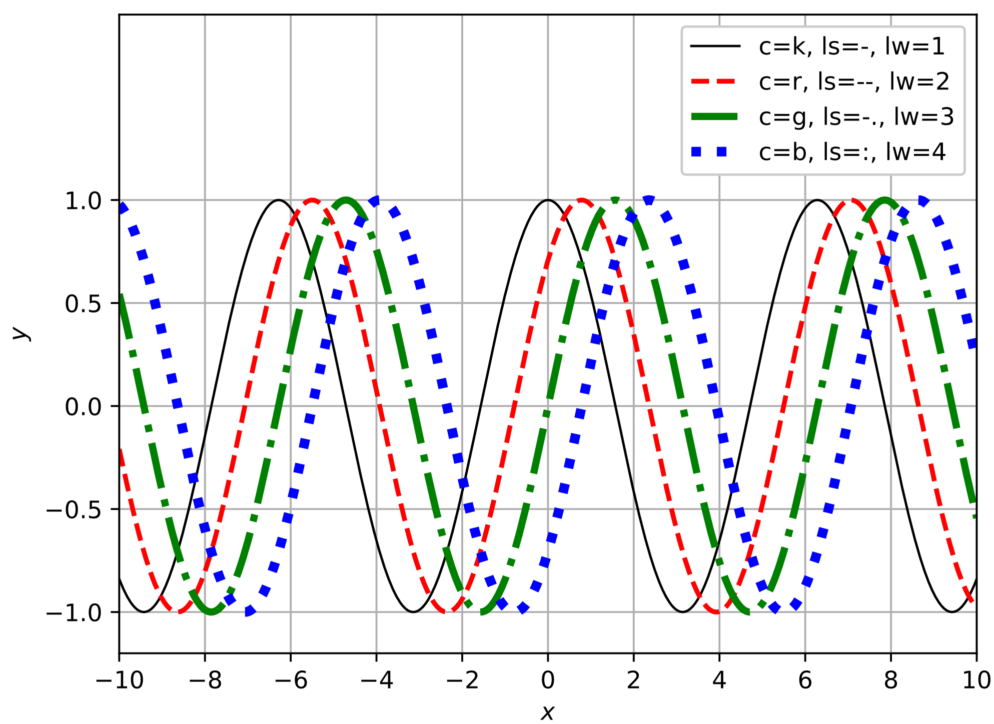


Figure 29: Plot created by the script `ai202209_s04_13.py`.

Try following practice.

Practice 04-14

Make your own Python script to create a plot of three lines (or three curves) using different colour, line style, and line width.

10 Plotting data

10.1 Plotting points

Plot points. Here is an example.

Python Code 15: ai202209_s04_14.py

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

#
# Time-stamp: <2022/10/02 23:36:37 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
rng    = numpy.random.default_rng ()
data_x = rng.uniform (0.0, 100.0, 50)
data_y = rng.uniform (0.0, 100.0, 50)

# output file name
file_output = 'ai202209_s04_14.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, c='b', ls='None', marker='.', label='marker=.')

# setting ranges of x-axis and y-axis
ax.set_xlim (-1.0, +101.0)
ax.set_ylim (-1.0, +101.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, +100.0, 11))
ax.set_yticks (numpy.linspace (0.0, +100.0, 11))

# setting aspect ratio
ax.set_aspect ('equal')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_14.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_05.png        ai202209_s04_10.png
ai202209_s04_01.png        ai202209_s04_06.png        ai202209_s04_11.png
ai202209_s04_02.png        ai202209_s04_07.png        ai202209_s04_12.png
ai202209_s04_03.png        ai202209_s04_08.png        ai202209_s04_13.png
ai202209_s04_04.png        ai202209_s04_09.png        ai202209_s04_14.png
```

Display the PNG image file. (30)

```
% feh -dF ai202209_s04_14.png
```

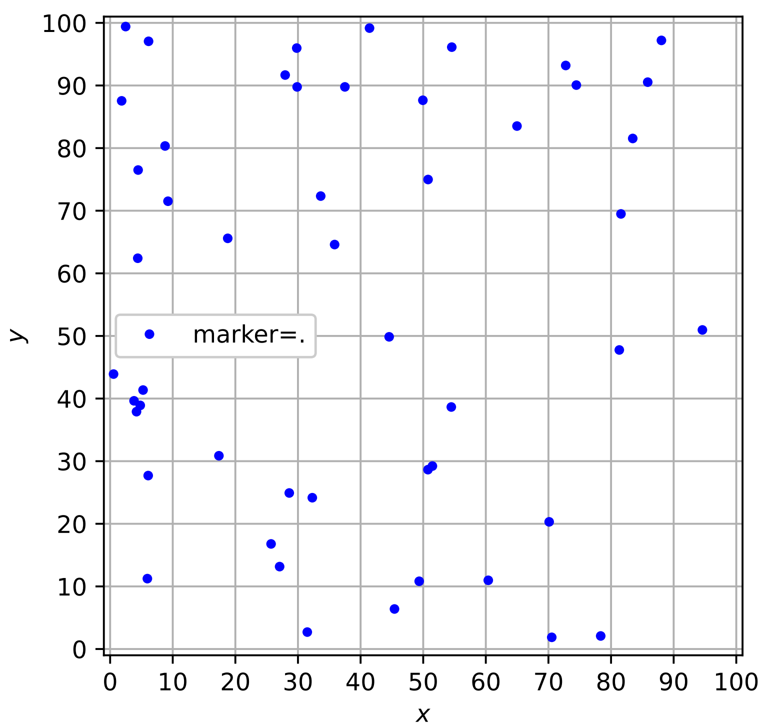


Figure 30: Plot created by the script `ai202209_s04_14.py`.

Try following practice.

Practice 04-15

Make your own Python script to create a plot of a set of data points.

10.2 Plotting points using different markers

Plot points using specified markers. Here is an example.

Python Code 16: ai202209_s04_15.py

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

#
```

```
# Time-stamp: <2022/10/02 10:53:04 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.concatenate ([
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4) ])
data_y = numpy.concatenate ([
    numpy.repeat (1.0, 4), \
    numpy.repeat (2.0, 4), \
    numpy.repeat (3.0, 4), \
    numpy.repeat (4.0, 4) ])

# list of markers
list_marker = [
    '.', 'o', 'v', '^', \
    '<', '>', 's', 'p', \
    'P', '*', 'h', '+', \
    'x', 'X', 'D', 'd' \
]

# list of labels
list_label = [
    'marker="."', 'marker="o"', 'marker="v"', 'marker="^"', \
    'marker="<"', 'marker=">"', 'marker="s"', 'marker="p"', \
    'marker="P"', 'marker="*"', 'marker="h"', 'marker="+"', \
    'marker="x"', 'marker="X"', 'marker="D"', 'marker="d"' \
]

# output file name
file_output = 'ai202209_s04_15.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
for i in range (len (data_x)):
    ax.plot (data_x[i], data_y[i], ls='None', marker=list_marker[i], \
             label=list_label[i])

# setting ranges of x-axis and y-axis
ax.set_xlim (0.0, +8.0)
ax.set_ylim (0.0, +6.0)
```

```
# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, +8.0, 9))
ax.set_yticks (numpy.linspace (0.0, +6.0, 7))

# setting aspect ratio
ax.set_aspect ('equal')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_15.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_06.png        ai202209_s04_12.png
ai202209_s04_01.png        ai202209_s04_07.png        ai202209_s04_13.png
ai202209_s04_02.png        ai202209_s04_08.png        ai202209_s04_14.png
ai202209_s04_03.png        ai202209_s04_09.png        ai202209_s04_15.png
ai202209_s04_04.png        ai202209_s04_10.png
ai202209_s04_05.png        ai202209_s04_11.png
```

Display the PNG image file. (31)

```
% feh -dF ai202209_s04_15.png
```

For the choices of markers, visit following page. (Fig. 32)

- https://matplotlib.org/stable/api/markers_api.html

Try following practice.

Practice 04-16

Make your own Python script to create a plot of a set of data points using your favourite marker.

10.3 Plotting points of different marker sizes

Plot points using specified marker sizes. Here is an example.

Python Code 17: ai202209_s04_16.py

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

#
# Time-stamp: <2022/10/02 11:05:50 (CST) daisuke>
#

# importing numpy module
```

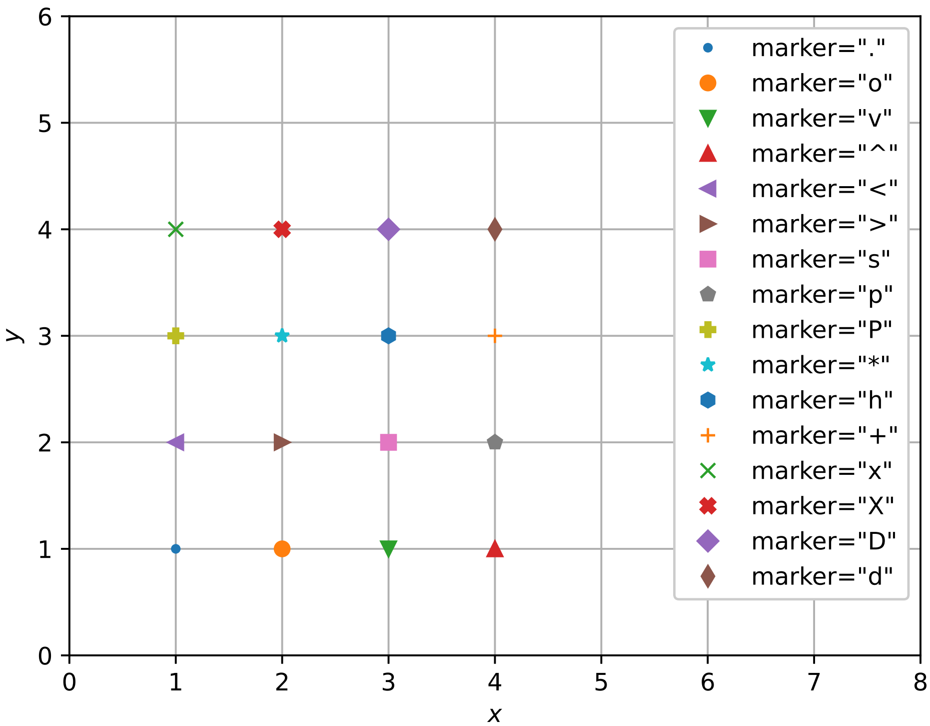


Figure 31: Plot created by the script `ai202209_s04_15.py`.

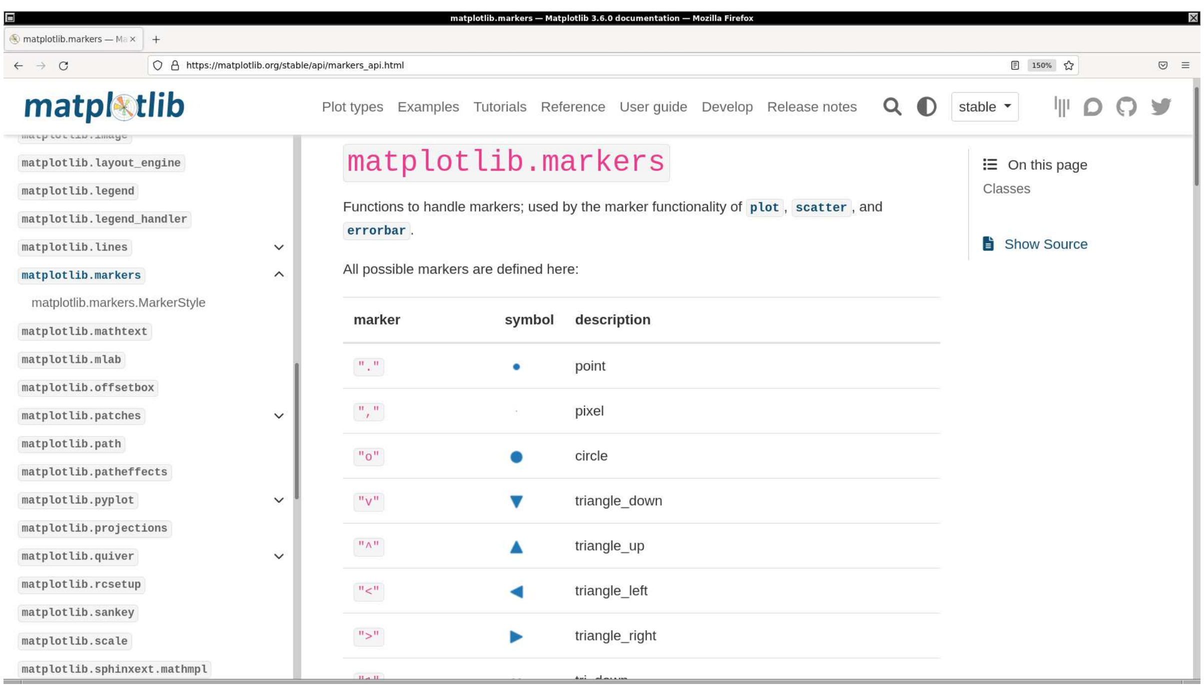


Figure 32: The list of markers for Matplotlib.

```
import numpy

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

# data to be plotted
data_x = numpy.concatenate ([
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4), \
    numpy.linspace (1.0, 4.0, 4) ])
data_y = numpy.concatenate ([
    numpy.repeat (1.0, 4), \
    numpy.repeat (2.0, 4), \
    numpy.repeat (3.0, 4), \
    numpy.repeat (4.0, 4) ])

# output file name
file_output = 'ai202209_s04_16.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
for i in range (len (data_x)):
    ax.plot (data_x[i], data_y[i], ls='None', marker='o', \
             markersize=i+1, label=f'markersize={i+1}')

# setting ranges of x-axis and y-axis
ax.set_xlim (0.0, +8.0)
ax.set_ylim (0.0, +6.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, +8.0, 9))
ax.set_yticks (numpy.linspace (0.0, +6.0, 7))

# setting aspect ratio
ax.set_aspect ('equal')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_16.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_06.png        ai202209_s04_12.png
ai202209_s04_01.png        ai202209_s04_07.png        ai202209_s04_13.png
ai202209_s04_02.png        ai202209_s04_08.png        ai202209_s04_14.png
ai202209_s04_03.png        ai202209_s04_09.png        ai202209_s04_15.png
ai202209_s04_04.png        ai202209_s04_10.png        ai202209_s04_16.png
ai202209_s04_05.png        ai202209_s04_11.png
```

Display the PNG image file. (33)

```
% feh -dF ai202209_s04_16.png
```

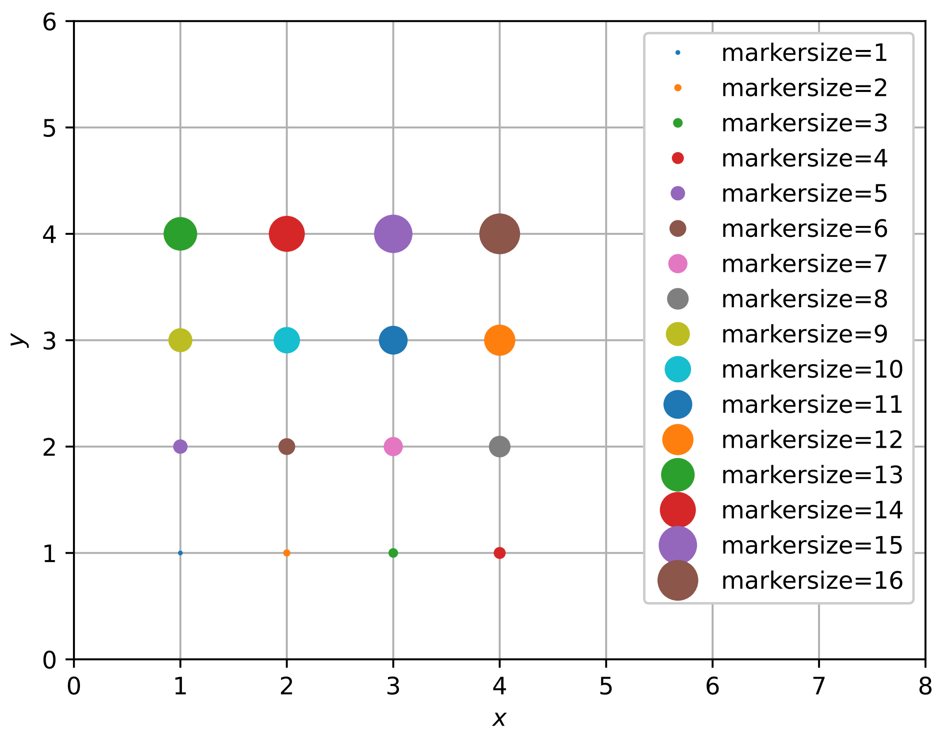


Figure 33: Plot created by the script `ai202209_s04_16.py`.

Try following practice.

Practice 04-17

Make your own Python script to create a plot of a set of data points using marker size of 20.

11 Attaching error bars to data points

We always need to show uncertainties for our scientific measurements.

11.1 Attaching error bars along vertical axis

Attach error bars along vertical axis. Here is an example.

Python Code 18: ai202209_s04_17.py

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

#
# Time-stamp: <2022/10/02 15:39:49 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (1.0, 10.0, 10)
data_y = 2.0 * data_x + 3.0

# generating random numbers for errors of y-value
rng = numpy.random.default_rng ()
data_y_err = rng.uniform (1.0, 3.0, 10)

# output file name
file_output = 'ai202209_s04_17.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.errorbar (data_x, data_y, yerr=data_y_err,
             linestyle='None', marker='o', markersize=8.0, color='green', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='Sample data')

# setting ranges of x-axis and y-axis
ax.set_xlim (0.0, +11.0)
ax.set_ylim (0.0, +30.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, +10.0, 11))
ax.set_yticks (numpy.linspace (0.0, +30.0, 7))

# showing grid
ax.grid ()

# adding legend to the plot
```

```
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_17.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_06.png        ai202209_s04_12.png
ai202209_s04_01.png        ai202209_s04_07.png        ai202209_s04_13.png
ai202209_s04_02.png        ai202209_s04_08.png        ai202209_s04_14.png
ai202209_s04_03.png        ai202209_s04_09.png        ai202209_s04_15.png
ai202209_s04_04.png        ai202209_s04_10.png        ai202209_s04_16.png
ai202209_s04_05.png        ai202209_s04_11.png        ai202209_s04_17.png
```

Display the PNG image file. (34)

```
% feh -dF ai202209_s04_17.png
```

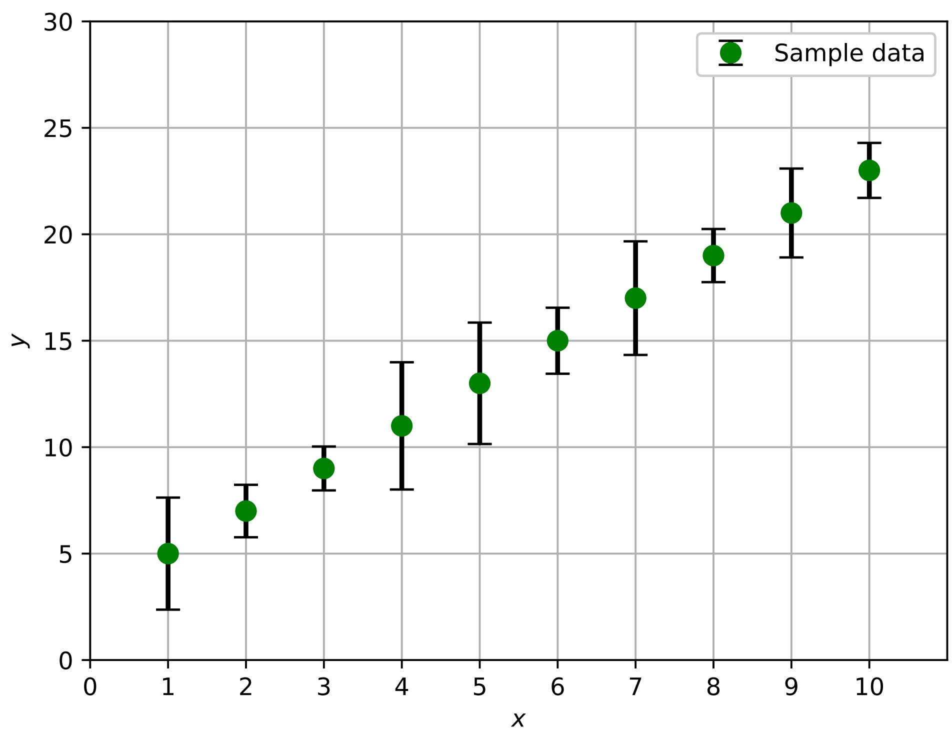


Figure 34: Plot created by the script `ai202209_s04_17.py`.

Try following practice.

Practice 04-18

Make your own Python script to create a plot of a set of data points with error bars along y-axis.

11.2 Attaching error bars along vertical axis and horizontal axis

Attach error bars along vertical axis and horizontal axis. Here is an example.

Python Code 19: ai202209_s04_18.py

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

#
# Time-stamp: <2022/10/02 15:41:25 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (1.0, 10.0, 10)
data_y = 2.0 * data_x + 3.0

# generating random numbers for errors of y-value
rng = numpy.random.default_rng ()
data_x_err = rng.uniform (0.3, 0.7, 10)
data_y_err = rng.uniform (1.0, 3.0, 10)

# output file name
file_output = 'ai202209_s04_18.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.errorbar (data_x, data_y, xerr=data_x_err, yerr=data_y_err,
             linestyle='None', marker='o', markersize=8.0, color='green', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='Sample data')

# setting ranges of x-axis and y-axis
ax.set_xlim (0.0, +11.0)
ax.set_ylim (0.0, +30.0)

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# setting ticks
ax.set_xticks (numpy.linspace (0.0, +10.0, 11))
ax.set_yticks (numpy.linspace (0.0, +30.0, 7))

# showing grid
ax.grid ()
```

```
# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_18.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_07.png        ai202209_s04_14.png
ai202209_s04_01.png        ai202209_s04_08.png        ai202209_s04_15.png
ai202209_s04_02.png        ai202209_s04_09.png        ai202209_s04_16.png
ai202209_s04_03.png        ai202209_s04_10.png        ai202209_s04_17.png
ai202209_s04_04.png        ai202209_s04_11.png        ai202209_s04_18.png
ai202209_s04_05.png        ai202209_s04_12.png
ai202209_s04_06.png        ai202209_s04_13.png
```

Display the PNG image file. (35)

```
% feh -dF ai202209_s04_18.png
```

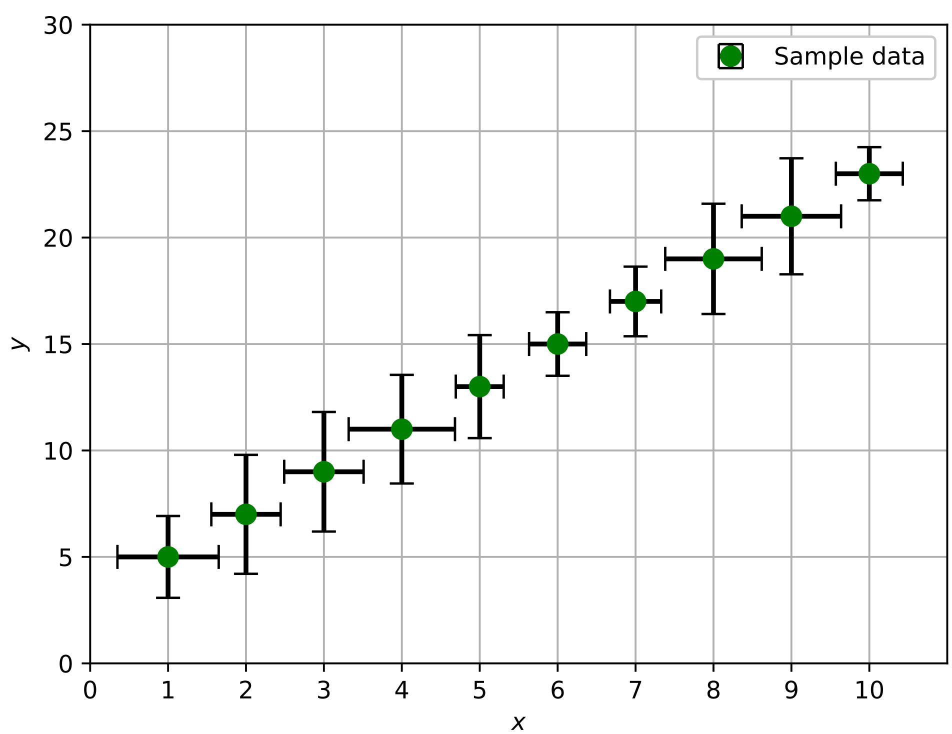


Figure 35: Plot created by the script `ai202209_s04_18.py`.

Try following practice.

Practice 04-19

Make your own Python script to create a plot of a set of data points with error bars both along x-axis and y-axis.

12 Using log scale for plots

Learn about using log scale for axes of plots.

12.1 Making a semi-log plot

First, make a plot of an exponential function using linear scale.

Python Code 20: ai202209_s04_19.py

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

#
# Time-stamp: <2022/10/02 15:59:51 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (-5.0, 5.0, 101)
data_y = 10.0**data_x

# output file name
file_output = 'ai202209_s04_19.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='orange', \
         label='Sample data')

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_19.py
% ls *.png
```

```
ai202209_s04_00.png      ai202209_s04_07.png      ai202209_s04_14.png
ai202209_s04_01.png      ai202209_s04_08.png      ai202209_s04_15.png
ai202209_s04_02.png      ai202209_s04_09.png      ai202209_s04_16.png
ai202209_s04_03.png      ai202209_s04_10.png      ai202209_s04_17.png
ai202209_s04_04.png      ai202209_s04_11.png      ai202209_s04_18.png
ai202209_s04_05.png      ai202209_s04_12.png      ai202209_s04_19.png
ai202209_s04_06.png      ai202209_s04_13.png
```

Display the PNG image file. (36)

```
% feh -dF ai202209_s04_19.png
```

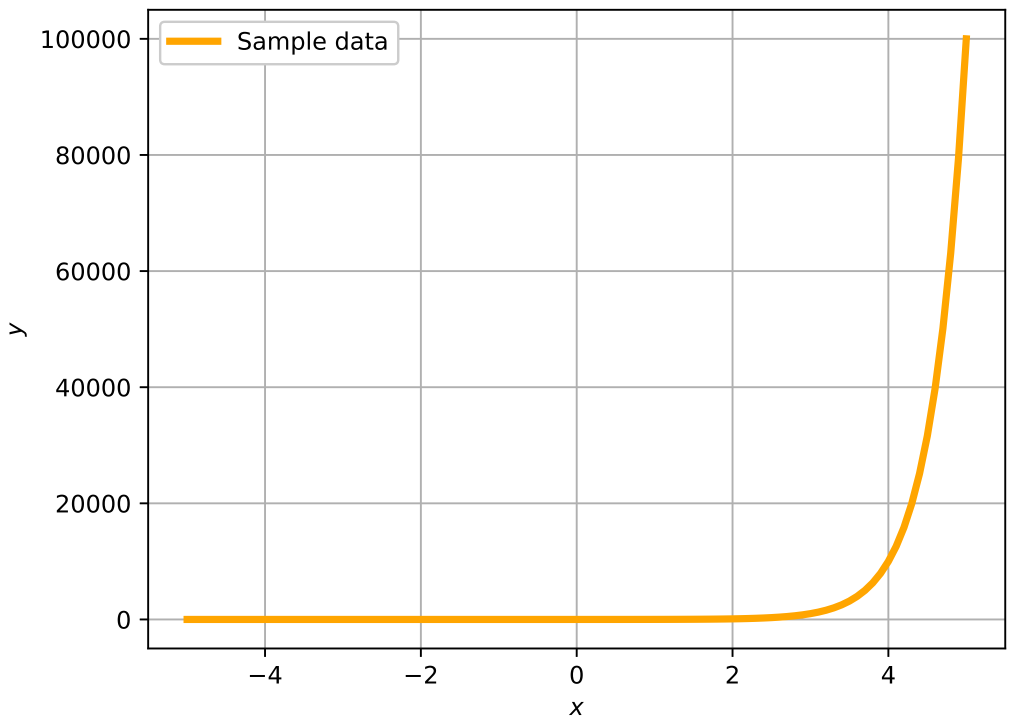


Figure 36: Plot created by the script `ai202209_s04_19.py`.

Try following practice.

Practice 04-20

Make your own Python script to create a plot of an exponential function using linear scale.

Then, make a plot of an exponential function using logarithmic scale.

Python Code 21: ai202209_s04_20.py

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

#
# Time-stamp: <2022/10/02 16:00:10 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

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

# data to be plotted
data_x = numpy.linspace (-5.0, 5.0, 101)
data_y = 10.0**data_x

# output file name
file_output = 'ai202209_s04_20.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='orange', \
         label='Sample data')

# setting log-scale
ax.set_yscale ('log')

# setting labels for x-axis and y-axis
ax.set_xlabel ('$x$')
ax.set_ylabel ('$y$')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_20.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_07.png        ai202209_s04_14.png
ai202209_s04_01.png        ai202209_s04_08.png        ai202209_s04_15.png
ai202209_s04_02.png        ai202209_s04_09.png        ai202209_s04_16.png
ai202209_s04_03.png        ai202209_s04_10.png        ai202209_s04_17.png
ai202209_s04_04.png        ai202209_s04_11.png        ai202209_s04_18.png
ai202209_s04_05.png        ai202209_s04_12.png        ai202209_s04_19.png
ai202209_s04_06.png        ai202209_s04_13.png        ai202209_s04_20.png
```

Display the PNG image file. (37)

```
% feh -dF ai202209_s04_20.png
```

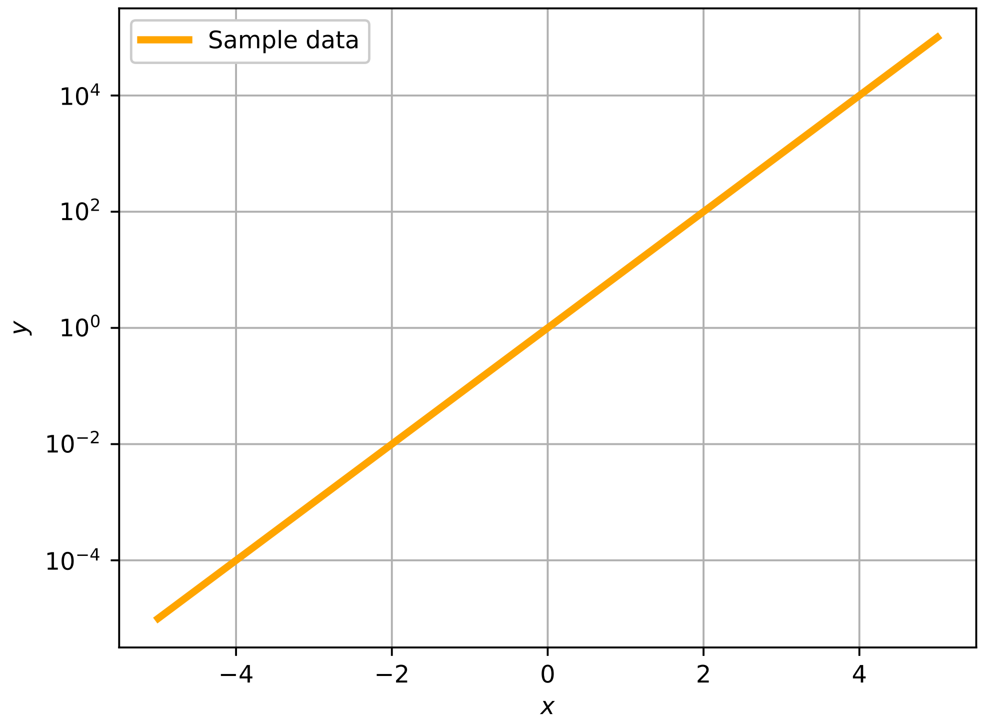


Figure 37: Plot created by the script `ai202209_s04_20.py`.

Try following practice.

Practice 04-21

Make your own Python script to create a plot of an exponential function using logarithmic scale.

12.2 Making a log-log plot

Make a log-log plot. Here is an example.

Python Code 22: ai202209_s04_21.py

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

#
# Time-stamp: <2022/10/02 16:20:35 (CST) daisuke>
#

# importing numpy module
import numpy

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

# semimajor axis and orbital period of planets
dic_planets = {
    'Mercury': {'a':  0.3871, 'P':    88.0, 'marker': 's', 'colour': 'b'},
    'Venus':   {'a':  0.7233, 'P':   224.7, 'marker': '^', 'colour': 'y'},
    'Earth':   {'a':  1.0000, 'P':   365.2, 'marker': 'o', 'colour': 'g'},
```

```
    'Mars':    {'a':  1.5237, 'P':   687.0, 'marker': 'v', 'colour': 'r'},
    'Jupiter': {'a':  5.2034, 'P':  4331.0, 'marker': 's', 'colour': 'm'},
    'Saturn':  {'a':  9.5371, 'P': 10747.0, 'marker': '^', 'colour': 'g'},
    'Uranus':  {'a': 19.1913, 'P': 30589.0, 'marker': 'o', 'colour': 'c'},
    'Neptune': {'a': 30.0690, 'P': 59800.0, 'marker': 'v', 'colour': 'b'},
}

# line to be plotted
line_x = numpy.linspace (0.1, 100.0, 10**3)
line_y = 365.256363004 * line_x**1.5

# output file name
file_output = 'ai202209_s04_21.png'

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# making a plot using object-oriented interface
ax.plot (line_x, line_y, linestyle='--', linewidth=3, color='coral')
for planet in dic_planets.keys ():
    ax.plot (dic_planets[planet]['a'], dic_planets[planet]['P'], \
             linestyle='None', \
             marker=dic_planets[planet]['marker'], markersize=10, \
             color=dic_planets[planet]['colour'], label=planet)

# setting log-scale
ax.set_xscale ('log')
ax.set_yscale ('log')

# setting labels for x-axis and y-axis
ax.set_xlabel ('Semimajor Axis [au]')
ax.set_ylabel ('Orbital Period [day]')

# showing grid
ax.grid ()

# adding legend to the plot
ax.legend ()

# saving a plot as a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_21.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_08.png       ai202209_s04_16.png
ai202209_s04_01.png       ai202209_s04_09.png       ai202209_s04_17.png
ai202209_s04_02.png       ai202209_s04_10.png       ai202209_s04_18.png
ai202209_s04_03.png       ai202209_s04_11.png       ai202209_s04_19.png
ai202209_s04_04.png       ai202209_s04_12.png       ai202209_s04_20.png
ai202209_s04_05.png       ai202209_s04_13.png       ai202209_s04_21.png
ai202209_s04_06.png       ai202209_s04_14.png
```

```
ai202209_s04_07.png        ai202209_s04_15.png
```

Display the PNG image file. (38)

```
% feh -dF ai202209_s04_21.png
```

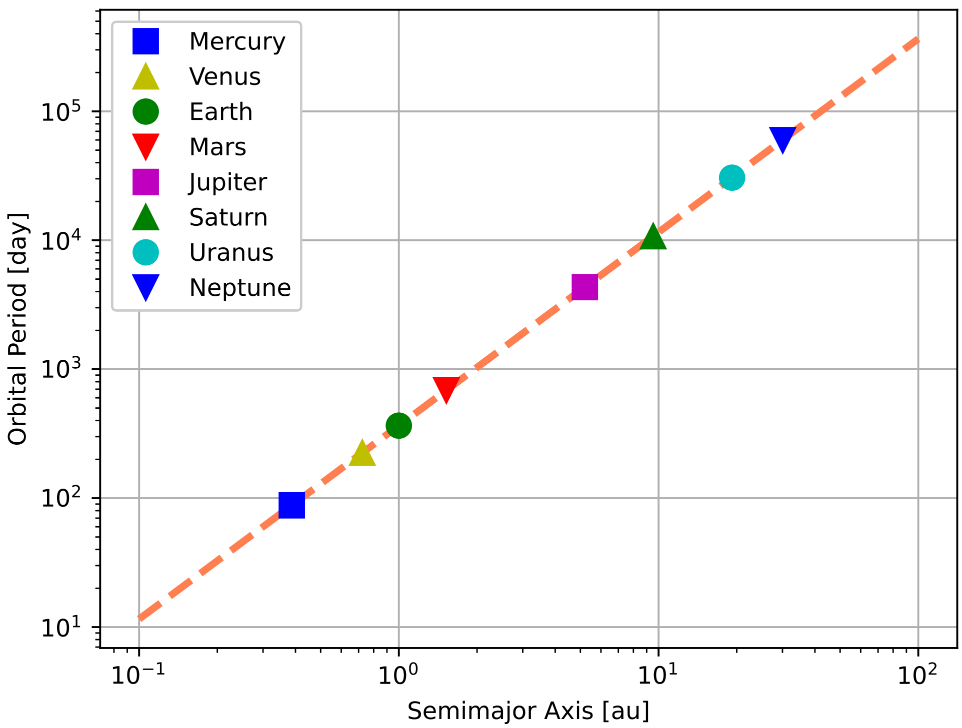


Figure 38: Plot created by the script `ai202209_s04_21.py`.

Try following practice.

Practice 04-22

Add dwarf planets (1) Ceres and (136199) Eris to above example and make a log-log plot.

13 Dealing with date/time

In astronomy, we often need to deal with date/time.

13.1 Downloading data file

Download data file.

Python Code 23: ai202209_s04_22.py

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

#
# Time-stamp: <2022/10/02 16:35:22 (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://s3b.astro.ncu.edu.tw/ai_202209/data/alf_ori.data'

# output file name
file_output = 'alf_ori.data'

# printing status
print (f'Now, fetching the file {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 the file {url_data}!')

# converting raw byte data into string
data_str = data_byte.decode ('utf-8')

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

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

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

Execute above script.

```
% ./ai202209_s04_22.py
% ls *.data
alf_ori.data
```

Show the first few lines of the data file.

```
% head alf_ori.data
2019-12-30.955  1.218  0.040  V  NOT
2020-01-01.171  1.410  0.008  V  SAH
2020-01-07.863  1.360  0.030  V  NOT
2020-01-08.711  1.510  0.000  V  NAO
2020-01-12.279  1.463  0.020  V  PTOB
2020-01-13.904  1.518  0.003  V  RZD
2020-01-13.904  1.579  0.003  V  RZD
2020-01-14.250  1.487  0.013  V  PTOB
```

```
2020-01-15.138  1.515  0.008  V  SAH
2020-01-16.928  1.536  0.020  V  NOT
```

13.2 Making a lightcurve of Betelgeuse

Read the data file and construct a lightcurve of Betelgeuse. Here is an example.

Python Code 24: ai202209_s04_23.py

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

# importing numpy module
import numpy

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

# importing datetime
import datetime

# input data file
file_input = 'alf_ori.data'

# output image file
file_output = 'ai202209_s04_23.png'

# making empty list and Numpy arrays
data_date  = numpy.array ([], dtype='datetime64[ms]')
data_mag   = numpy.array ([], dtype='float64')
data_error = numpy.array ([], dtype='float64')

# opening input file
with open (file_input, 'r') as fh_in:
    # reading data line-by-line
    for line in fh_in:
        # splitting data
        (date_str, mag_str, error_str, band, observer) = line.split ()
        # conversion from string to datetime, and then to datetime64
        date1 = datetime.datetime.strptime (date_str[:-4], '%Y-%m-%d')
        day   = float (date_str[-3:]) / 1000
        date2 = datetime.timedelta (days=day)
        date_datetime = date1 + date2
        date_datetime64 = numpy.datetime64 (date_datetime, 'ms')
        # conversion from string to float
        mag   = float (mag_str)
        error = float (error_str)
        # appending data to list and Numpy arrays
        data_date  = numpy.append (data_date, date_datetime64)
        data_mag   = numpy.append (data_mag, mag)
        data_error = numpy.append (data_error, error)

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
```

```
# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('Date [YYYY-MM-DD]')
ax.set_ylabel ('V-band Magnitude [mag]')

# axis settings
ax.set_xlim (numpy.datetime64 ('2019-12-20'), numpy.datetime64 ('2020-04-01'))
ax.set_ylim (+1.9, +0.9)

# plotting data
ax.errorbar (data_date, data_mag, yerr=data_error, linestyle='None', \
             marker='o', markersize=5.0, color='red', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='Betelgeuse')

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

# formatting labels
fig.autofmt_xdate()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_23.py
% ls *.png
ai202209_s04_00.png     ai202209_s04_08.png     ai202209_s04_16.png
ai202209_s04_01.png     ai202209_s04_09.png     ai202209_s04_17.png
ai202209_s04_02.png     ai202209_s04_10.png     ai202209_s04_18.png
ai202209_s04_03.png     ai202209_s04_11.png     ai202209_s04_19.png
ai202209_s04_04.png     ai202209_s04_12.png     ai202209_s04_20.png
ai202209_s04_05.png     ai202209_s04_13.png     ai202209_s04_21.png
ai202209_s04_06.png     ai202209_s04_14.png     ai202209_s04_23.png
ai202209_s04_07.png     ai202209_s04_15.png
```

Display the PNG image file. (39)

```
% feh -dF ai202209_s04_23.png
```

Try following practice.

Practice 04-23

Make your own Python script to create a plot dealing with date/time.

13.3 Changing ticks

Change ticks and make a lightcurve of Betelgeuse again. Here is an example.

Python Code 25: ai202209_s04_24.py

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

# importing numpy module
```

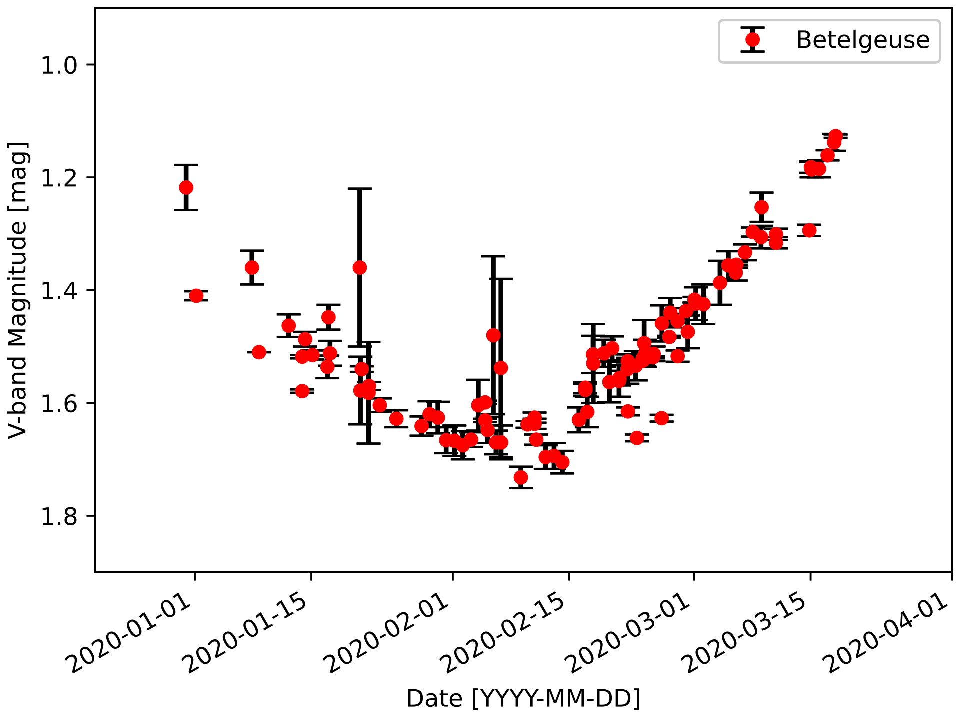


Figure 39: Plot created by the script `ai202209_s04_23.py`.

```
import numpy

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

# importing datetime
import datetime

# input data file
file_input = 'alf_ori.data'

# output image file
file_output = 'ai202209_s04_24.png'

# making empty list and Numpy arrays
data_date  = numpy.array ([], dtype='datetime64[ms]')
data_mag   = numpy.array ([], dtype='float64')
data_error = numpy.array ([], dtype='float64')

# opening input file
with open (file_input, 'r') as fh_in:
    # reading data line-by-line
    for line in fh_in:
        # splitting data
        (date_str, mag_str, error_str, band, observer) = line.split ()
        # conversion from string to datetime, and then to datetime64
```

```
        date1 = datetime.datetime.strptime (date_str[:-4], '%Y-%m-%d')
        day   = float (date_str[-3:]) / 1000
        date2 = datetime.timedelta (days=day)
        date_datetime = date1 + date2
        date_datetime64 = numpy.datetime64 (date_datetime, 'ms')
        # conversion from string to float
        mag   = float (mag_str)
        error = float (error_str)
        # appending data to list and Numpy arrays
        data_date  = numpy.append (data_date, date_datetime64)
        data_mag   = numpy.append (data_mag, mag)
        data_error = numpy.append (data_error, error)

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('Date [YYYY-MM-DD]')
ax.set_ylabel ('V-band Magnitude [mag]')

# axis settings
ax.set_xlim (numpy.datetime64 ('2019-12-20'), numpy.datetime64 ('2020-04-01'))
ax.set_ylim (+1.9, +0.9)

# plotting data
ax.errorbar (data_date, data_mag, yerr=data_error, linestyle='None', \
             marker='o', markersize=5.0, color='red', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='Betelgeuse')

# ticks
months     = matplotlib.dates.MonthLocator ()
days       = matplotlib.dates.DayLocator ()
months_fmt = matplotlib.dates.DateFormatter ('%Y-%m-%d')
ax.xaxis.set_major_locator (months)
ax.xaxis.set_major_formatter (months_fmt)
ax.xaxis.set_minor_locator (days)

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

# formatting labels
fig.autofmt_xdate()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_24.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_08.png       ai202209_s04_16.png
ai202209_s04_01.png       ai202209_s04_09.png       ai202209_s04_17.png
```

```
ai202209_s04_02.png     ai202209_s04_10.png     ai202209_s04_18.png
ai202209_s04_03.png     ai202209_s04_11.png     ai202209_s04_19.png
ai202209_s04_04.png     ai202209_s04_12.png     ai202209_s04_20.png
ai202209_s04_05.png     ai202209_s04_13.png     ai202209_s04_21.png
ai202209_s04_06.png     ai202209_s04_14.png     ai202209_s04_23.png
ai202209_s04_07.png     ai202209_s04_15.png     ai202209_s04_24.png
```

Display the PNG image file. (40)

```
% feh -dF ai202209_s04_24.png
```

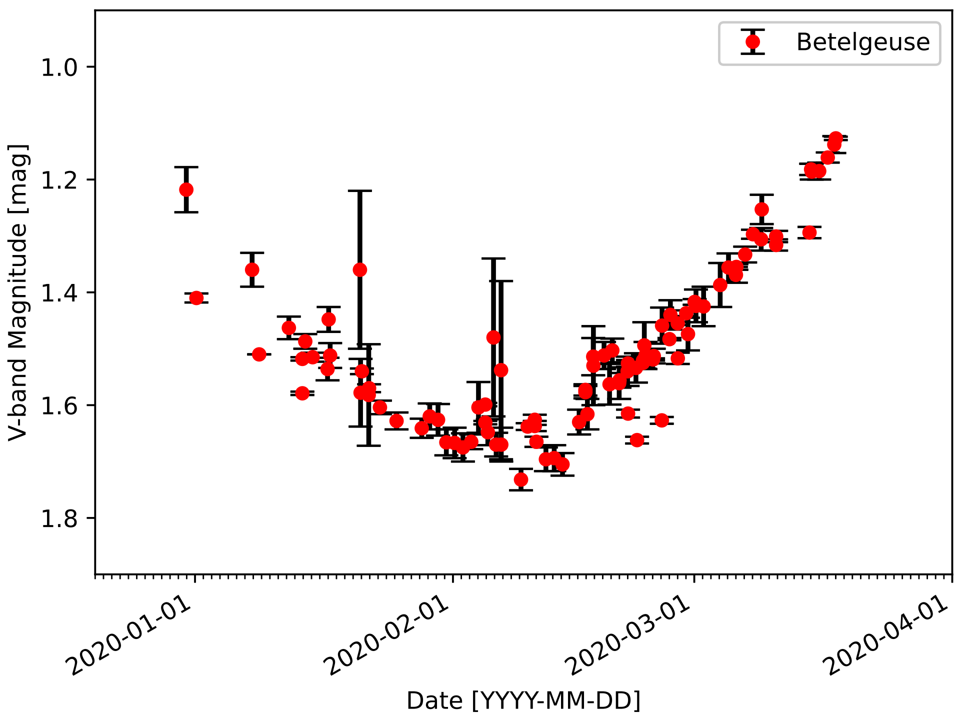


Figure 40: Plot created by the script `ai202209_s04_24.py`.

Try following practice.

Practice 04-24

Make your own Python script to create a plot dealing with date/time. Give settings for ticks.

14 Making histograms

14.1 Making a histogram using `.bar ()` method

Generate a set of random numbers of Gaussian distribution, construct a histogram, and then make a plot using `.bar ()` method. Here is an example.

Python Code 26: ai202209_s04_25.py

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

```
# importing numpy module
import numpy

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

# output image file
file_output = 'ai202209_s04_25.png'

# parameters for random number generation
mean    = 1000.0
stddev  = 100.0
n       = 10**6

# parameters for histogram
bin_min   = 500.0
bin_max   = 1500.0
bin_width = 50.0
bin_n     = int ((bin_max - bin_min) / bin_width) + 1

# generating random numbers
rng  = numpy.random.default_rng ()
dist = rng.normal (loc=mean, scale=stddev, size=n)

# initialisation of numpy arrays for histogram
hist_x = numpy.linspace (bin_min, bin_max, bin_n)
hist_y = numpy.zeros (bin_n, dtype='int64')

# construction of a histogram
for i in range (len (dist)):
    # if data is outside of [bin_min, bin_max], then skip
    if ( (dist[i] < bin_min) or (dist[i] > bin_max) ):
        continue
    # counting number of data in each bin
    hist_y[int ( (dist[i] - bin_min) / bin_width)] += 1

# printing histogram
for i in range (bin_n):
    bin_0 = bin_min + bin_width * i
    bin_1 = bin_min + bin_width * (i+1)
    print (f'{bin_0:6.1f}-{bin_1:6.1f}  {hist_y[i]:6d}')

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('$x$')
ax.set_ylabel ('Number of data')

# axis settings
```

```
ax.set_xlim (bin_min, bin_max)

# plotting data
ax.bar (hist_x, hist_y, bin_width, edgecolor='black', linewidth=0.3, \
        align='edge', label='Gaussian distribution')

# legend
ax.legend ()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_25.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_09.png       ai202209_s04_18.png
ai202209_s04_01.png       ai202209_s04_10.png       ai202209_s04_19.png
ai202209_s04_02.png       ai202209_s04_11.png       ai202209_s04_20.png
ai202209_s04_03.png       ai202209_s04_12.png       ai202209_s04_21.png
ai202209_s04_04.png       ai202209_s04_13.png       ai202209_s04_23.png
ai202209_s04_05.png       ai202209_s04_14.png       ai202209_s04_24.png
ai202209_s04_06.png       ai202209_s04_15.png       ai202209_s04_25.png
ai202209_s04_07.png       ai202209_s04_16.png
ai202209_s04_08.png       ai202209_s04_17.png
```

Display the PNG image file. (41)

```
% feh -dF ai202209_s04_25.png
```

Try following practice.

Practice 04-25

Make your own Python script to generate a set of random numbers of uniform distribution and construct a histogram using `.bar ()` method.

14.2 Making a histogram using `.hist ()` method

Generate a set of random numbers of Gaussian distribution, and then make a plot using `.hist ()` method. Here is an example.

Python Code 27: ai202209_s04_26.py

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

# importing numpy module
import numpy

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

# output image file
file_output = 'ai202209_s04_26.png'

# parameters for random number generation
```

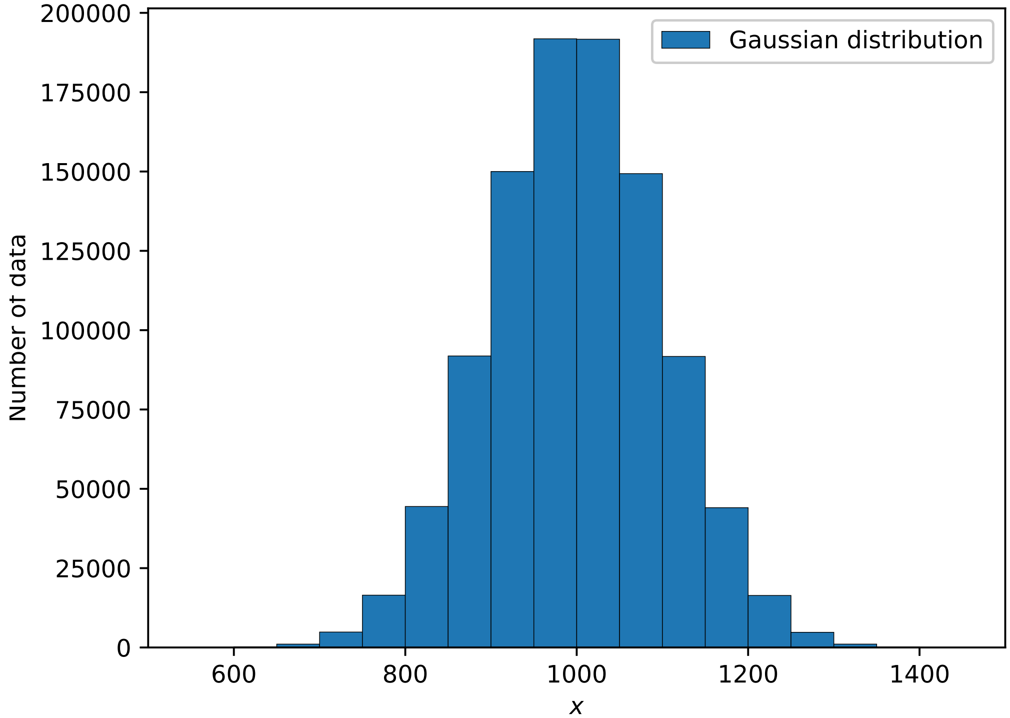


Figure 41: Plot created by the script `ai202209_s04_25.py`.

```
mean   = 1000.0
stddev = 100.0
n      = 10**7

# parameters for histogram
bin_min   = 500.0
bin_max   = 1500.0
bin_width = 20.0
bin_n     = int ((bin_max - bin_min) / bin_width) + 1

# generating random numbers
rng  = numpy.random.default_rng ()
dist = rng.normal (loc=mean, scale=stddev, size=n)

# initialisation of numpy arrays for histogram
bins = numpy.linspace (bin_min, bin_max, bin_n)

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('$x$')
```

```
ax.set_ylabel ('Number of data')

# axis settings
ax.set_xlim (bin_min, bin_max)

# plotting data
ax.hist (dist, bins=bins, histtype='bar', \
         edgecolor='black', linewidth=0.3, align='mid', \
         label='Gaussian distribution')

# legend
ax.legend ()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_26.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_09.png        ai202209_s04_18.png
ai202209_s04_01.png        ai202209_s04_10.png        ai202209_s04_19.png
ai202209_s04_02.png        ai202209_s04_11.png        ai202209_s04_20.png
ai202209_s04_03.png        ai202209_s04_12.png        ai202209_s04_21.png
ai202209_s04_04.png        ai202209_s04_13.png        ai202209_s04_23.png
ai202209_s04_05.png        ai202209_s04_14.png        ai202209_s04_24.png
ai202209_s04_06.png        ai202209_s04_15.png        ai202209_s04_25.png
ai202209_s04_07.png        ai202209_s04_16.png        ai202209_s04_26.png
ai202209_s04_08.png        ai202209_s04_17.png
```

Display the PNG image file. (42)

```
% feh -dF ai202209_s04_26.png
```

Try following practice.

Practice 04-26

Make your own Python script to generate a set of random numbers of uniform distribution and construct a histogram using `.hist ()` method.

15 Making a scatter plot

Download Yale Bright Star Catalogue, and make a HR diagram.

15.1 Downloading Yale Bright Star Catalogue

Make a Python script to download Yale Bright Star Catalogue.

Python Code 28: ai202209_s04_27.py

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

#
# Time-stamp: <2022/10/02 17:58:08 (CST) daisuke>
#
```

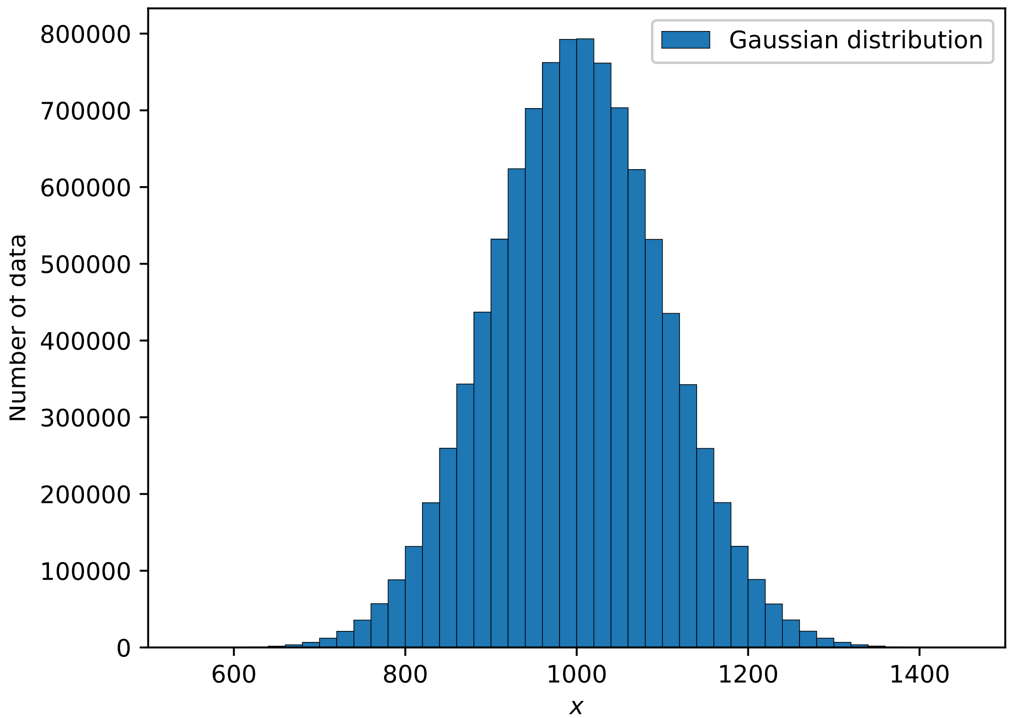


Figure 42: Plot created by the script ai202209_s04_26.py.

```
# 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://cdsarc.cds.unistra.fr/ftp/V/50/catalog.gz'

# output file name
file_output = 'catalog.gz'

# printing status
print (f'Now, fetching the file {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 the file {url_data}!')

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

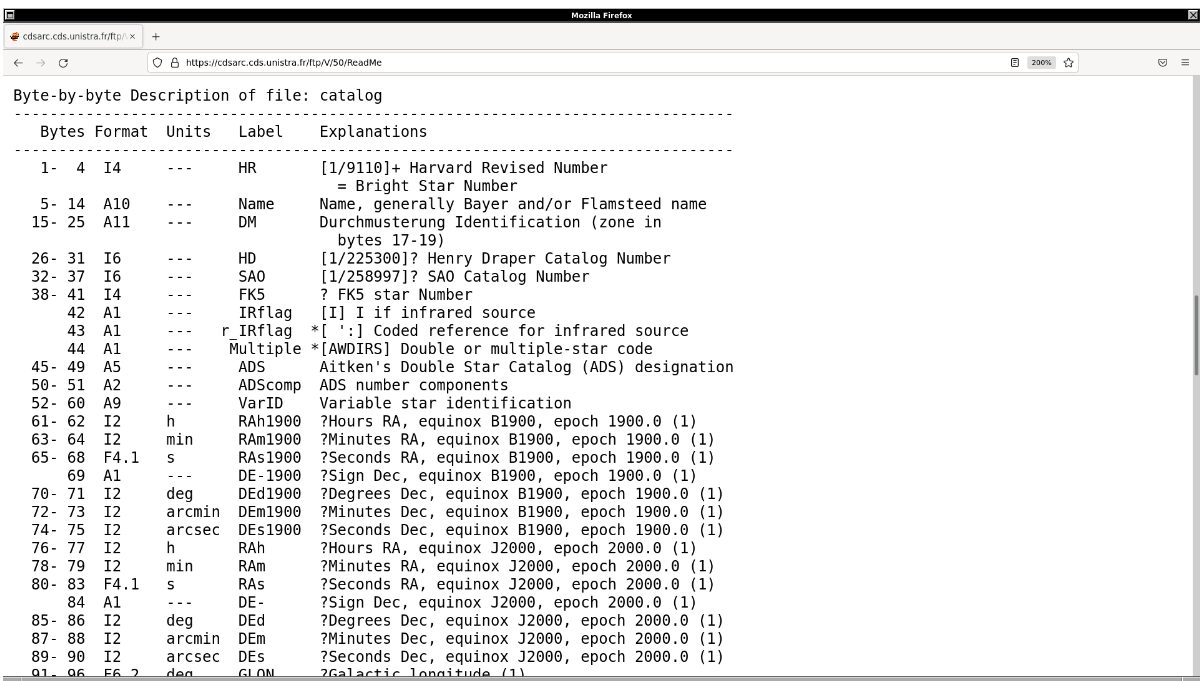
```
# opening file for writing
with open (file_output , 'wb') as fh_write:
    # writing data
    fh_write.write (data_byte)

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

Execute above script.

```
% ./ai202209_s04_27.py
% ls -l catalog.gz
-rw-r--r--  1 daisuke  taiwan  573921 Oct  2 17:58 catalog.gz
```

To learn about the format of the catalogue file, read following file. (Fig. 43)

- https://cdsarc.cds.unistra.fr/ftp/V/50/ReadMe



```
Byte-by-byte Description of file: catalog
--------------------------------------------------------------------------------
   Bytes Format  Units   Label    Explanations
--------------------------------------------------------------------------------
   1-  4  I4     ---     HR       [1/9110]+ Harvard Revised Number
                                    = Bright Star Number
   5- 14  A10    ---     Name     Name, generally Bayer and/or Flamsteed name
  15- 25  A11    ---     DM       Durchmusterung Identification (zone in
                                    bytes 17-19)
  26- 31  I6     ---     HD       [1/225300]? Henry Draper Catalog Number
  32- 37  I6     ---     SAO      [1/258997]? SAO Catalog Number
  38- 41  I4     ---     FK5      ? FK5 star Number
      42  A1     ---     IRflag   [I] I if infrared source
      43  A1     ---   r_IRflag  *[ ':] Coded reference for infrared source
      44  A1     ---    Multiple *[AWDIRS] Double or multiple-star code
  45- 49  A5     ---     ADS      Aitken's Double Star Catalog (ADS) designation
  50- 51  A2     ---     ADScomp  ADS number components
  52- 60  A9     ---     VarID    Variable star identification
  61- 62  I2     h       RAh1900  ?Hours RA, equinox B1900, epoch 1900.0 (1)
  63- 64  I2     min     RAm1900  ?Minutes RA, equinox B1900, epoch 1900.0 (1)
  65- 68  F4.1   s       RAs1900  ?Seconds RA, equinox B1900, epoch 1900.0 (1)
      69  A1     ---     DE-1900  ?Sign Dec, equinox B1900, epoch 1900.0 (1)
  70- 71  I2     deg     DEd1900  ?Degrees Dec, equinox B1900, epoch 1900.0 (1)
  72- 73  I2     arcmin  DEm1900  ?Minutes Dec, equinox B1900, epoch 1900.0 (1)
  74- 75  I2     arcsec  DEs1900  ?Seconds Dec, equinox B1900, epoch 1900.0 (1)
  76- 77  I2     h       RAh      ?Hours RA, equinox J2000, epoch 2000.0 (1)
  78- 79  I2     min     RAm      ?Minutes RA, equinox J2000, epoch 2000.0 (1)
  80- 83  F4.1   s       RAs      ?Seconds RA, equinox J2000, epoch 2000.0 (1)
      84  A1     ---     DE-      ?Sign Dec, equinox J2000, epoch 2000.0 (1)
  85- 86  I2     deg     DEd      ?Degrees Dec, equinox J2000, epoch 2000.0 (1)
  87- 88  I2     arcmin  DEm      ?Minutes Dec, equinox J2000, epoch 2000.0 (1)
  89- 90  I2     arcsec  DEs      ?Seconds Dec, equinox J2000, epoch 2000.0 (1)
  91- 96  F6.2   deg     GLON     ?Galactic longitude (1)
```

Figure 43: The ReadMe file of Yale Bright Star Catalogue.

15.2 Reading information from the catalogue file

Make a Python script to read information from the catalogue file.

Python Code 29: ai202209_s04_28.py

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

#
# Time-stamp: <2022/10/02 18:21:20 (CST) daisuke>
#

# importing gzip module
import gzip
```

```
# importing numpy module
import numpy

# catalogue file name
file_catalogue = 'catalog.gz'

# dictionary for storing data
stars = {}

# opening catalogue file
with gzip.open (file_catalogue, 'rb') as fh:
    # reading catalogue line-by-line
    for line in fh:
        # Harvard Revised Number of star
        HR = line[0:4].strip ()
        # name
        name = line[4:14].strip ()
        # Vmag
        mag_V = line[102:107].strip ()
        # B-V colour
        colour_BV = line[109:114].strip ()
        # spectral type
        sptype = line[127:147].strip ()
        # dynamical parallax flag
        dynamical_parallax = line[160]
        # parallax
        parallax = line[161:166]

        # skip, if any of mag_V, colour_BV, parallax is missing
        if ( (mag_V == '') or (colour_BV == '') or (parallax == '') ):
            continue
        # skip, if parallax is dynamical parallax
        if (dynamical_parallax == 'D'):
            continue
        # reformat parallax
        if (parallax[:2] == '+.'):
            parallax = '+0.' + parallax[2:]

        # conversion from string to float
        try:
            mag_V     = float (mag_V)
        except:
            continue
        try:
            colour_BV = float (colour_BV)
        except:
            continue
        try:
            parallax  = float (parallax)
        except:
            continue

        # skip, if parallax is negative
        if (parallax < 0.0):
            continue

        # skip, if parallax is zero
        if (parallax < 10**-4):
```

```
            continue

        # distance in parsec
        dist_pc = 1.0 / parallax

        # absolute magnitude
        absmag_V = mag_V - 5.0 * numpy.log10 (dist_pc) + 5.0

        # constructing the dictionary
        stars[HR] = {}
        stars[HR]["mag_V"]      = mag_V
        stars[HR]["colour_BV"] = colour_BV
        stars[HR]["parallax"]  = parallax
        stars[HR]["dist_pc"]   = dist_pc
        stars[HR]["absmag_V"]  = absmag_V
        stars[HR]["sptype"]    = sptype
        stars[HR]["name"]      = name

# printing header
print ("# Vmag, (B-V), parallax, distance, absmag_V, HR, name")

# printing information of 1st mag stars
for key, value in sorted (stars.items (), key=lambda x: x[1]['mag_V']):
    if (stars[key]['mag_V'] >= 1.5):
        break
    print (f'{stars[key]["mag_V"]:+6.3f} ', \
           f'{stars[key]["colour_BV"]:+6.3f} ', \
           f'{stars[key]["parallax"]:+6.3f} ', \
           f'{stars[key]["dist_pc"]:+8.3f} ', \
           f'{stars[key]["absmag_V"]:+6.3f} ', \
           f'{int (key.decode ("utf-8")):4d} ', \
           f'{stars[key]["name"].decode ("utf-8")}')
```

Execute above script.

```
% ./ai202209_s04_28.py
# Vmag, (B-V), parallax, distance, absmag_V, HR, name
-1.460  +0.000  +0.375    +2.667  +1.410  2491  9Alp CMa
-0.720  +0.150  +0.028   +35.714  -3.484  2326  Alp Car
-0.040  +1.230  +0.090   +11.111  -0.269  5340  16Alp Boo
-0.010  +0.710  +0.751    +1.332  +4.368  5459  Alp1Cen
+0.030  +0.000  +0.123    +8.130  +0.480  7001  3Alp Lyr
+0.080  +0.800  +0.073   +13.699  -0.603  1708  13Alp Aur
+0.120  -0.030  +0.013   +76.923  -4.310  1713  19Bet Ori
+0.380  +0.420  +0.288    +3.472  +2.677  2943  10Alp CMi
+0.460  -0.160  +0.026   +38.462  -2.465   472  Alp Eri
+0.500  +1.850  +0.005  +200.000  -6.005  2061  58Alp Ori
+0.610  -0.230  +0.009  +111.111  -4.619  5267  Bet Cen
+0.770  +0.220  +0.198    +5.051  +2.253  7557  53Alp Aql
+0.850  +1.540  +0.048   +20.833  -0.744  1457  87Alp Tau
+0.960  +1.830  +0.024   +41.667  -2.139  6134  21Alp Sco
+0.980  -0.230  +0.023   +43.478  -2.211  5056  67Alp Vir
+1.140  +1.000  +0.094   +10.638  +1.006  2990  78Bet Gem
+1.160  +0.090  +0.149    +6.711  +2.026  8728  24Alp PsA
+1.330  -0.240  +0.008  +125.000  -4.155  4730  Alp1Cru
+1.330  +0.880  +0.751    +1.332  +5.708  5460  Alp2Cen
+1.350  -0.110  +0.045   +22.222  -0.384  3982  32Alp Leo
```

15.3 Making HR diagram of the brightest stars

Make a Python script to generate a HR diagram of the brightest stars. Here is an example.

Python Code 30: ai202209_s04_29.py

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

#
# Time-stamp: <2022/10/02 18:51:23 (CST) daisuke>
#

# importing gzip module
import gzip

# importing numpy module
import numpy

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

# catalogue file name
file_catalogue = 'catalog.gz'

# output file name
file_output = 'ai202209_s04_29.png'

# dictionary for storing data
stars = {}

# opening catalogue file
with gzip.open (file_catalogue, 'rb') as fh:
    # reading catalogue line-by-line
    for line in fh:
        # Harvard Revised Number of star
        HR = line[0:4].strip ()
        # name
        name = line[4:14].strip ()
        # Vmag
        mag_V = line[102:107].strip ()
        # B-V colour
        colour_BV = line[109:114].strip ()
        # spectral type
        sptype = line[127:147].strip ()
        # dynamical parallax flag
        dynamical_parallax = line[160]
        # parallax
        parallax = line[161:166]

        # skip, if any of mag_V, colour_BV, parallax is missing
        if ( (mag_V == '') or (colour_BV == '') or (parallax == '') ):
            continue
        # skip, if parallax is dynamical parallax
        if (dynamical_parallax == 'D'):
            continue
        # reformat parallax
        if (parallax[:2] == '+.'):
            parallax = '+0.' + parallax[2:]

        # conversion from string to float
```

```
        try:
            mag_V      = float (mag_V)
        except:
            continue
        try:
            colour_BV = float (colour_BV)
        except:
            continue
        try:
            parallax  = float (parallax)
        except:
            continue

        # skip, if parallax is negative
        if (parallax < 0.0):
            continue

        # skip, if parallax is zero
        if (parallax < 10**-4):
            continue

        # distance in parsec
        dist_pc = 1.0 / parallax

        # absolute magnitude
        absmag_V = mag_V - 5.0 * numpy.log10 (dist_pc) + 5.0

        # constructing the dictionary
        stars[HR] = {}
        stars[HR]["mag_V"]      = mag_V
        stars[HR]["colour_BV"] = colour_BV
        stars[HR]["parallax"]  = parallax
        stars[HR]["dist_pc"]   = dist_pc
        stars[HR]["absmag_V"]  = absmag_V
        stars[HR]["sptype"]    = sptype
        stars[HR]["name"]      = name

# making empty numpy arrays for plotting
colour = numpy.array ([])
absmag = numpy.array ([])

# printing header
print ("# Vmag, (B-V), parallax, distance, absmag_V, HR, name")

# printing information of 1st mag stars
for key, value in sorted (stars.items (), key=lambda x: x[1]['mag_V']):
    # if mag of star is equal to or greater than 1.5, then skip
    if (stars[key]['mag_V'] >= 1.5):
        break
    # printing information
    print (f'{stars[key]["mag_V"]:+6.3f} ', \
           f'{stars[key]["colour_BV"]:+6.3f} ', \
           f'{stars[key]["parallax"]:+6.3f} ', \
           f'{stars[key]["dist_pc"]:+8.3f} ', \
           f'{stars[key]["absmag_V"]:+6.3f} ', \
           f'{int (key.decode ("utf-8")):4d} ', \
           f'{stars[key]["name"].decode ("utf-8")}')
    # appending data into numpy arrays
    colour = numpy.append (colour, stars[key]['colour_BV'])
```

```
    absmag = numpy.append (absmag, stars[key]['absmag_V'])

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('$(B-V)$ Colour Index')
ax.set_ylabel ('Absolute Magnitude')

# flipping direction of Y-axis
ax.invert_yaxis ()

# plotting data
ax.plot (colour, absmag, linestyle='None', marker='o', color='red', \
         label='Bright stars')

# grid
ax.grid ()

# legend
ax.legend ()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_29.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_09.png        ai202209_s04_18.png
ai202209_s04_01.png        ai202209_s04_10.png        ai202209_s04_19.png
ai202209_s04_02.png        ai202209_s04_11.png        ai202209_s04_20.png
ai202209_s04_03.png        ai202209_s04_12.png        ai202209_s04_21.png
ai202209_s04_04.png        ai202209_s04_13.png        ai202209_s04_23.png
ai202209_s04_05.png        ai202209_s04_14.png        ai202209_s04_24.png
ai202209_s04_06.png        ai202209_s04_15.png        ai202209_s04_25.png
ai202209_s04_07.png        ai202209_s04_16.png        ai202209_s04_26.png
ai202209_s04_08.png        ai202209_s04_17.png        ai202209_s04_29.png
```

Display the PNG image file. (44)

```
% feh -dF ai202209_s04_29.png
```

Try following practice.

Practice 04-27

Modify the sample script "`ai202209_s04_29.py`" and change colour and marker shape.

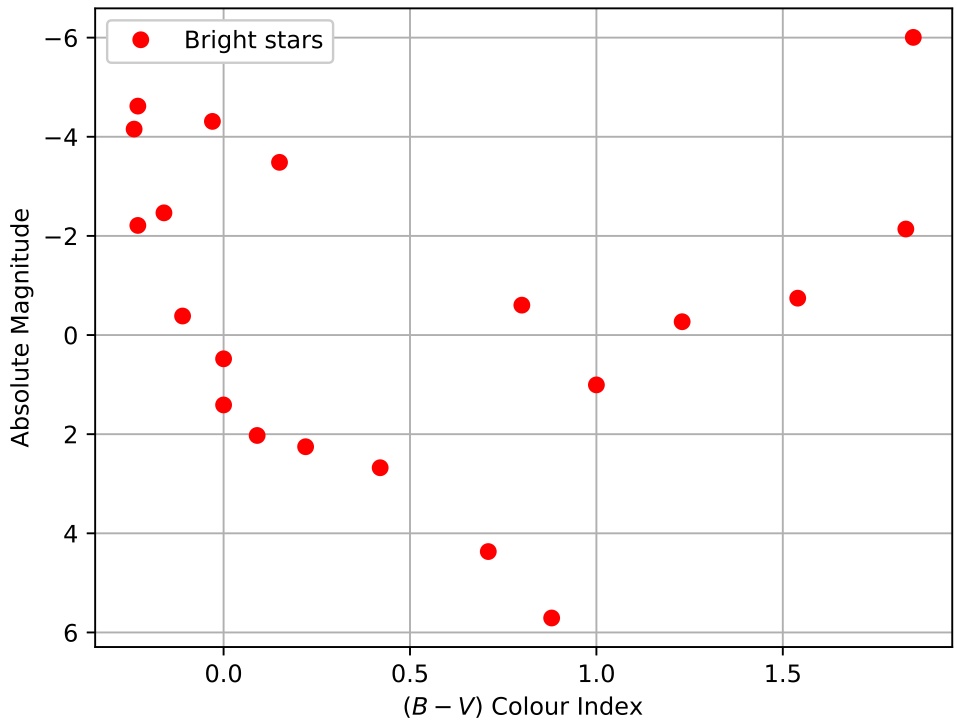


Figure 44: Plot created by the script `ai202209_s04_29.py`.

15.4 Changing marker colour

Change marker colour for different stars. Here is an example.

Python Code 31: ai202209_s04_30.py

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

#
# Time-stamp: <2022/10/02 18:51:43 (CST) daisuke>
#

# importing gzip module
import gzip

# importing numpy module
import numpy

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

# catalogue file name
file_catalogue = 'catalog.gz'

# output file name
file_output = 'ai202209_s04_30.png'

# dictionary for storing data
stars = {}
```

```
# opening catalogue file
with gzip.open (file_catalogue, 'rb') as fh:
    # reading catalogue line-by-line
    for line in fh:
        # Harvard Revised Number of star
        HR = line[0:4].strip ()
        # name
        name = line[4:14].strip ()
        # Vmag
        mag_V = line[102:107].strip ()
        # B-V colour
        colour_BV = line[109:114].strip ()
        # spectral type
        sptype = line[127:147].strip ()
        # dynamical parallax flag
        dynamical_parallax = line[160]
        # parallax
        parallax = line[161:166]

        # skip, if any of mag_V, colour_BV, parallax is missing
        if ( (mag_V == '') or (colour_BV == '') or (parallax == '') ):
            continue
        # skip, if parallax is dynamical parallax
        if (dynamical_parallax == 'D'):
            continue
        # reformat parallax
        if (parallax[:2] == '+.'):
            parallax = '+0.' + parallax[2:]

        # conversion from string to float
        try:
            mag_V     = float (mag_V)
        except:
            continue
        try:
            colour_BV = float (colour_BV)
        except:
            continue
        try:
            parallax  = float (parallax)
        except:
            continue

        # skip, if parallax is negative
        if (parallax < 0.0):
            continue

        # skip, if parallax is zero
        if (parallax < 10**-4):
            continue

        # distance in parsec
        dist_pc = 1.0 / parallax

        # absolute magnitude
        absmag_V = mag_V - 5.0 * numpy.log10 (dist_pc) + 5.0

        # constructing the dictionary
```

```
        stars[HR] = {}
        stars[HR]["mag_V"]      = mag_V
        stars[HR]["colour_BV"] = colour_BV
        stars[HR]["parallax"]  = parallax
        stars[HR]["dist_pc"]   = dist_pc
        stars[HR]["absmag_V"]  = absmag_V
        stars[HR]["sptype"]    = sptype
        stars[HR]["name"]      = name

# making empty numpy arrays for plotting
colour = numpy.array ([])
absmag = numpy.array ([])
label  = numpy.array ([], dtype=str)

# printing header
print ("# Vmag, (B-V), parallax, distance, absmag_V, HR, name")

# printing information of 1st mag stars
for key, value in sorted (stars.items (), key=lambda x: x[1]['mag_V']):
    # if mag of star is equal to or greater than 1.5, then skip
    if (stars[key]['mag_V'] >= 1.5):
        break
    # printing information
    print (f'{stars[key]["mag_V"]:+6.3f} ', \
           f'{stars[key]["colour_BV"]:+6.3f} ', \
           f'{stars[key]["parallax"]:+6.3f} ', \
           f'{stars[key]["dist_pc"]:+8.3f} ', \
           f'{stars[key]["absmag_V"]:+6.3f} ', \
           f'{int (key.decode ("utf-8")):4d} ', \
           f'{stars[key]["name"].decode ("utf-8")}')
    # appending data into numpy arrays
    colour = numpy.append (colour, stars[key]['colour_BV'])
    absmag = numpy.append (absmag, stars[key]['absmag_V'])
    label  = numpy.append (label, f'{stars[key]["name"].decode ("utf-8")}')

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# adjustment of plot
box = ax.get_position ()
ax.set_position ([box.x0, box.y0, box.width * 0.8, box.height])

# labels
ax.set_xlabel ('$(B-V)$ Colour Index')
ax.set_ylabel ('Absolute Magnitude')

# flipping direction of Y-axis
ax.invert_yaxis ()

# plotting data
for i in range (len (colour)):
    size = 15 - i * 0.5
    ax.plot (colour[i], absmag[i], linestyle='None', marker='o', \
```

```
              markersize=size, label=label[i])

# grid
ax.grid ()

# legend
ax.legend (bbox_to_anchor=(1.0, 1.05), loc='upper left')

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_30.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_10.png        ai202209_s04_20.png
ai202209_s04_01.png        ai202209_s04_11.png        ai202209_s04_21.png
ai202209_s04_02.png        ai202209_s04_12.png        ai202209_s04_23.png
ai202209_s04_03.png        ai202209_s04_13.png        ai202209_s04_24.png
ai202209_s04_04.png        ai202209_s04_14.png        ai202209_s04_25.png
ai202209_s04_05.png        ai202209_s04_15.png        ai202209_s04_26.png
ai202209_s04_06.png        ai202209_s04_16.png        ai202209_s04_29.png
ai202209_s04_07.png        ai202209_s04_17.png        ai202209_s04_30.png
ai202209_s04_08.png        ai202209_s04_18.png
ai202209_s04_09.png        ai202209_s04_19.png
```

Display the PNG image file. (45)

```
% feh -dF ai202209_s04_30.png
```

Try following practice.

Practice 04-28

Modify the sample script "`ai202209_s04_30.py`" and change marker shape.

15.5 Making a scatter plot

Make HR diagram of the brightest stars as a scatter plot. Here is an example.

Python Code 32: ai202209_s04_31.py

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

#
# Time-stamp: <2022/10/02 19:33:04 (CST) daisuke>
#

# importing gzip module
import gzip

# importing numpy module
import numpy

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

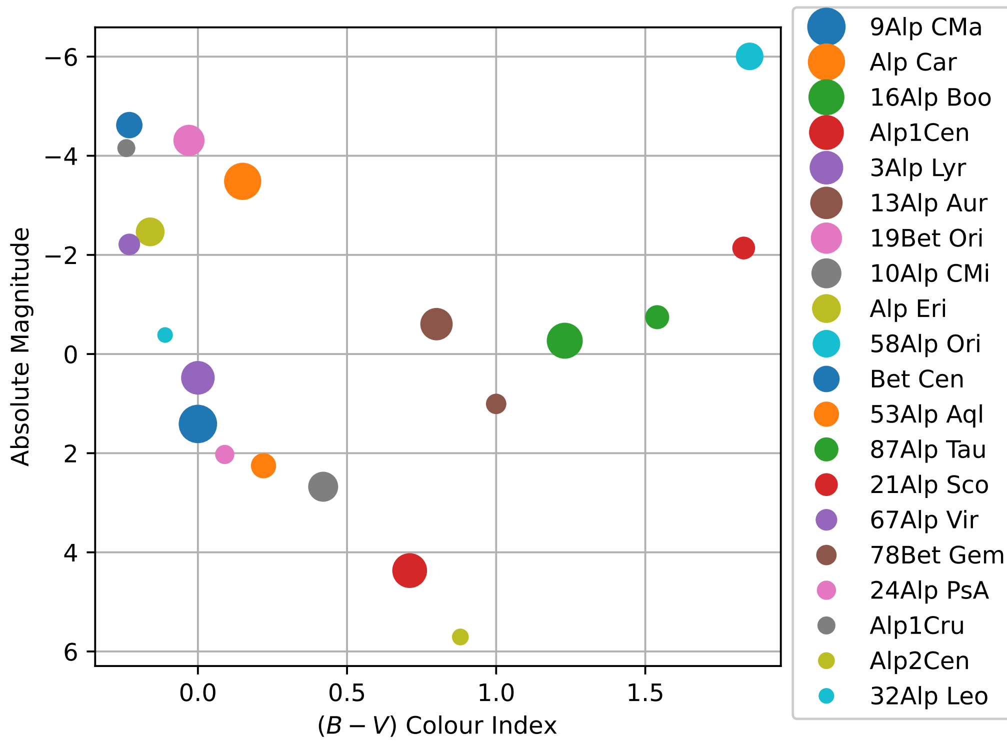


Figure 45: Plot created by the script `ai202209_s04_30.py`.

```
# catalogue file name
file_catalogue = 'catalog.gz'

# output file name
file_output = 'ai202209_s04_31.png'

# dictionary for storing data
stars = {}

# opening catalogue file
with gzip.open (file_catalogue, 'rb') as fh:
    # reading catalogue line-by-line
    for line in fh:
        # Harvard Revised Number of star
        HR = line[0:4].strip ()
        # name
        name = line[4:14].strip ()
        # Vmag
        mag_V = line[102:107].strip ()
        # B-V colour
        colour_BV = line[109:114].strip ()
        # spectral type
        sptype = line[127:147].strip ()
        # dynamical parallax flag
        dynamical_parallax = line[160]
        # parallax
        parallax = line[161:166]
```

```
        # skip, if any of mag_V, colour_BV, parallax is missing
        if ( (mag_V == '') or (colour_BV == '') or (parallax == '') ):
            continue
        # skip, if parallax is dynamical parallax
        if (dynamical_parallax == 'D'):
            continue
        # reformat parallax
        if (parallax[:2] == '+.'):
            parallax = '+0.' + parallax[2:]

        # conversion from string to float
        try:
            mag_V     = float (mag_V)
        except:
            continue
        try:
            colour_BV = float (colour_BV)
        except:
            continue
        try:
            parallax  = float (parallax)
        except:
            continue

        # skip, if parallax is negative
        if (parallax < 0.0):
            continue

        # skip, if parallax is zero
        if (parallax < 10**-4):
            continue

        # distance in parsec
        dist_pc = 1.0 / parallax

        # absolute magnitude
        absmag_V = mag_V - 5.0 * numpy.log10 (dist_pc) + 5.0

        # constructing the dictionary
        stars[HR] = {}
        stars[HR]["mag_V"]     = mag_V
        stars[HR]["colour_BV"] = colour_BV
        stars[HR]["parallax"]  = parallax
        stars[HR]["dist_pc"]   = dist_pc
        stars[HR]["absmag_V"]  = absmag_V
        stars[HR]["sptype"]    = sptype
        stars[HR]["name"]      = name

# making empty numpy arrays for plotting
colour = numpy.array ([])
appmag = numpy.array ([])
absmag = numpy.array ([])

# printing header
print ("# Vmag, (B-V), parallax, distance, absmag_V, HR, name")

# printing information of 1st mag stars
for key, value in sorted (stars.items (), key=lambda x: x[1]['mag_V']):
    # if colour index of star is greater than 2.5, then skip
```

```
    if (stars[key]['colour_BV'] > 2.5):
        break
    # printing information
    print (f'{stars[key]["mag_V"]:+6.3f} ', \
           f'{stars[key]["colour_BV"]:+6.3f} ', \
           f'{stars[key]["parallax"]:+6.3f} ', \
           f'{stars[key]["dist_pc"]:+8.3f} ', \
           f'{stars[key]["absmag_V"]:+6.3f} ', \
           f'{int (key.decode ("utf-8")):4d} ', \
           f'{stars[key]["name"].decode ("utf-8")}')
    # appending data into numpy arrays
    colour = numpy.append (colour, stars[key]['colour_BV'])
    appmag = numpy.append (appmag, stars[key]['mag_V'])
    absmag = numpy.append (absmag, stars[key]['absmag_V'])

# marker size and colour
marker_size   = 100.0 / (appmag + 2.0)**2
marker_colour = 4.0 - colour

# making a fig object
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# labels
ax.set_xlabel ('$(B-V)$ Colour Index')
ax.set_ylabel ('Absolute Magnitude')

# flipping direction of Y-axis
ax.invert_yaxis ()

# plotting data
ax.scatter (colour, absmag, s=marker_size, c=marker_colour, cmap='Spectral')

# grid
ax.grid ()

# saving the figure to a file
fig.savefig (file_output)
```

Execute above script.

```
% ./ai202209_s04_31.py
% ls *.png
ai202209_s04_00.png      ai202209_s04_10.png      ai202209_s04_20.png
ai202209_s04_01.png      ai202209_s04_11.png      ai202209_s04_21.png
ai202209_s04_02.png      ai202209_s04_12.png      ai202209_s04_23.png
ai202209_s04_03.png      ai202209_s04_13.png      ai202209_s04_24.png
ai202209_s04_04.png      ai202209_s04_14.png      ai202209_s04_25.png
ai202209_s04_05.png      ai202209_s04_15.png      ai202209_s04_26.png
ai202209_s04_06.png      ai202209_s04_16.png      ai202209_s04_29.png
ai202209_s04_07.png      ai202209_s04_17.png      ai202209_s04_30.png
ai202209_s04_08.png      ai202209_s04_18.png      ai202209_s04_31.png
ai202209_s04_09.png      ai202209_s04_19.png
```

Display the PNG image file. (46)

```
% feh -dF ai202209_s04_31.png
```

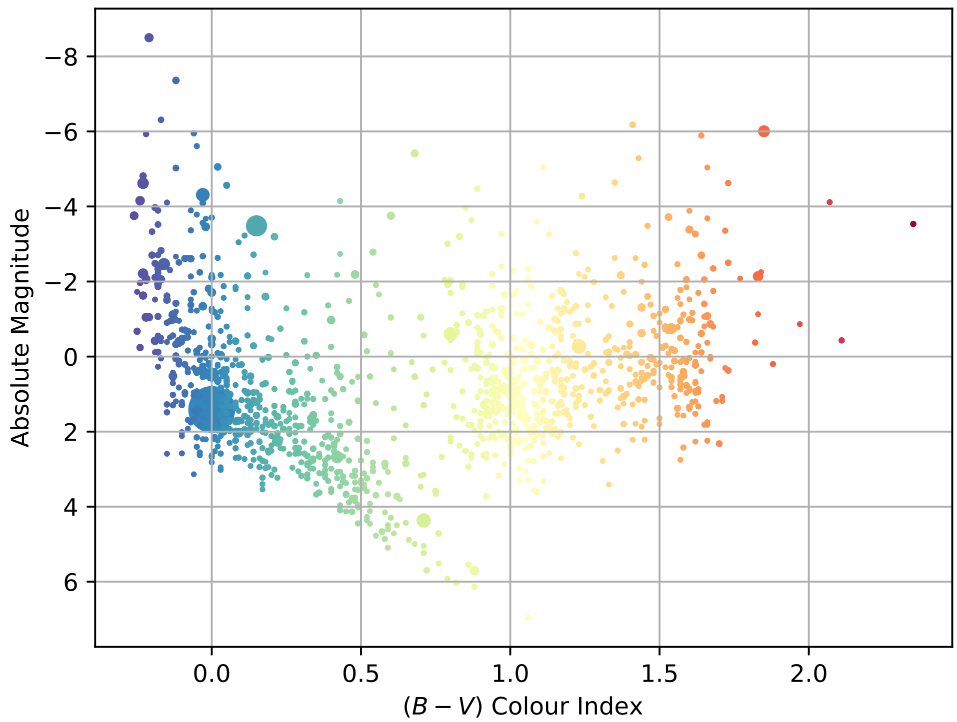


Figure 46: Plot created by the script `ai202209_s04_31.py`.

Try following practice.

Practice 04-29

Modify the sample script "`ai202209_s04_31.py`" and change marker size, marker colour, and colour map.

16 Animation

Animation can also be created by Matplotlib.

16.1 A simple animation example

First, make a plot of an ellipse and a point.

Python Code 33: ai202209_s04_32.py

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

#
# Time-stamp: <2022/10/02 20:41:06 (CST) daisuke>
#

# importing numpy module
import numpy
```

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

# output image file
file_image = 'ai202209_s04_32.png'

# an ellipse
theta = numpy.linspace (0.0, 2.0 * numpy.pi, 10**4)
ellipse_x = 5.0 * numpy.cos (theta)
ellipse_y = 3.0 * numpy.sin (theta)

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# plotting ellipse
ax.plot (ellipse_x, ellipse_y, linestyle='-', linewidth=3.0, color='black', \
         label='Ellipse')

# range of plot
ax.set_xlim (-6.0, +6.0)
ax.set_ylim (-6.0, +6.0)

# plotting a point
x = numpy.deg2rad (45.0)
ax.plot (5.0 * numpy.cos (x), 3.0 * numpy.sin (x), linestyle='None', \
         color='red', marker='o', markersize=15.0, label='Point')

# aspect of plot
ax.set_aspect ('equal')

# saving file
fig.savefig (file_image)
```

Execute above script.

```
% ./ai202209_s04_32.py
% ls *.png
ai202209_s04_00.png       ai202209_s04_10.png       ai202209_s04_20.png
ai202209_s04_01.png       ai202209_s04_11.png       ai202209_s04_21.png
ai202209_s04_02.png       ai202209_s04_12.png       ai202209_s04_23.png
ai202209_s04_03.png       ai202209_s04_13.png       ai202209_s04_24.png
ai202209_s04_04.png       ai202209_s04_14.png       ai202209_s04_25.png
ai202209_s04_05.png       ai202209_s04_15.png       ai202209_s04_26.png
ai202209_s04_06.png       ai202209_s04_16.png       ai202209_s04_29.png
ai202209_s04_07.png       ai202209_s04_17.png       ai202209_s04_30.png
ai202209_s04_08.png       ai202209_s04_18.png       ai202209_s04_31.png
ai202209_s04_09.png       ai202209_s04_19.png       ai202209_s04_32.png
```

Display the PNG image file. (47)

```
% feh -dF ai202209_s04_32.png
```

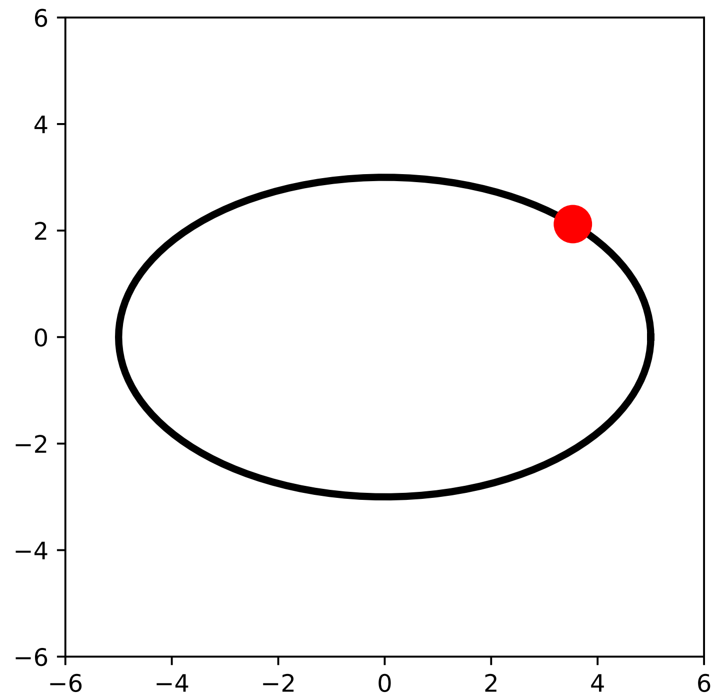


Figure 47: Plot created by the script ai202209_s04_32.py.

Here is an example of simple animation.

Python Code 34: ai202209_s04_33.py

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

#
# Time-stamp: <2022/10/02 20:48:07 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output animation file
file_anim  = 'ai202209_s04_33.mp4'

# an ellipse
theta = numpy.linspace (0.0, 2.0 * numpy.pi, 10**4)
ellipse_x = 5.0 * numpy.cos (theta)
ellipse_y = 3.0 * numpy.sin (theta)
```

```
# number of frames for animation
n_frame = 600

# an empty list for storing frames for animation
list_frame = []

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

for i in range (n_frame):
    # initialisation of object list for plotting
    list_obj = []

    # plotting ellipse
    ellipse, = ax.plot (ellipse_x, ellipse_y, linestyle='-', linewidth=3.0, \
                        color='black', label='Ellipse')
    list_obj.append (ellipse)

    # range of plot
    ax.set_xlim (-6.0, +6.0)
    ax.set_ylim (-6.0, +6.0)

    # plotting a point
    x = numpy.deg2rad (i / n_frame * 720.0)
    point, = ax.plot (5.0 * numpy.cos (x), 3.0 * numpy.sin (x), \
                      linestyle='None', color='red', marker='o', \
                      markersize=15.0, label='Point')
    list_obj.append (point)

    # aspect of plot
    ax.set_aspect ('equal')

    # appending frame
    list_frame.append (list_obj)

# making animation
anim = matplotlib.animation.ArtistAnimation (fig, list_frame, interval=50)

# saving file
anim.save (file_anim)
```

Execute above script.

```
% ./ai202209_s04_33.py
% ls -l *.mp4
-rw-r--r--  1 daisuke  taiwan  76168 Oct  2 20:48 ai202209_s04_33.mp4
```

Play the movie file.

```
% mplayer ai202209_s04_33.mp4
```

Try following practice.

Practice 04-30

Make your own animation using Matplotlib.

16.2 Visualisation of orbital motion of planets

Make a Python script to visualise orbital motion of planets in solar system.

16.2.1 Downloading positions of planets

Download positions of the Sun and four terrestrial planets.

Python Code 35: ai202209_s04_34.py

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

#
# Time-stamp: <2022/10/02 21:11:53 (CST) daisuke>
#

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

# setting for solar system ephemeris
astropy.coordinates.solar_system_ephemeris.set ('jpl')

# time t = 2022-10-03T00:00:00 (UTC)
t = astropy.time.Time ('2022-10-03T00:00:00', format='isot', scale='utc')

# getting positions of Sun, Mercury, Venus, Earth, and Mars
sun     = astropy.coordinates.get_body_barycentric ('sun', t)
mercury = astropy.coordinates.get_body_barycentric ('mercury', t)
venus   = astropy.coordinates.get_body_barycentric ('venus', t)
earth   = astropy.coordinates.get_body_barycentric ('earth', t)
mars    = astropy.coordinates.get_body_barycentric ('mars', t)

# printing positions of the Sun and planets
print (f'Positions of the Sun and the planets at t = {t}')
print (f'  Sun     : {sun}')
print (f'  Mercury : {mercury}')
print (f'  Venus   : {venus}')
print (f'  Earth   : {earth}')
print (f'  Mars    : {mars}')
```

Execute above script.

```
% ./ai202209_s04_34.py
Positions of the Sun and the planets at t = 2022-10-03T00:00:00.000
  Sun     : (-1359606.91092811, 112692.44733626, 82143.21768793) km
  Mercury : (26672328.72021633, 34198126.43851055, 15385099.0063862) km
  Venus   : (-1.08730515e+08, 2682864.78820182, 8031811.81225642) km
  Earth   : (1.46318963e+08, 22695522.16446194, 9871115.46977863) km
  Mars    : (1.6551646e+08, 1.27725976e+08, 54113213.07823729) km
```

Use au (astronomical unit) to show the positions of planets.

Python Code 36: ai202209_s04_35.py

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

#
# Time-stamp: <2022/10/02 21:14:30 (CST) daisuke>
#

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

# units
u_au = astropy.units.au

# setting for solar system ephemeris
astropy.coordinates.solar_system_ephemeris.set ('jpl')

# time t = 2022-10-03T00:00:00 (UTC)
t = astropy.time.Time ('2022-10-03T00:00:00', format='isot', scale='utc')

# getting positions of Sun, Earth, and Moon
sun     = astropy.coordinates.get_body_barycentric ('sun', t)
mercury = astropy.coordinates.get_body_barycentric ('mercury', t)
venus   = astropy.coordinates.get_body_barycentric ('venus', t)
earth   = astropy.coordinates.get_body_barycentric ('earth', t)
mars    = astropy.coordinates.get_body_barycentric ('mars', t)

# printing positions of the Sun and planets
print (f'Positions of the Sun and the planets at t = {t}')
print (f'  Sun:')
print (f'    X = {sun.x} = {sun.x.to (u_au)}')
print (f'    Y = {sun.y} = {sun.y.to (u_au)}')
print (f'    Z = {sun.z} = {sun.z.to (u_au)}')
print (f'  Mercury:')
print (f'    X = {mercury.x} = {mercury.x.to (u_au)}')
print (f'    Y = {mercury.y} = {mercury.y.to (u_au)}')
print (f'    Z = {mercury.z} = {mercury.z.to (u_au)}')
print (f'  Venus:')
print (f'    X = {venus.x} = {venus.x.to (u_au)}')
print (f'    Y = {venus.y} = {venus.y.to (u_au)}')
print (f'    Z = {venus.z} = {venus.z.to (u_au)}')
print (f'  Earth:')
print (f'    X = {earth.x} = {earth.x.to (u_au)}')
print (f'    Y = {earth.y} = {earth.y.to (u_au)}')
print (f'    Z = {earth.z} = {earth.z.to (u_au)}')
print (f'  Mars:')
print (f'    X = {mars.x} = {mars.x.to (u_au)}')
print (f'    Y = {mars.y} = {mars.y.to (u_au)}')
print (f'    Z = {mars.z} = {mars.z.to (u_au)}')
```

Execute above script.

```
% ./ai202209_s04_35.py
Positions of the Sun and the planets at t = 2022-10-03T00:00:00.000
  Sun:
    X = -1359606.9109281122 km = -0.009088410848137241 AU
    Y = 112692.44733625517 km = 0.0007533024822408463 AU
```

```
    Z = 82143.21768792676 km = 0.0005490934951384089 AU
  Mercury:
    X = 26672328.720216326 km = 0.17829350508406885 AU
    Y = 34198126.43851055 km = 0.2286003555965757 AU
    Z = 15385099.0063862 km = 0.10284303469291425 AU
  Venus:
    X = -108730514.89265476 km = -0.7268185996490575 AU
    Y = 2682864.7882018164 km = 0.017933843414001324 AU
    Z = 8031811.812256415 km = 0.05368934580869282 AU
  Earth:
    X = 146318962.657391 km = 0.9780818535232735 AU
    Y = 22695522.164461944 km = 0.1517101951937204 AU
    Z = 9871115.469778635 km = 0.0659843313517071 AU
  Mars:
    X = 165516459.54380506 km = 1.1064091939899854 AU
    Y = 127725975.96491131 km = 0.8537954141142151 AU
    Z = 54113213.07823729 km = 0.3617244872873534 AU
```

16.2.2 Plotting positions of planets

Make a Python script to visualise positions of planets. Here is an example.

Python Code 37: ai202209_s04_36.py

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

#
# Time-stamp: <2022/10/02 21:20:19 (CST) daisuke>
#

# importing sys module
import sys

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

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

# output file name
file_output = 'ai202209_s04_36.png'

# units
u_au = astropy.units.au

# setting for solar system ephemeris
astropy.coordinates.solar_system_ephemeris.set ('jpl')

# date/time
t_str = f'2022-10-03T00:00:00'

# Astropy's time object
t = astropy.time.Time (t_str, format='isot', scale='utc')

# getting positions of Sun, Earth, and Moon
```

```
sun     = astropy.coordinates.get_body_barycentric ('sun', t)
mercury = astropy.coordinates.get_body_barycentric ('mercury', t)
venus   = astropy.coordinates.get_body_barycentric ('venus', t)
earth   = astropy.coordinates.get_body_barycentric ('earth', t)
mars    = astropy.coordinates.get_body_barycentric ('mars', t)

# printing positions of the Sun and planets
print (f'Positions of the Sun and the planets at t = {t}')
print (f'  Sun:')
print (f'    X = {sun.x} = {sun.x.to (u_au)}')
print (f'    Y = {sun.y} = {sun.y.to (u_au)}')
print (f'    Z = {sun.z} = {sun.z.to (u_au)}')
print (f'  Mercury:')
print (f'    X = {mercury.x} = {mercury.x.to (u_au)}')
print (f'    Y = {mercury.y} = {mercury.y.to (u_au)}')
print (f'    Z = {mercury.z} = {mercury.z.to (u_au)}')
print (f'  Venus:')
print (f'    X = {venus.x} = {venus.x.to (u_au)}')
print (f'    Y = {venus.y} = {venus.y.to (u_au)}')
print (f'    Z = {venus.z} = {venus.z.to (u_au)}')
print (f'  Earth:')
print (f'    X = {earth.x} = {earth.x.to (u_au)}')
print (f'    Y = {earth.y} = {earth.y.to (u_au)}')
print (f'    Z = {earth.z} = {earth.z.to (u_au)}')
print (f'  Mars:')
print (f'    X = {mars.x} = {mars.x.to (u_au)}')
print (f'    Y = {mars.y} = {mars.y.to (u_au)}')
print (f'    Z = {mars.z} = {mars.z.to (u_au)}')

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# settings for plot
ax.set_aspect ('equal')
ax.set_xlim (-2.0, +2.0)
ax.set_ylim (-2.0, +2.0)
ax.set_xlabel ("X [au]")
ax.set_ylabel ("Y [au]")
ax.set_title ("Positions of the Sun and planets")

# plotting the Sun
ax.plot (sun.x.to (u_au) / u_au, sun.y.to (u_au) / u_au, \
         marker='o', markersize=25, color='yellow', label='Sun')
ax.text (sun.x.to (u_au) / u_au + 0.1, sun.y.to (u_au) / u_au - 0.3, \
         f'Sun')

# plotting Mercury
ax.plot (mercury.x.to (u_au) / u_au, mercury.y.to (u_au) / u_au, \
         marker='o', markersize=5, color='cyan', label='Mercury')
ax.text (mercury.x.to (u_au) / u_au + 0.1, mercury.y.to (u_au) / u_au - 0.3, \
         f'Mercury')

# plotting Venus
```

```
ax.plot (venus.x.to (u_au) / u_au, venus.y.to (u_au) / u_au, \
         marker='o', markersize=15, color='gold', label='Venus')
ax.text (venus.x.to (u_au) / u_au + 0.1, venus.y.to (u_au) / u_au - 0.3, \
         f'Venus')

# plotting Earth
ax.plot (earth.x.to (u_au) / u_au, earth.y.to (u_au) / u_au, \
         marker='o', markersize=15, color='blue', label='Earth')
ax.text (earth.x.to (u_au) / u_au + 0.1, earth.y.to (u_au) / u_au - 0.3, \
         f'Earth')

# plotting Mars
ax.plot (mars.x.to (u_au) / u_au, mars.y.to (u_au) / u_au, \
         marker='o', markersize=10, color='red', label='Mars')
ax.text (mars.x.to (u_au) / u_au + 0.1, mars.y.to (u_au) / u_au - 0.3, \
         f'Mars')

# plotting the time
ax.text (-1.9, -1.9, f'Date/Time: {t} (UTC)')

# grid
ax.grid ()

# saving plot
fig.savefig (file_output, dpi=225)
```

Execute above script.

```
% ./ai202209_s04_36.py
Positions of the Sun and the planets at t = 2022-10-03T00:00:00.000
  Sun:
    X = -1359606.9109281122 km = -0.009088410848137241 AU
    Y = 112692.44733625517 km = 0.0007533024822408463 AU
    Z = 82143.21768792676 km = 0.0005490934951384089 AU
  Mercury:
    X = 26672328.720216326 km = 0.17829350508406885 AU
    Y = 34198126.43851055 km = 0.2286003555965757 AU
    Z = 15385099.0063862 km = 0.10284303469291425 AU
  Venus:
    X = -108730514.89265476 km = -0.7268185996490575 AU
    Y = 2682864.7882018164 km = 0.017933843414001324 AU
    Z = 8031811.812256415 km = 0.05368934580869282 AU
  Earth:
    X = 146318962.657391 km = 0.9780818535232735 AU
    Y = 22695522.164461944 km = 0.1517101951937204 AU
    Z = 9871115.469778635 km = 0.0659843313517071 AU
  Mars:
    X = 165516459.54380506 km = 1.1064091939899854 AU
    Y = 127725975.96491131 km = 0.8537954141142151 AU
    Z = 54113213.07823729 km = 0.3617244872873534 AU
% ls *.png
ai202209_s04_00.png       ai202209_s04_11.png       ai202209_s04_23.png
ai202209_s04_01.png       ai202209_s04_12.png       ai202209_s04_24.png
ai202209_s04_02.png       ai202209_s04_13.png       ai202209_s04_25.png
ai202209_s04_03.png       ai202209_s04_14.png       ai202209_s04_26.png
ai202209_s04_04.png       ai202209_s04_15.png       ai202209_s04_29.png
ai202209_s04_05.png       ai202209_s04_16.png       ai202209_s04_30.png
ai202209_s04_06.png       ai202209_s04_17.png       ai202209_s04_31.png
ai202209_s04_07.png       ai202209_s04_18.png       ai202209_s04_32.png
```

```
ai202209_s04_08.png        ai202209_s04_19.png        ai202209_s04_36.png
ai202209_s04_09.png        ai202209_s04_20.png
ai202209_s04_10.png        ai202209_s04_21.png
```

Display the PNG image file. (49)

```
% feh -dF ai202209_s04_36.png
```

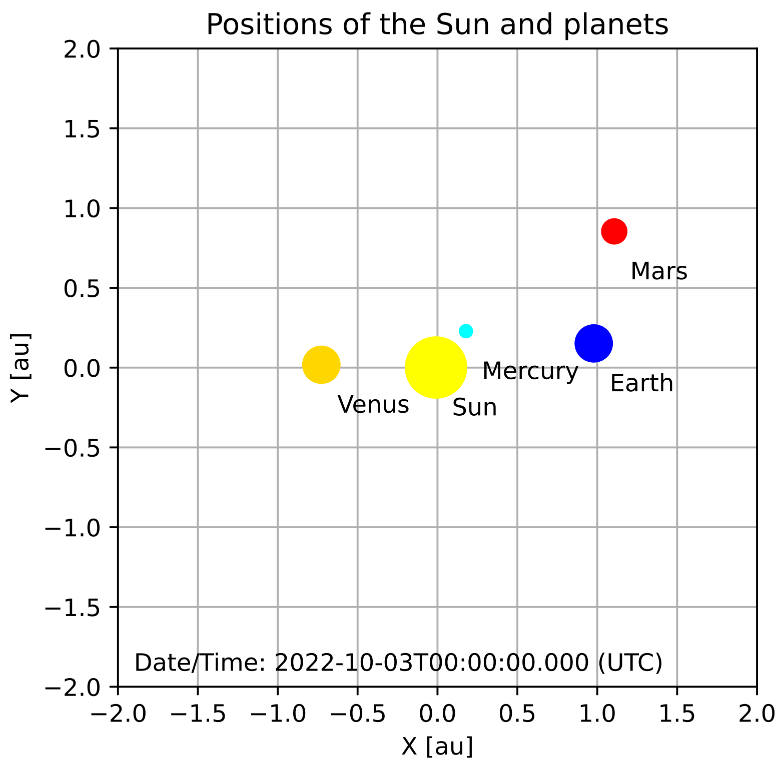


Figure 48: Plot created by the script `ai202209_s04_36.py`.

Try following practice.

Practice 04-31

Modify sample script "`ai202209_s04_36.py`", and plot the positions of planets on at 00:00:00 (UT) on 01 January 2023.

16.2.3 Making an animation

Make a Python script to create an animation to visualise orbital motion of planets. Here is an example.

Python Code 38: ai202209_s04_37.py

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

#
# Time-stamp: <2022/10/02 21:32:05 (CST) daisuke>
#

# importing sys module
```

```
import sys

# importing numpy module
import numpy

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

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

# output movie file
file_output = 'ai202209_s04_37.mp4'

# units
u_au = astropy.units.au
u_hr = astropy.units.hour

# setting for solar system ephemeris
astropy.coordinates.solar_system_ephemeris.set ('jpl')

# date/time
t0_str = f'2022-10-01T00:00:00'

# time to start the simulation
t0 = astropy.time.Time (t0_str, format='isot', scale='utc')

# number of steps to calculate
n_steps = 600

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111)

# an empty list of frames for animation
list_frame = []

for i in range (n_steps):
    # initialisation of object list
    list_obj = []

    # time t
    t = t0 + i * 6.0 * u_hr

    # getting positions of Sun, Earth, and Moon
    sun     = astropy.coordinates.get_body_barycentric ('sun', t)
    mercury = astropy.coordinates.get_body_barycentric ('mercury', t)
    venus   = astropy.coordinates.get_body_barycentric ('venus', t)
    earth   = astropy.coordinates.get_body_barycentric ('earth', t)
```

```
    mars    = astropy.coordinates.get_body_barycentric ('mars', t)

    # printing positions of the Sun and planets
    if (i % 100 == 0):
        print (f'Positions of the Sun and the planets at t = {t}')
        print (f'  Sun:')
        print (f'    X = {sun.x} = {sun.x.to (u_au)}')
        print (f'    Y = {sun.y} = {sun.y.to (u_au)}')
        print (f'    Z = {sun.z} = {sun.z.to (u_au)}')
        print (f'  Mercury:')
        print (f'    X = {mercury.x} = {mercury.x.to (u_au)}')
        print (f'    Y = {mercury.y} = {mercury.y.to (u_au)}')
        print (f'    Z = {mercury.z} = {mercury.z.to (u_au)}')
        print (f'  Venus:')
        print (f'    X = {venus.x} = {venus.x.to (u_au)}')
        print (f'    Y = {venus.y} = {venus.y.to (u_au)}')
        print (f'    Z = {venus.z} = {venus.z.to (u_au)}')
        print (f'  Earth:')
        print (f'    X = {earth.x} = {earth.x.to (u_au)}')
        print (f'    Y = {earth.y} = {earth.y.to (u_au)}')
        print (f'    Z = {earth.z} = {earth.z.to (u_au)}')
        print (f'  Mars:')
        print (f'    X = {mars.x} = {mars.x.to (u_au)}')
        print (f'    Y = {mars.y} = {mars.y.to (u_au)}')
        print (f'    Z = {mars.z} = {mars.z.to (u_au)}')

    # settings for plot
    ax.set_aspect ('equal')
    ax.set_xlim (-2.0, +2.0)
    ax.set_ylim (-2.0, +2.0)
    ax.set_xlabel ("X [au]")
    ax.set_ylabel ("Y [au]")
    ax.set_title ("Positions of the Sun and planets")

    # plotting grids
    grid_x = numpy.linspace (-2.0, +2.0, 9)
    grid_y = numpy.linspace (-2.0, +2.0, 9)
    for x in grid_x:
        grid, = ax.plot ([x, x], [-100, +100], \
                         linestyle='-', color='gray', alpha=0.3)
        list_obj.append (grid)
    for y in grid_y:
        grid, = ax.plot ([-100, +100], [y, y], \
                         linestyle='-', color='gray', alpha=0.3)
        list_obj.append (grid)

    # plotting the Sun
    sun_p, = ax.plot (sun.x.to (u_au) / u_au, sun.y.to (u_au) / u_au, \
                      marker='o', markersize=25, color='yellow', label='Sun')
    sun_t = ax.text (sun.x.to (u_au) / u_au + 0.1, \
                     sun.y.to (u_au) / u_au - 0.3, \
                     f'Sun')
    list_obj.append (sun_p)
    list_obj.append (sun_t)

    # plotting Mercury
    mercury_p, = ax.plot (mercury.x.to (u_au) / u_au, \
                          mercury.y.to (u_au) / u_au, \
                          marker='o', markersize=5, color='orange', \
```

```
                      label='Mercury')
    mercury_t = ax.text (mercury.x.to (u_au) / u_au + 0.1, \
                      mercury.y.to (u_au) / u_au - 0.3, \
                      f'Mercury')
    list_obj.append (mercury_p)
    list_obj.append (mercury_t)

    # plotting Venus
    venus_p, = ax.plot (venus.x.to (u_au) / u_au, venus.y.to (u_au) / u_au, \
                    marker='o', markersize=15, color='green', label='Venus')
    venus_t = ax.text (venus.x.to (u_au) / u_au + 0.1, \
                    venus.y.to (u_au) / u_au - 0.3, \
                    f'Venus')
    list_obj.append (venus_p)
    list_obj.append (venus_t)

    # plotting Earth
    earth_p, = ax.plot (earth.x.to (u_au) / u_au, earth.y.to (u_au) / u_au, \
                    marker='o', markersize=15, color='blue', label='Earth')
    earth_t = ax.text (earth.x.to (u_au) / u_au + 0.1, \
                    earth.y.to (u_au) / u_au - 0.3, \
                    f'Earth')
    list_obj.append (earth_p)
    list_obj.append (earth_t)

    # plotting Mars
    mars_p, = ax.plot (mars.x.to (u_au) / u_au, mars.y.to (u_au) / u_au, \
                   marker='o', markersize=10, color='red', label='Mars')
    mars_t = ax.text (mars.x.to (u_au) / u_au + 0.1, \
                   mars.y.to (u_au) / u_au - 0.3, \
                   f'Mars')
    list_obj.append (mars_p)
    list_obj.append (mars_t)

    # plotting the time
    time_t = ax.text (-1.9, -1.9, f'Date/Time: {t} (UTC)')
    list_obj.append (time_t)

    # appending frame
    list_frame.append (list_obj)

# making animation
anim = matplotlib.animation.ArtistAnimation (fig, list_frame, interval=50)

# saving file
anim.save (file_output, dpi=225)
```

Execute above script. It may take a few minutes.

```
% ./ai202209_s04_37.py
Positions of the Sun and the planets at t = 2022-10-01T00:00:00.000
  Sun:
    X = -1359555.7552425857 km = -0.009088068893500539 AU
    Y = 115194.00767457488 km = 0.0007700243802639558 AU
    Z = 83202.82881166792 km = 0.0005561765580107814 AU
  Mercury:
    X = 34491181.53145044 km = 0.2305593078969569 AU
    Y = 29110614.012016147 km = 0.19459243554605052 AU
    Z = 11856669.207730576 km = 0.07925693829899262 AU
```

```
  Venus:
    X = -108215180.3226651 km = -0.7233738008188448 AU
    Y = 8222458.849165574 km = 0.05496374253651442 AU
    Z = 10491738.045775872 km = 0.0701329370310073 AU
  Earth:
    X = 147163616.687637 km = 0.983728016976628 AU
    Y = 18044786.38427393 km = 0.12062194668840251 AU
    Z = 7854867.922300407 km = 0.0525065489605288 AU
  Mars:
    X = 167999132.72958964 km = 1.1230048391964822 AU
    Y = 124474812.6747821 km = 0.8320627298526255 AU
    Z = 52554900.693608865 km = 0.3513078123885948 AU

.....

Positions of the Sun and the planets at t = 2023-02-03T00:00:00.000
  Sun:
    X = -1348823.077724259 km = -0.009016325375574074 AU
    Y = -40067.69881552119 km = -0.00026783602352116356 AU
    Z = 17132.230481651615 km = 0.00011452188725338332 AU
  Mercury:
    X = -51483620.87012198 km = -0.3441467490761684 AU
    Y = -40434385.764847234 km = -0.27028717438053235 AU
    Z = -16365198.236670982 km = -0.10939459338622112 AU
  Venus:
    X = 104312662.02763678 km = 0.6972870772794815 AU
    Y = 24466574.585909918 km = 0.16354894940299386 AU
    Z = 4359088.499894538 km = 0.029138706851223505 AU
  Earth:
    X = -102940134.0407443 km = -0.688112294373347 AU
    Y = 97987287.2164075 km = 0.655004558272817 AU
    Z = 42511317.73861195 km = 0.2841706071061876 AU
  Mars:
    X = -58133921.76637931 km = -0.3886012648065004 AU
    Y = 210953632.81705666 km = 1.410137937324643 AU
    Z = 98327619.96306455 km = 0.657279542168407 AU
% ls -l *.mp4
-rw-r--r--  1 daisuke  taiwan   76168 Oct  2 20:48 ai202209_s04_33.mp4
-rw-r--r--  1 daisuke  taiwan  421164 Oct  2 21:35 ai202209_s04_37.mp4
```

Play the movie file.

```
% mplayer ai202209_s04_37.mp4
```

Try following practice.

Practice 04-32

Modify sample script "`ai202209_s04_36.py`", and plot the positions of planets on at 00:00:00 (UT) on 01 January 2023.

17 Making 3-dimensional plots

Try to make 3-dimensional plots using Matplotlib.

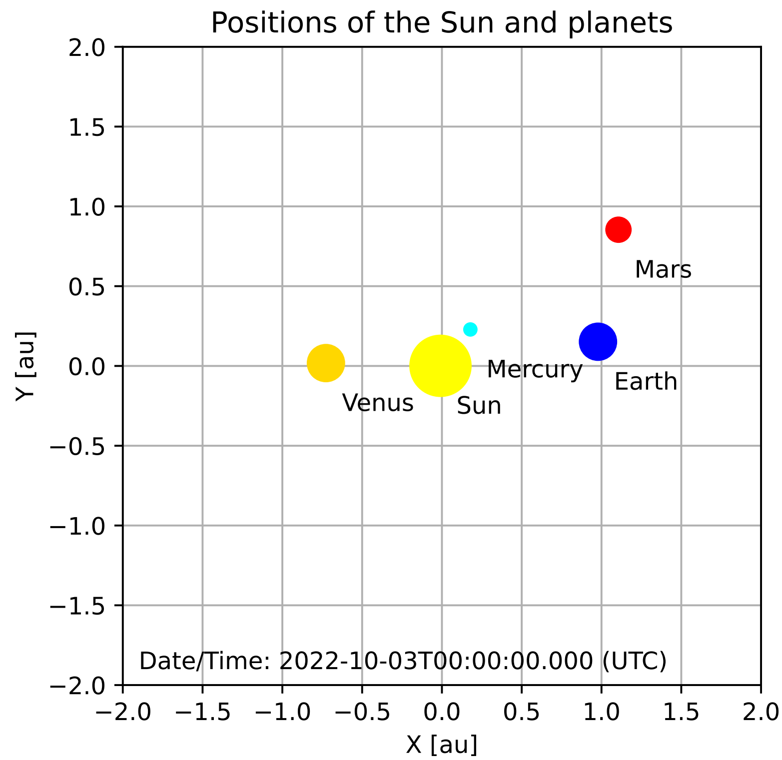


Figure 49: Plot created by the script `ai202209_s04_36.py`.

17.1 Making 3D scatter plot

Try following sample script.

Python Code 39: ai202209_s04_38.py

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

#
# Time-stamp: <2022/10/02 22:38:24 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output file name
file_output = 'ai202209_s04_38.png'

# parameters for random number generation
mean   = 0.0
stddev = 100.0
n      = 10**4

# data to be plotted
rng = numpy.random.default_rng ()
x   = rng.normal (loc=mean, scale=stddev, size=n)
```

```
y   = rng.normal (loc=mean, scale=stddev, size=n)
z   = rng.normal (loc=mean, scale=stddev, size=n)

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111, projection='3d')

# settings for plot
ax.set_xlim (-500.0, +500.0)
ax.set_ylim (-500.0, +500.0)
ax.set_zlim (-500.0, +500.0)
ax.set_box_aspect ( (1.0, 1.0, 1.0) )

# viewing angles of camera
el = 45.0
az = 60.0
ax.view_init (elev=el, azim=az)

# plotting data points
ax.scatter (x, y, z, s=1.0, color='blue', alpha=0.1)

# saving file
fig.savefig (file_output, dpi=200)
```

Execute above script.

```
% ./ai202209_s04_38.py
% ls *.png
ai202209_s04_00.png      ai202209_s04_11.png      ai202209_s04_23.png
ai202209_s04_01.png      ai202209_s04_12.png      ai202209_s04_24.png
ai202209_s04_02.png      ai202209_s04_13.png      ai202209_s04_25.png
ai202209_s04_03.png      ai202209_s04_14.png      ai202209_s04_26.png
ai202209_s04_04.png      ai202209_s04_15.png      ai202209_s04_29.png
ai202209_s04_05.png      ai202209_s04_16.png      ai202209_s04_30.png
ai202209_s04_06.png      ai202209_s04_17.png      ai202209_s04_31.png
ai202209_s04_07.png      ai202209_s04_18.png      ai202209_s04_32.png
ai202209_s04_08.png      ai202209_s04_19.png      ai202209_s04_36.png
ai202209_s04_09.png      ai202209_s04_20.png      ai202209_s04_38.png
ai202209_s04_10.png      ai202209_s04_21.png
```

Display the PNG image file. (50)

```
% feh -dF ai202209_s04_38.png
```

Try following practice.

Practice 04-33

Make your own Python script to generate 3D scatter plot using Matplotlib.

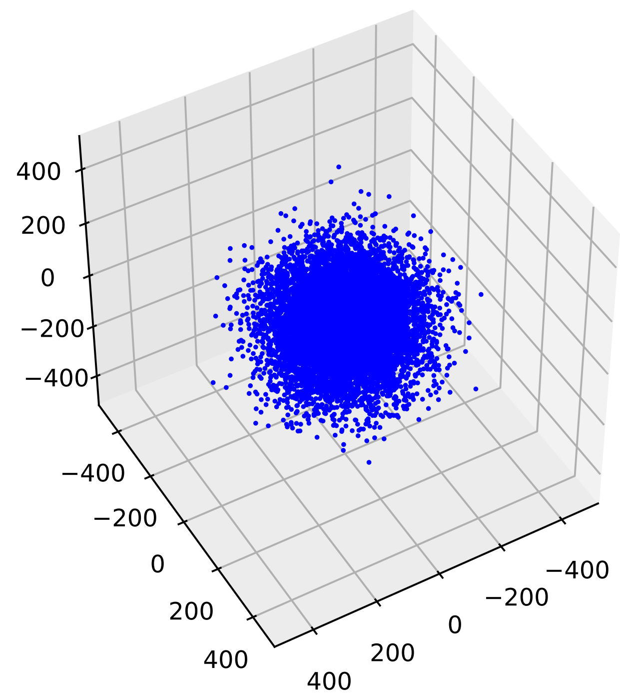


Figure 50: Plot created by the script `ai202209_s04_38.py`.

17.2 Making 3D line plot

Try following sample script.

Python Code 40: ai202209_s04_39.py

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

#
# Time-stamp: <2022/10/02 22:29:01 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output file name
file_output = 'ai202209_s04_39.png'

# data to be plotted
theta = numpy.linspace (0.0, 10.0 * numpy.pi, 1000)
x = numpy.cos (theta)
y = numpy.sin (theta)
z = theta * 0.2

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()
```

```
# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111, projection='3d')

# settings for plot
ax.set_xlim (-1.5, +1.5)
ax.set_ylim (-1.5, +1.5)
ax.set_zlim (-0.5, +5.5)
ax.set_box_aspect ( (3.0, 3.0, 6.0) )

# viewing angles of camera
el = 30.0
az = -60.0
ax.view_init (elev=el, azim=az)

# plotting data points
ax.plot (x, y, z, color='blue')

# saving file
fig.savefig (file_output, dpi=200)
```

Execute above script.

```
% ./ai202209_s04_39.py
% ls *.png
ai202209_s04_00.png        ai202209_s04_11.png        ai202209_s04_23.png
ai202209_s04_01.png        ai202209_s04_12.png        ai202209_s04_24.png
ai202209_s04_02.png        ai202209_s04_13.png        ai202209_s04_25.png
ai202209_s04_03.png        ai202209_s04_14.png        ai202209_s04_26.png
ai202209_s04_04.png        ai202209_s04_15.png        ai202209_s04_29.png
ai202209_s04_05.png        ai202209_s04_16.png        ai202209_s04_30.png
ai202209_s04_06.png        ai202209_s04_17.png        ai202209_s04_31.png
ai202209_s04_07.png        ai202209_s04_18.png        ai202209_s04_32.png
ai202209_s04_08.png        ai202209_s04_19.png        ai202209_s04_36.png
ai202209_s04_09.png        ai202209_s04_20.png        ai202209_s04_38.png
ai202209_s04_10.png        ai202209_s04_21.png        ai202209_s04_39.png
```

Display the PNG image file. (51)

```
% feh -dF ai202209_s04_39.png
```

Try following practice.

Practice 04-34

Make your own Python script to generate 3D line plot using Matplotlib.

18 3D structure of inner solar system

Visualise the 3-dimensional structure of inner solar system.

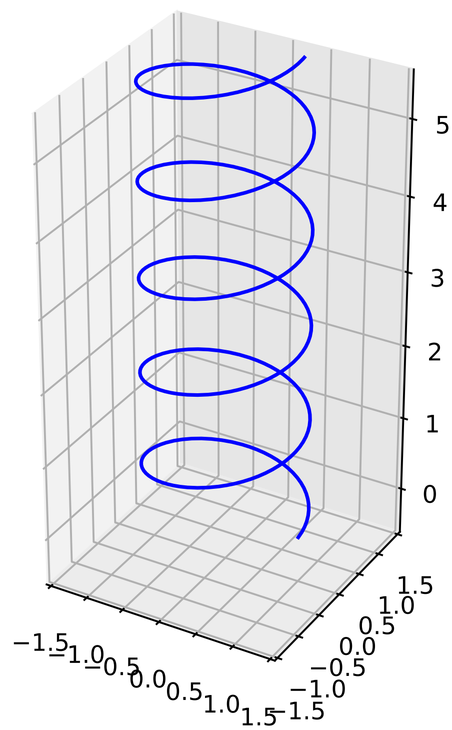


Figure 51: Plot created by the script ai202209_s04_39.py.

18.1 Making a movie of orbital motion of planets and 50 asteroids

Python Code 41: ai202209_s04_40.py

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

#
# Time-stamp: <2022/10/02 22:43:49 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# importing astroquery module
import astroquery.jplhorizons

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

```
# output file name prefix
file_prefix = 'solsys_3d_struct'

# output file name extension
file_ext    = 'png'

# units
u_au = astropy.units.au
u_hr = astropy.units.hour

# number of steps to calculate
n_steps = 600

# number of asteroids to plot
n_asteroids = 50

# step size in hr
step_hr  = 12
step_str = f'{step_hr}h'
step     = step_hr * u_hr

# an empty list for storing asteroids positions
list_asteroids = []

# date/time to start the simulation
t_start_str = f'2022-07-01T00:00:00.000'

# time to start the simulation in astropy.time object
t_start = astropy.time.Time (t_start_str, format='isot', scale='utc')

# time to stop the simulation in astropy.time object
t_stop  = t_start + step * n_steps

# an empty list for storing major planets positions
list_major = []

# major body names (Sun, Mercury, Venus, Earth, Mars, Jupiter)
list_names = ['10', '199', '299', '399', '499', '599']

# getting positions of the Sun, Mercury, Venus, Earth, Mars, and Jupiter
# from JPL/Horizons
print (f'Now, getting positions of the Sun and planets...')
for i in list_names:
    print (i)
    query = astroquery.jplhorizons.Horizons (id_type=None, id=f'{i}', \
                                             location='@0', \
                                             epochs={'start': t_start.iso, \
                                                     'stop': t_stop.iso, \
                                                     'step': step_str})
    vec = query.vectors ()
    print (vec)
    x = vec['x']
    y = vec['y']
    z = vec['z']
    list_major.append ( [x, y, z] )
print (f'Finished getting positions of the Sun and planets!')

# getting asteroids positions from JPL/Horizons
```

```
print (f'Now, getting asteroids positions...')
for i in range (1, n_asteroids + 1):
    if (i % 10 == 0):
        print (f'  now, getting positions of asteroid ({i})...')
    ast_query = astroquery.jplhorizons.Horizons (id_type='smallbody', \
                                                  id=f'{i}', \
                                                  location='@0', \
                                                  epochs={'start': t_start.iso, \
                                                          'stop': t_stop.iso, \
                                                          'step': step_str})
    ast_vec = ast_query.vectors ()
    x = ast_vec['x']
    y = ast_vec['y']
    z = ast_vec['z']
    list_asteroids.append ( [x, y, z] )
print (f'Finished getting asteroids positions...')

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()
fig.subplots_adjust (left=0.0, right=1.0, bottom=0.0, top=1.0)

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111, projection='3d')

# an empty list of frames for animation
list_frame = []

# definition of a function for making a sphere
def make_sphere (x_c, y_c, z_c, radius, colour):
    u = numpy.linspace (0, 2 * numpy.pi, 1000)
    v = numpy.linspace (0, numpy.pi, 1000)
    x = radius * numpy.outer (numpy.cos(u), numpy.sin(v)) + x_c
    y = radius * numpy.outer (numpy.sin(u), numpy.sin(v)) + y_c
    z = radius * numpy.outer (numpy.ones(numpy.size(u)), numpy.cos(v)) + z_c
    # plotting the surface
    sphere = ax.plot_surface (x, y, z, color=colour, antialiased=False, \
                              shade=True, rcount=100, ccount=100)
    return (sphere)

# initial value of 'elev' angle
el = 90.0

# initial value of 'azim' angle
az = 0.0

for i in range (n_steps):
    # clearing previous axes
    ax.cla ()

    # time t
    t = t_start + i * 12.0 * u_hr

    # printing positions of the Sun, planets, and asteroids
    if (i % 10 == 0):
        print (f'Now, making a plot for {t}...')
```

```
    # settings for plot
    ax.set_xlim (-6.0, +6.0)
    ax.set_ylim (-6.0, +6.0)
    ax.set_zlim (-2.0, +2.0)
    ax.set_box_aspect ( (6.0, 6.0, 2.0) )

    # viewing angles of camera
    ax.view_init (elev=el, azim=az)

    # using black background colour
    fig.set_facecolor ('black')
    ax.set_facecolor ('black')
    ax.grid (False)
    ax.w_xaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_yaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_zaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))

    # plotting the Sun
    sun = make_sphere (list_major[0][0][i], \
                       list_major[0][1][i], \
                       list_major[0][2][i], \
                       0.25, 'yellow')

    # plotting Mercury
    mercury = make_sphere (list_major[1][0][i], \
                           list_major[1][1][i], \
                           list_major[1][2][i], \
                           0.05, 'cyan')

    # plotting Venus
    venus = make_sphere (list_major[2][0][i], \
                         list_major[2][1][i], \
                         list_major[2][2][i], \
                         0.15, 'gold')

    # plotting Earth
    earth = make_sphere (list_major[3][0][i], \
                         list_major[3][1][i], \
                         list_major[3][2][i], \
                         0.15, 'blue')

    # plotting Mars
    mars = make_sphere (list_major[4][0][i], \
                        list_major[4][1][i], \
                        list_major[4][2][i], \
                        0.15, 'red')

    # plotting Jupiter
    jupiter = make_sphere (list_major[5][0][i], \
                           list_major[5][1][i], \
                           list_major[5][2][i], \
                           0.15, 'bisque')

    # plotting asteroids
    for j in range (0, n_asteroids):
        asteroid = ax.scatter (list_asteroids[j][0][i], \
                               list_asteroids[j][1][i], \
                               list_asteroids[j][2][i], \
                               s=0.1, \
```

```
                                color='saddlebrown')

    # title
    title = ax.text2D (0.5, 0.95, f'Inner Solar System', \
                       color='white', \
                       horizontalalignment='center', \
                       transform=ax.transAxes)

    # plotting the time
    time = ax.text2D (0.5, 0.05, f'Date/Time: {t} (UTC)', \
                      color='white', \
                      horizontalalignment='center', \
                      transform=ax.transAxes)

    # image file
    file_image = f'{file_prefix}_{i:06d}.{file_ext}'
    fig.savefig (file_image, dpi=255)
```

Execute above script.

```
% ./ai202209_s04_40.py
Now, getting positions of the Sun and planets...
10
targetname datetime_jd ...         range                range_rate
   ---           d     ...           AU                  AU / d
---------- ----------- ... -------------------- ----------------------
  Sun (10)   2459761.5 ...  0.00919225694612516 -4.110279681942871e-07
  Sun (10)   2459762.0 ... 0.009192050821591085 -4.134694264547141e-07
  Sun (10)   2459762.5 ... 0.009191843477503799 -4.159060529885512e-07
  Sun (10)   2459763.0 ...  0.00919163491647408 -4.183370625836328e-07
  Sun (10)   2459763.5 ... 0.009191425141513318 -4.207616386659481e-07
  Sun (10)   2459764.0 ... 0.009191214156048148 -4.231789376034708e-07
  Sun (10)   2459764.5 ...  0.00919100196393274 -4.255880938532694e-07
  Sun (10)   2459765.0 ... 0.009190788569458251 -4.279882260847911e-07
  Sun (10)   2459765.5 ... 0.009190573977358986 -4.303784443249411e-07
  Sun (10)   2459766.0 ... 0.009190358192814718 -4.327578580932779e-07
  Sun (10)   2459766.5 ...  0.00919014122144873  -4.35125585428295e-07
  Sun (10)   2459767.0 ... 0.009189923069321133  -4.37480762648432e-07
  Sun (10)   2459767.5 ...  0.00918970374291717 -4.398225546441054e-07
  Sun (10)   2459768.0 ... 0.009189483249130296 -4.421501654596851e-07
  Sun (10)   2459768.5 ...  0.00918926159523994 -4.444628488970648e-07
       ...         ... ...                  ...                    ...
  Sun (10)   2460054.0 ... 0.008924970359344905 -1.370193489082736e-06
  Sun (10)   2460054.5 ...  0.00892428511440673 -1.370785591967988e-06
  Sun (10)   2460055.0 ... 0.008923599574404022 -1.371373787812672e-06
  Sun (10)   2460055.5 ... 0.008922913741230698 -1.371958312474024e-06
  Sun (10)   2460056.0 ... 0.008922227616666499 -1.372539386985973e-06
  Sun (10)   2460056.5 ... 0.008921541202384083 -1.373117218859561e-06
  Sun (10)   2460057.0 ... 0.008920854499955479 -1.373692003317508e-06
  Sun (10)   2460057.5 ... 0.008920167510857929 -1.374263924459444e-06
  Sun (10)   2460058.0 ... 0.008919480236479179  -1.37483315635616e-06
  Sun (10)   2460058.5 ... 0.008918792678122225 -1.375399864072787e-06
  Sun (10)   2460059.0 ... 0.008918104837009591 -1.375964204621995e-06
  Sun (10)   2460059.5 ... 0.008917416714287123 -1.376526327849376e-06
  Sun (10)   2460060.0 ...  0.00891672831102735 -1.377086377253719e-06
  Sun (10)   2460060.5 ... 0.008916039628232493 -1.377644490745359e-06
  Sun (10)   2460061.0 ... 0.008915350666837062 -1.378200801345849e-06
  Sun (10)   2460061.5 ... 0.008914661427710146  -1.37875543783199e-06
Length = 601 rows
```

```
199
  targetname  datetime_jd ...       range              range_rate
     ---           d      ...         AU                  AU / d
------------- ----------- ... ------------------ ----------------------
Mercury (199)   2459761.5 ... 0.3274393111343241  -0.004565830548086274
Mercury (199)   2459762.0 ... 0.3252023885183809  -0.004379622776293806
Mercury (199)   2459762.5 ... 0.323061919358339 7   -0.0041800318660763 1
Mercury (199)   2459763.0 ... 0.3210245651769797  -0.003967192900115614
Mercury (199)   2459763.5 ... 0.3190968970577558  -0.003741333630063151
Mercury (199)   2459764.0 ... 0.3172853477717369  -0.003502780467817527
Mercury (199)   2459764.5 ... 0.3155961611762532  -0.003251963361740028
Mercury (199)   2459765.0 ... 0.3140353395111439  -0.002989419285078328
Mercury (199)   2459765.5 ... 0.3126085893527096  -0.002715794074705812
Mercury (199)   2459766.0 ... 0.3113212671107578   -0.00243184238347338
Mercury (199)   2459766.5 ... 0.3101783250629669  -0.002138425550114537
Mercury (199)   2459767.0 ... 0.3091842590054414    -0.0018365072466839
Mercury (199)   2459767.5 ... 0.3083430586513591  -0.001527146833589715
Mercury (199)   2459768.0 ... 0.3076581619245999  -0.001211490433849091
Mercury (199)   2459768.5 ... 0.3071324142673004 -0.0008907598277119476
          ...         ... ...                ...                    ...
Mercury (199)   2460054.0 ...  0.393483108044801   0.005801934294249164
Mercury (199)   2460054.5 ... 0.3963689808676535   0.005740627356391798
Mercury (199)   2460055.0 ... 0.3992228324346696   0.005673902118757671
Mercury (199)   2460055.5 ... 0.4020420316210633   0.005602068195627878
Mercury (199)   2460056.0 ... 0.4048240984003011   0.005525420409186732
Mercury (199)   2460056.5 ...  0.407566696519905   0.005444239089678288
Mercury (199)   2460057.0 ... 0.4102676263450556   0.005358790443726078
Mercury (199)   2460057.5 ... 0.4129248179001903   0.005269326975648785
Mercury (199)   2460058.0 ... 0.4155363241317504    0.00517608794874178
Mercury (199)   2460058.5 ... 0.4181003144092107   0.005079299875418875
Mercury (199)   2460059.0 ... 0.4206150682764054   0.004979177026825403
Mercury (199)   2460059.5 ... 0.4230789694608592    0.00487592195404486
Mercury (199)   2460060.0 ... 0.4254905001452272   0.004769726014347556
Mercury (199)   2460060.5 ... 0.4278482355019786   0.004660769897090876
Mercury (199)   2460061.0 ... 0.4301508384900073   0.004549224144886354
Mercury (199)   2460061.5 ... 0.4323970549098948   0.004435249666515193
Length = 601 rows

.....

599
  targetname  datetime_jd ...       range             range_rate
     ---           d      ...         AU                AU / d
------------- ----------- ... ----------------- ----------------------
Jupiter (599)   2459761.5 ...   4.9538498697702 -0.0001151013025872309
Jupiter (599)   2459762.0 ... 4.953792202039543 -0.0001155078574475694
Jupiter (599)   2459762.5 ... 4.953734546658476 -0.0001149273103477378
Jupiter (599)   2459763.0 ... 4.953677305141171 -0.0001141870856057821
Jupiter (599)   2459763.5 ... 4.953620158902407 -0.0001145158410096909
Jupiter (599)   2459764.0 ... 4.953562853808891 -0.0001145022107028578
Jupiter (599)   2459764.5 ... 4.953505826007736 -0.0001135851731288924
Jupiter (599)   2459765.0 ... 4.953449109623551 -0.0001135338035627637
Jupiter (599)   2459765.5 ... 4.953392145953957 -0.0001142752345458102
Jupiter (599)   2459766.0 ... 4.953335027348261  -0.000113974440272014
Jupiter (599)   2459766.5 ... 4.953278258075731 -0.0001131958475067818
Jupiter (599)   2459767.0 ... 4.953221661376215 -0.0001133038183902508
Jupiter (599)   2459767.5 ... 4.953165018997601 -0.0001130515132368666
Jupiter (599)   2459768.0 ... 4.953108811456524 -0.0001117100486845901
Jupiter (599)   2459768.5 ... 4.953053180985981 -0.0001110648566765412
```

```
        ...           ... ...             ...                   ...
Jupiter (599)   2460054.0 ...  4.94476376624791  5.372708298809731e-05
Jupiter (599)   2460054.5 ... 4.944790761781209  5.449456699472935e-05
Jupiter (599)   2460055.0 ... 4.944818342816371  5.572112590526076e-05
Jupiter (599)   2460055.5 ... 4.944846286238746  5.589534776253733e-05
Jupiter (599)   2460056.0 ... 4.944874252883687  5.614093619470511e-05
Jupiter (599)   2460056.5 ... 4.944902586677033  5.722390271093653e-05
Jupiter (599)   2460057.0 ...  4.94493132862853  5.748972050871348e-05
Jupiter (599)   2460057.5 ... 4.944959902988639  5.681701865819641e-05
Jupiter (599)   2460058.0 ... 4.944988323829351   5.70960790142461e-05
Jupiter (599)   2460058.5 ... 4.945017094355454  5.789704712465381e-05
Jupiter (599)   2460059.0 ... 4.945046028676242  5.766902170669467e-05
Jupiter (599)   2460059.5 ... 4.945074762345876  5.743699615842592e-05
Jupiter (599)   2460060.0 ... 4.945103658150575   5.82326176466909e-05
Jupiter (599)   2460060.5 ... 4.945132901440893  5.852248883376549e-05
Jupiter (599)   2460061.0 ... 4.945162017455077  5.794554607016282e-05
Jupiter (599)   2460061.5 ... 4.945191031301317  5.836848559158515e-05
Length = 601 rows
Finished getting positions of the Sun and planets!
Now, getting asteroids positions...
  now, getting positions of asteroid (10)...
  now, getting positions of asteroid (20)...
  now, getting positions of asteroid (30)...
  now, getting positions of asteroid (40)...
  now, getting positions of asteroid (50)...
Finished getting asteroids positions...
Now, making a plot for 2022-07-01T00:00:00.000...
Now, making a plot for 2022-07-06T00:00:00.000...
Now, making a plot for 2022-07-11T00:00:00.000...
Now, making a plot for 2022-07-16T00:00:00.000...
Now, making a plot for 2022-07-21T00:00:00.000...
Now, making a plot for 2022-07-26T00:00:00.000...
Now, making a plot for 2022-07-31T00:00:00.000...
Now, making a plot for 2022-08-05T00:00:00.000...
Now, making a plot for 2022-08-10T00:00:00.000...
Now, making a plot for 2022-08-15T00:00:00.000...

.....

Now, making a plot for 2023-03-08T00:00:00.000...
Now, making a plot for 2023-03-13T00:00:00.000...
Now, making a plot for 2023-03-18T00:00:00.000...
Now, making a plot for 2023-03-23T00:00:00.000...
Now, making a plot for 2023-03-28T00:00:00.000...
Now, making a plot for 2023-04-02T00:00:00.000...
Now, making a plot for 2023-04-07T00:00:00.000...
Now, making a plot for 2023-04-12T00:00:00.000...
Now, making a plot for 2023-04-17T00:00:00.000...
Now, making a plot for 2023-04-22T00:00:00.000...
% ls solsys_3d_struct_*.png
solsys_3d_struct_000000.png          solsys_3d_struct_000300.png
solsys_3d_struct_000001.png          solsys_3d_struct_000301.png
solsys_3d_struct_000002.png          solsys_3d_struct_000302.png
solsys_3d_struct_000003.png          solsys_3d_struct_000303.png
solsys_3d_struct_000004.png          solsys_3d_struct_000304.png
solsys_3d_struct_000005.png          solsys_3d_struct_000305.png
solsys_3d_struct_000006.png          solsys_3d_struct_000306.png
solsys_3d_struct_000007.png          solsys_3d_struct_000307.png
solsys_3d_struct_000008.png          solsys_3d_struct_000308.png
```

```
solsys_3d_struct_000009.png            solsys_3d_struct_000309.png

.....

solsys_3d_struct_000290.png            solsys_3d_struct_000590.png
solsys_3d_struct_000291.png            solsys_3d_struct_000591.png
solsys_3d_struct_000292.png            solsys_3d_struct_000592.png
solsys_3d_struct_000293.png            solsys_3d_struct_000593.png
solsys_3d_struct_000294.png            solsys_3d_struct_000594.png
solsys_3d_struct_000295.png            solsys_3d_struct_000595.png
solsys_3d_struct_000296.png            solsys_3d_struct_000596.png
solsys_3d_struct_000297.png            solsys_3d_struct_000597.png
solsys_3d_struct_000298.png            solsys_3d_struct_000598.png
solsys_3d_struct_000299.png            solsys_3d_struct_000599.png
```

Now, you have 600 PNG files. Construct a MPEG-4 movie file from these 600 PNG files using the command `ffmpeg`.

```
% ffmpeg5 -f image2 -start_number 0 -framerate 30 -i solsys_3d_struct_%06d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 8 solsys_3d_struct.mp4
% ls -l *.mp4
-rw-r--r--  1 daisuke  taiwan   76168 Oct  2 20:48 ai202209_s04_33.mp4
-rw-r--r--  1 daisuke  taiwan  421164 Oct  2 21:35 ai202209_s04_37.mp4
-rw-r--r--  1 daisuke  taiwan  364013 Oct  2 23:42 solsys_3d_struct.mp4
```

Play the movie file.

```
% mplayer solsys_3d_struct.mp4
```

18.2 Changing camera viewing angle

Change the camera viewing angle.

Python Code 42: ai202209_s04_41.py

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

#
# Time-stamp: <2022/10/02 23:01:53 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# importing astroquery module
import astroquery.jplhorizons

# importing matplotlib module
import matplotlib.animation
```

```
import matplotlib.backends.backend_agg
import matplotlib.figure

# output file name prefix
file_prefix = 'solsys_3d_struct2'

# output file name extension
file_ext    = 'png'

# units
u_au = astropy.units.au
u_hr = astropy.units.hour

# number of steps to calculate
n_steps = 600

# number of asteroids to plot
n_asteroids = 50

# step size in hr
step_hr  = 12
step_str = f'{step_hr}h'
step     = step_hr * u_hr

# an empty list for storing asteroids positions
list_asteroids = []

# date/time to start the simulation
t_start_str = f'2022-07-01T00:00:00.000'

# time to start the simulation in astropy.time object
t_start = astropy.time.Time (t_start_str, format='isot', scale='utc')

# time to stop the simulation in astropy.time object
t_stop  = t_start + step * n_steps

# an empty list for storing major planets positions
list_major = []

# major body names (Sun, Mercury, Venus, Earth, Mars, Jupiter)
list_names = ['10', '199', '299', '399', '499', '599']

# getting positions of the Sun, Mercury, Venus, Earth, Mars, and Jupiter
# from JPL/Horizons
print (f'Now, getting positions of the Sun and planets...')
for i in list_names:
    print (i)
    query = astroquery.jplhorizons.Horizons (id_type=None, id=f'{i}', \
                                             location='@0', \
                                             epochs={'start': t_start.iso, \
                                                     'stop': t_stop.iso, \
                                                     'step': step_str})
    vec = query.vectors ()
    print (vec)
    x = vec['x']
    y = vec['y']
    z = vec['z']
    list_major.append ( [x, y, z] )
print (f'Finished getting positions of the Sun and planets!')
```

```
# getting asteroids positions from JPL/Horizons
print (f'Now, getting asteroids positions...')
for i in range (1, n_asteroids + 1):
    if (i % 10 == 0):
        print (f'  now, getting positions of asteroid ({i})...')
    ast_query = astroquery.jplhorizons.Horizons (id_type='smallbody', \
                                                 id=f'{i}', \
                                                 location='@0', \
                                                 epochs={'start': t_start.iso, \
                                                         'stop': t_stop.iso, \
                                                         'step': step_str})
    ast_vec = ast_query.vectors ()
    x = ast_vec['x']
    y = ast_vec['y']
    z = ast_vec['z']
    list_asteroids.append ( [x, y, z] )
print (f'Finished getting asteroids positions...')

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()
fig.subplots_adjust (left=0.0, right=1.0, bottom=0.0, top=1.0)

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

# making an axes object
ax = fig.add_subplot (111, projection='3d')

# an empty list of frames for animation
list_frame = []

# definition of a function for making a sphere
def make_sphere (x_c, y_c, z_c, radius, colour):
    u = numpy.linspace (0, 2 * numpy.pi, 1000)
    v = numpy.linspace (0, numpy.pi, 1000)
    x = radius * numpy.outer (numpy.cos(u), numpy.sin(v)) + x_c
    y = radius * numpy.outer (numpy.sin(u), numpy.sin(v)) + y_c
    z = radius * numpy.outer (numpy.ones(numpy.size(u)), numpy.cos(v)) + z_c
    # plotting the surface
    sphere = ax.plot_surface (x, y, z, color=colour, antialiased=False, \
                              shade=True, rcount=100, ccount=100)
    return (sphere)

# initial value of 'elev' angle
el = 30.0

# initial value of 'azim' angle
az = 0.0

for i in range (n_steps):
    # clearing previous axes
    ax.cla ()

    # time t
    t = t_start + i * 12.0 * u_hr

    # printing positions of the Sun, planets, and asteroids
    if (i % 10 == 0):
```

```
        print (f'Now, making a plot for {t}...')

    # settings for plot
    ax.set_xlim (-6.0, +6.0)
    ax.set_ylim (-6.0, +6.0)
    ax.set_zlim (-2.0, +2.0)
    ax.set_box_aspect ( (6.0, 6.0, 2.0) )

    # viewing angles of camera
    ax.view_init (elev=el, azim=az)

    # using black background colour
    fig.set_facecolor ('black')
    ax.set_facecolor ('black')
    ax.grid (False)
    ax.w_xaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_yaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_zaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))

    # plotting the Sun
    sun = make_sphere (list_major[0][0][i], \
                       list_major[0][1][i], \
                       list_major[0][2][i], \
                       0.25, 'yellow')

    # plotting Mercury
    mercury = make_sphere (list_major[1][0][i], \
                           list_major[1][1][i], \
                           list_major[1][2][i], \
                           0.05, 'cyan')

    # plotting Venus
    venus = make_sphere (list_major[2][0][i], \
                         list_major[2][1][i], \
                         list_major[2][2][i], \
                         0.15, 'gold')

    # plotting Earth
    earth = make_sphere (list_major[3][0][i], \
                         list_major[3][1][i], \
                         list_major[3][2][i], \
                         0.15, 'blue')

    # plotting Mars
    mars = make_sphere (list_major[4][0][i], \
                        list_major[4][1][i], \
                        list_major[4][2][i], \
                        0.15, 'red')

    # plotting Jupiter
    jupiter = make_sphere (list_major[5][0][i], \
                           list_major[5][1][i], \
                           list_major[5][2][i], \
                           0.15, 'bisque')

    # plotting asteroids
    for j in range (0, n_asteroids):
        asteroid = ax.scatter (list_asteroids[j][0][i], \
                               list_asteroids[j][1][i], \
```

```
                                list_asteroids[j][2][i], \
                                s=0.1, \
                                color='saddlebrown')

    # title
    title = ax.text2D (0.5, 0.95, f'Inner Solar System', \
                        color='white', \
                        horizontalalignment='center', \
                        transform=ax.transAxes)

    # plotting the time
    time = ax.text2D (0.5, 0.05, f'Date/Time: {t} (UTC)', \
                       color='white', \
                       horizontalalignment='center', \
                       transform=ax.transAxes)

    # image file
    file_image = f'{file_prefix}_{i:06d}.{file_ext}'
    fig.savefig (file_image, dpi=255)
```

Execute above script.

```
% ./ai202209_s04_41.py
% ls solsys_3d_struct2_*.png
solsys_3d_struct2_000000.png            solsys_3d_struct2_000300.png
solsys_3d_struct2_000001.png            solsys_3d_struct2_000301.png
solsys_3d_struct2_000002.png            solsys_3d_struct2_000302.png
solsys_3d_struct2_000003.png            solsys_3d_struct2_000303.png
solsys_3d_struct2_000004.png            solsys_3d_struct2_000304.png
solsys_3d_struct2_000005.png            solsys_3d_struct2_000305.png
solsys_3d_struct2_000006.png            solsys_3d_struct2_000306.png
solsys_3d_struct2_000007.png            solsys_3d_struct2_000307.png
solsys_3d_struct2_000008.png            solsys_3d_struct2_000308.png
solsys_3d_struct2_000009.png            solsys_3d_struct2_000309.png

.....

solsys_3d_struct2_000290.png            solsys_3d_struct2_000590.png
solsys_3d_struct2_000291.png            solsys_3d_struct2_000591.png
solsys_3d_struct2_000292.png            solsys_3d_struct2_000592.png
solsys_3d_struct2_000293.png            solsys_3d_struct2_000593.png
solsys_3d_struct2_000294.png            solsys_3d_struct2_000594.png
solsys_3d_struct2_000295.png            solsys_3d_struct2_000595.png
solsys_3d_struct2_000296.png            solsys_3d_struct2_000596.png
solsys_3d_struct2_000297.png            solsys_3d_struct2_000597.png
solsys_3d_struct2_000298.png            solsys_3d_struct2_000598.png
solsys_3d_struct2_000299.png            solsys_3d_struct2_000599.png
```

Now, you have 600 PNG files. Construct a MPEG-4 movie file from these 600 PNG files using the command `ffmpeg`.

```
% ffmpeg5 -f image2 -start_number 0 -framerate 30 -i solsys_3d_struct2_%06d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 8 solsys_3d_struct2.mp4
% ls -l *.mp4
-rw-r--r--  1 daisuke  taiwan   76168 Oct  3 00:13 ai202209_s04_33.mp4
-rw-r--r--  1 daisuke  taiwan  421164 Oct  3 00:21 ai202209_s04_37.mp4
-rw-r--r--  1 daisuke  taiwan  364013 Oct  2 23:42 solsys_3d_struct.mp4
-rw-r--r--  1 daisuke  taiwan  354186 Oct  3 00:30 solsys_3d_struct2.mp4
```

Play the movie file.

```
% mplayer solsys_3d_struct2.mp4
```

18.3 Making a movie of orbital motion of planets and 5000 asteroids

Make an animation of orbital motion of planets and 5000 asteroids.

Python Code 43: ai202209_s04_42.py

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

#
# Time-stamp: <2022/10/02 23:05:00 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# importing astroquery module
import astroquery.jplhorizons

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

# output file name prefix
file_prefix = 'solsys_3d_struct3'

# output file name extension
file_ext    = 'png'

# units
u_au = astropy.units.au
u_hr = astropy.units.hour

# number of steps to calculate
n_steps = 5000

# number of asteroids to plot
n_asteroids = 5000

# step size in hr
step_hr  = 12
step_str = f'{step_hr}h'
step     = step_hr * u_hr

# an empty list for storing asteroids positions
list_asteroids = []
```

```
# date/time to start the simulation
t_start_str = f'2022-07-01T00:00:00.000'

# time to start the simulation in astropy.time object
t_start = astropy.time.Time (t_start_str, format='isot', scale='utc')

# time to stop the simulation in astropy.time object
t_stop  = t_start + step * n_steps

# an empty list for storing major planets positions
list_major = []

# major body names (Sun, Mercury, Venus, Earth, Mars, Jupiter)
list_names = ['10', '199', '299', '399', '499', '599']

# getting positions of the Sun, Mercury, Venus, Earth, Mars, and Jupiter
# from JPL/Horizons
print (f'Now, getting positions of the Sun and planets...')
for i in list_names:
    print (i)
    query = astroquery.jplhorizons.Horizons (id_type=None, id=f'{i}', \
                                             location='@0', \
                                             epochs={'start': t_start.iso, \
                                                     'stop': t_stop.iso, \
                                                     'step': step_str})
    vec = query.vectors ()
    print (vec)
    x = vec['x']
    y = vec['y']
    z = vec['z']
    list_major.append ( [x, y, z] )
print (f'Finished getting positions of the Sun and planets!')

# getting asteroids positions from JPL/Horizons
print (f'Now, getting asteroids positions...')
for i in range (1, n_asteroids + 1):
    if (i % 10 == 0):
        print (f'  now, getting positions of asteroid ({i})...')
    ast_query = astroquery.jplhorizons.Horizons (id_type='smallbody', \
                                                 id=f'{i}', \
                                                 location='@0', \
                                                 epochs={'start': t_start.iso, \
                                                         'stop': t_stop.iso, \
                                                         'step': step_str})
    ast_vec = ast_query.vectors ()
    x = ast_vec['x']
    y = ast_vec['y']
    z = ast_vec['z']
    list_asteroids.append ( [x, y, z] )
print (f'Finished getting asteroids positions...')

# making a fig object using object-oriented interface
fig = matplotlib.figure.Figure ()
fig.subplots_adjust (left=0.0, right=1.0, bottom=0.0, top=1.0)

# making a canvas object
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
```

```
# making an axes object
ax = fig.add_subplot (111, projection='3d')

# an empty list of frames for animation
list_frame = []

# definition of a function for making a sphere
def make_sphere (x_c, y_c, z_c, radius, colour):
    u = numpy.linspace (0, 2 * numpy.pi, 1000)
    v = numpy.linspace (0, numpy.pi, 1000)
    x = radius * numpy.outer (numpy.cos(u), numpy.sin(v)) + x_c
    y = radius * numpy.outer (numpy.sin(u), numpy.sin(v)) + y_c
    z = radius * numpy.outer (numpy.ones(numpy.size(u)), numpy.cos(v)) + z_c
    # plotting the surface
    sphere = ax.plot_surface (x, y, z, color=colour, antialiased=False, \
                              shade=True, rcount=100, ccount=100)
    return (sphere)

# initial value of 'elev' angle
el0 = 90.0

# initial value of 'azim' angle
az0 = 0.0

for i in range (n_steps):
    # clearing previous axes
    ax.cla ()

    # camera viewing angle
    if (i < 200):
        el = el0
        az = az0
    elif ( (i >= 200) and (i < 1400) ):
        el = el0 - (i - 200) * 0.1
        az = az0
    elif ( (i >= 1400) and (i < 1600) ):
        el = -30.0
        az = az0
    elif ( (i >= 1600) and (i < 1900) ):
        el = -30 + (i - 1600) * 0.1
        az = az0
    elif ( (i >= 1900) and (i < 2100) ):
        el = 0.0
        az = az0
    elif ( (i >= 2100) and (i < 2700) ):
        el = (i - 2100) * 0.1
        az = az0
    elif ( (i >= 2700) and (i < 3600) ):
        el = 60.0
        az = 360.0 - (i - 2700) * 0.1
    elif ( (i >= 3600) and (i < 3800) ):
        el = 60.0
        az = 270.0
    elif ( (i >= 3800) and (i < 4700) ):
        el = 60.0
        az = 270.0 - (i - 3800) * 0.1
    else:
        el = 60.0
        az = 180.0
```

```
    # time t
    t = t_start + i * 12.0 * u_hr

    # printing positions of the Sun, planets, and asteroids
    if (i % 10 == 0):
        print (f'Now, making a plot for {t}...')

    # settings for plot
    ax.set_xlim (-6.0, +6.0)
    ax.set_ylim (-6.0, +6.0)
    ax.set_zlim (-2.0, +2.0)
    ax.set_box_aspect ( (6.0, 6.0, 2.0) )

    # viewing angles of camera
    ax.view_init (elev=el, azim=az)

    # using black background colour
    fig.set_facecolor ('black')
    ax.set_facecolor ('black')
    ax.grid (False)
    ax.w_xaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_yaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))
    ax.w_zaxis.set_pane_color ((0.0, 0.0, 0.0, 0.0))

    # plotting the Sun
    sun = make_sphere (list_major[0][0][i], \
                       list_major[0][1][i], \
                       list_major[0][2][i], \
                       0.25, 'yellow')

    # plotting Mercury
    mercury = make_sphere (list_major[1][0][i], \
                           list_major[1][1][i], \
                           list_major[1][2][i], \
                           0.05, 'cyan')

    # plotting Venus
    venus = make_sphere (list_major[2][0][i], \
                         list_major[2][1][i], \
                         list_major[2][2][i], \
                         0.15, 'gold')

    # plotting Earth
    earth = make_sphere (list_major[3][0][i], \
                         list_major[3][1][i], \
                         list_major[3][2][i], \
                         0.15, 'blue')

    # plotting Mars
    mars = make_sphere (list_major[4][0][i], \
                        list_major[4][1][i], \
                        list_major[4][2][i], \
                        0.15, 'red')

    # plotting Jupiter
    jupiter = make_sphere (list_major[5][0][i], \
                           list_major[5][1][i], \
                           list_major[5][2][i], \
```

```
                            0.15, 'bisque')

    # plotting asteroids
    for j in range (0, n_asteroids):
        asteroid = ax.scatter (list_asteroids[j][0][i], \
                               list_asteroids[j][1][i], \
                               list_asteroids[j][2][i], \
                               s=0.1, \
                               color='saddlebrown')

    # title
    title = ax.text2D (0.5, 0.95, f'Inner Solar System', \
                       color='white', \
                       horizontalalignment='center', \
                       transform=ax.transAxes)

    # plotting the time
    time = ax.text2D (0.5, 0.05, f'Date/Time: {t} (UTC)', \
                      color='white', \
                      horizontalalignment='center', \
                      transform=ax.transAxes)

    # image file
    file_image = f'{file_prefix}_{i:06d}.{file_ext}'
    fig.savefig (file_image, dpi=255)
```

Execute above script.

```
% ./ai202209_s04_42.py
```

Now, you have 5000 PNG files. Construct a MPEG-4 movie file from these 5000 PNG files using the command `ffmpeg`.

```
% ffmpeg5 -f image2 -start_number 0 -framerate 30 -i solsys_3d_struct3_%06d.png \
? -an -vcodec libx264 -pix_fmt yuv420p -threads 8 solsys_3d_struct3.mp4
% ls -l *.mp4
```

Play the movie file.

```
% mplayer solsys_3d_struct3.mp4
```

19 For your further reading

Read the official document of Matplotlib to learn more about it.

- Matplotlib: `https://matplotlib.org/`

20 Assignment

1. Visit the official website of Matplotlib.
    - Read "Examples", "Tutorials", and "Users Guide".
    - Summarise basic usage of Matplotlib.
2. Visit web page of Kamogata/Kiso/Kyoto Wide-field Survey.
    - `http://kws.cetus-net.org/~maehara/VSdata.py`

(a) Search for V-band photometric measurements of Mira. Download the data, and make a lightcurve of Mira. Show the plot you made. Show the source code of your Python script.

(b) Search for V-band photometric measurements of δ Cep. Download the data, and make a lightcurve of δ Cep. Show the plot you made. Show the source code of your Python script.

(c) Search for V-band photometric measurements of RR Lyrae. Download the data, and make a lightcurve of RR Lyrae. Show the plot you made. Show the source code of your Python script.

(d) Search for V-band photometric measurements of δ Sct. Download the data, and make a lightcurve of δ Sct. Show the plot you made. Show the source code of your Python script.

(e) Search for V-band photometric measurements of T Tau. Download the data, and make a lightcurve of T Tau. Show the plot you made. Show the source code of your Python script.

3. Download Yale Bright Star Catalogue from following website.

   - `http://cdsarc.u-strasbg.fr/viz-bin/Cat?V/50`

   (a) Make a histogram of V-band apparent magnitude distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (b) Make a histogram of $(U - B)$ colour index distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (c) Make a histogram of $(B - V)$ colour index distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (d) Make a histogram of $(R - I)$ colour index distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (e) Make a histogram of distance distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (f) Make a histogram of Galactic latitude distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

   (g) Make a histogram of Galactic longitude distribution of stars in Yale Bright Star Catalogue. Show the histogram you made. Show the source code of your Python script.

4. Visit following website and download Hipparcos catalogue.

   - `https://cdsarc.unistra.fr/viz-bin/cat/I/311`

   (a) Make a Python script to extract information from Hipparcos catalogue.

   (b) Make a simple HR-diagram. Show the diagram you made. Show the source code of your Python script.

   (c) Use different marker size for different apparent brightness of star, and update your HR-diagram. Show the diagram you made. Show the source code of your Python script.

   (d) Which area on the plot do you expect to find main-sequence stars? Explain why.

   (e) Which area on the plot do you expect to find horizontal branch stars? Explain why.

   (f) Which area on the plot do you expect to find AGB stars? Explain why.

   (g) Which area on the plot do you expect to find giants? Explain why.

   (h) Which area on the plot do you expect to find white dwarfs? Explain why.

   (i) Which area on the plot do you expect to find brown dwarfs? Explain why.

   (j) Which area on the plot do you expect to find Cephieds? Explain why.

   (k) Which area on the plot do you expect to find RR Lyrae? Explain why.

   (l) Explain what you can learn from the plot.