Astroinformatics 2022
Session 11: Periodicity Analysis 1 (Phase Dispersion Minimisation)

Kinoshita Daisuke

28 November 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 try periodicity analysis for time-series data using Phase Dispersion Minimisation (PDM). PDM is a period search technique widely used for astronomy. First, we generate synthetic time-series data sets, and try period searches. Then, we apply the technique to the real data.

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) "`s11`". (Fig. 3) Choose the file "`ai202209_s11.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

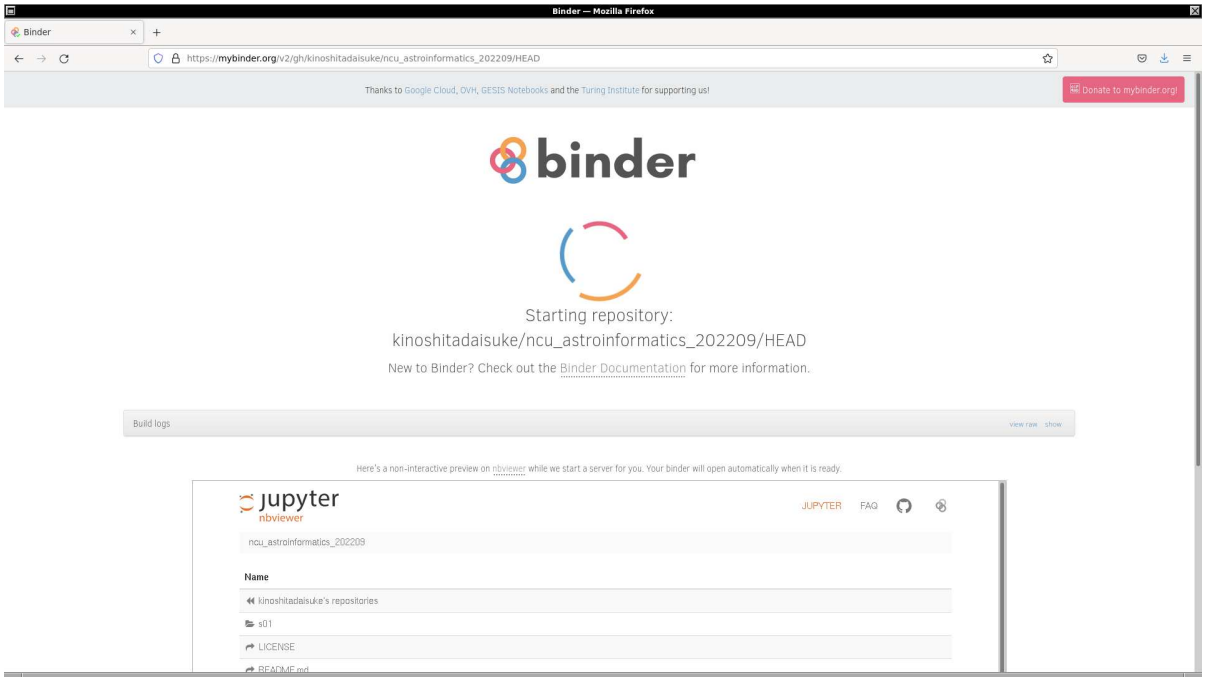

Figure 1: Using Binder to execute sample Python scripts for this session.

2 A very simple case

First, we generate a set of ideal synthetic data simulating time-series data with brightness change. We use sine curve to generate a data set.

2.1 Sine curve

Make a Python script to generate a set of ideal synthetic data using sine curve. Here is an example.

Python Code 1: ai202209_s11_00_00.py

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

#
# Time-stamp: <2022/11/27 19:23:38 (CST) daisuke>
#

# importing numpy module
import numpy

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

# constant
pi = numpy.pi

# units
u_sec = astropy.units.s
u_hr  = astropy.units.hr
```

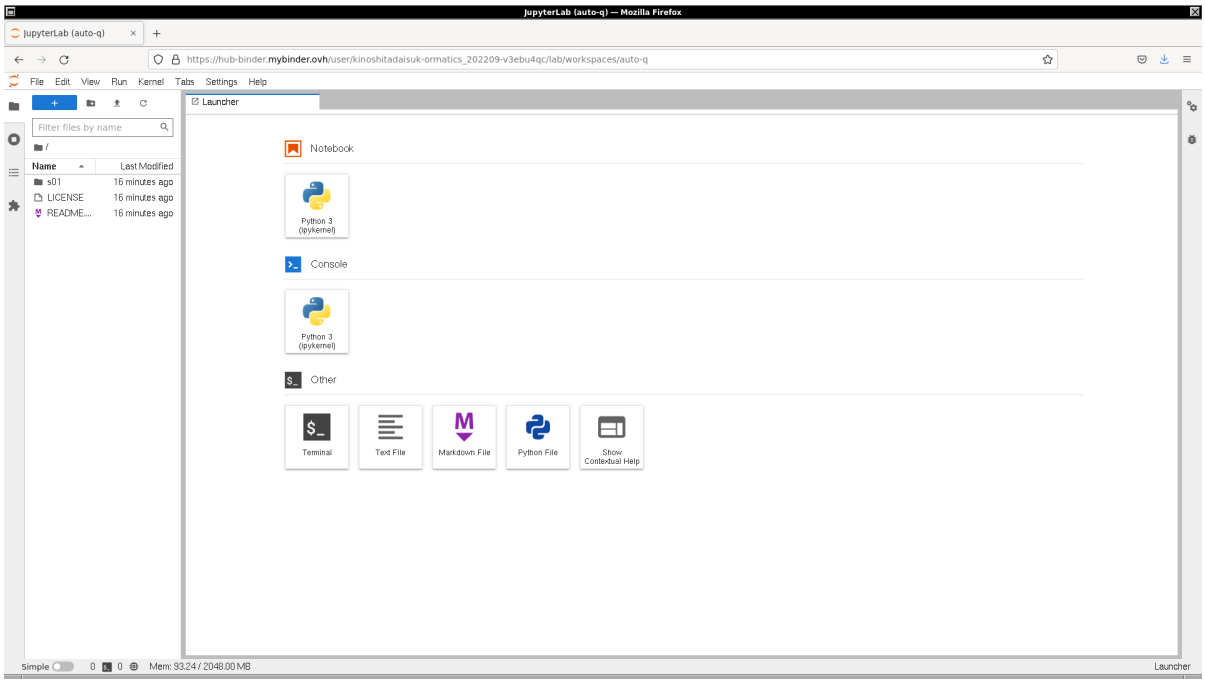

Figure 2: Using Binder to execute sample Python scripts for this session.

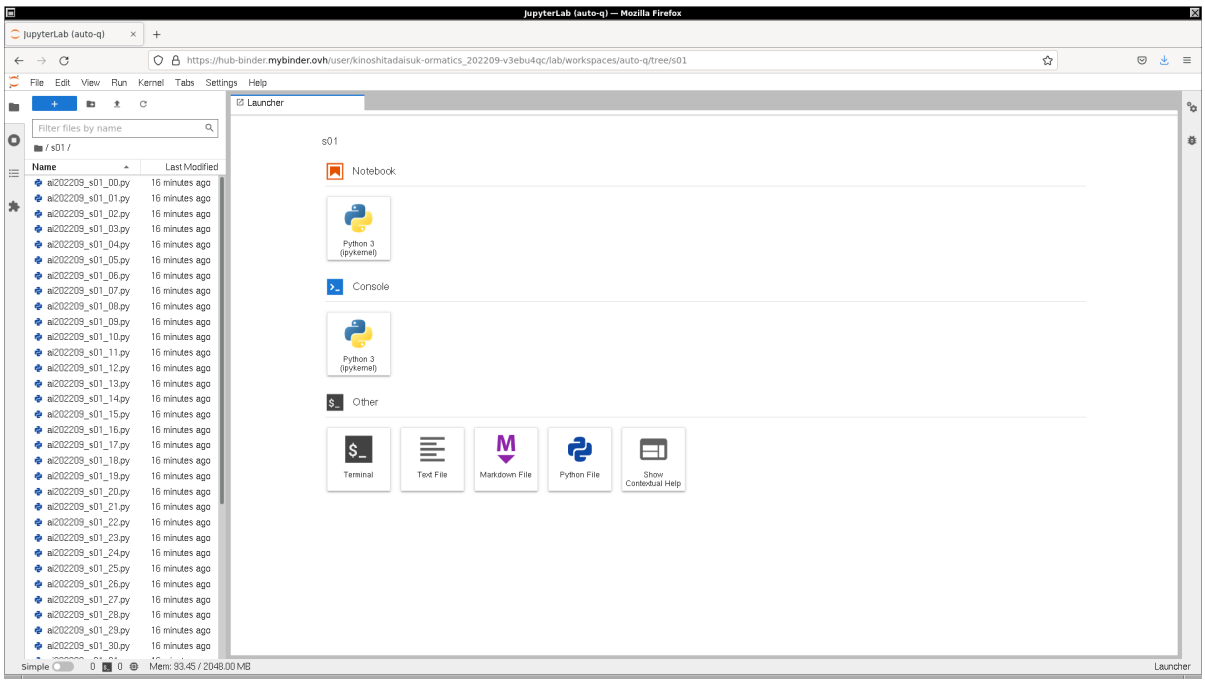

Figure 3: Using Binder to execute sample Python scripts for this session.

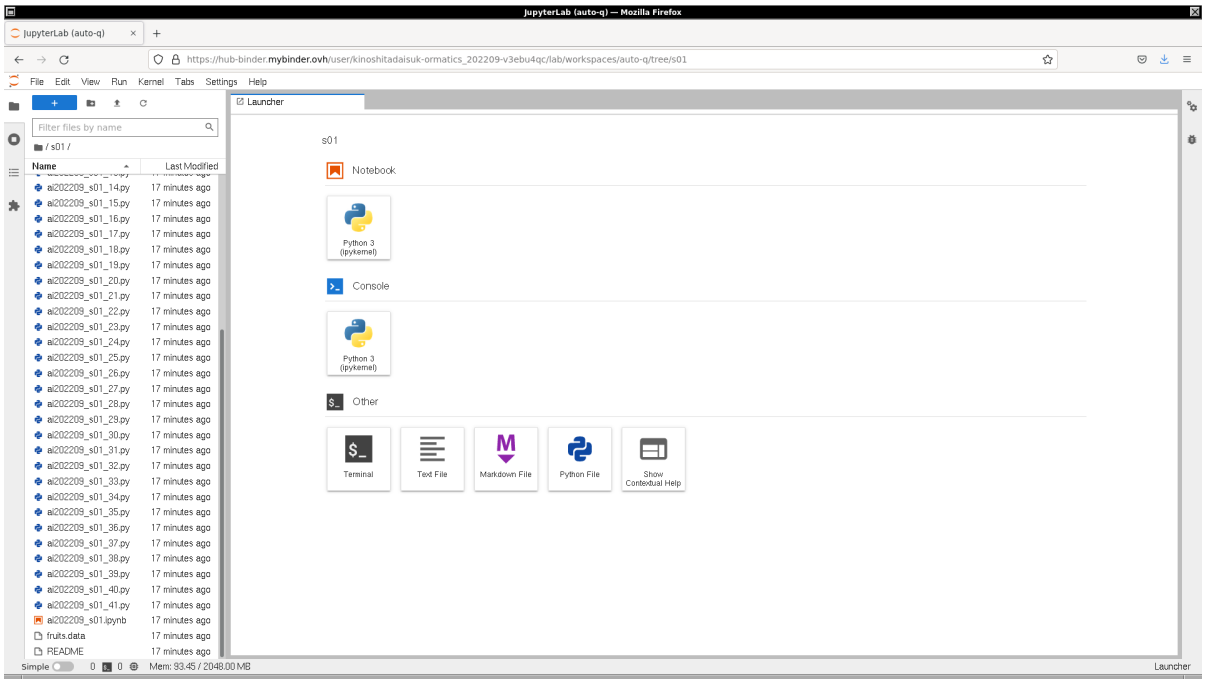

Figure 4: Using Binder to execute sample Python scripts for this session.

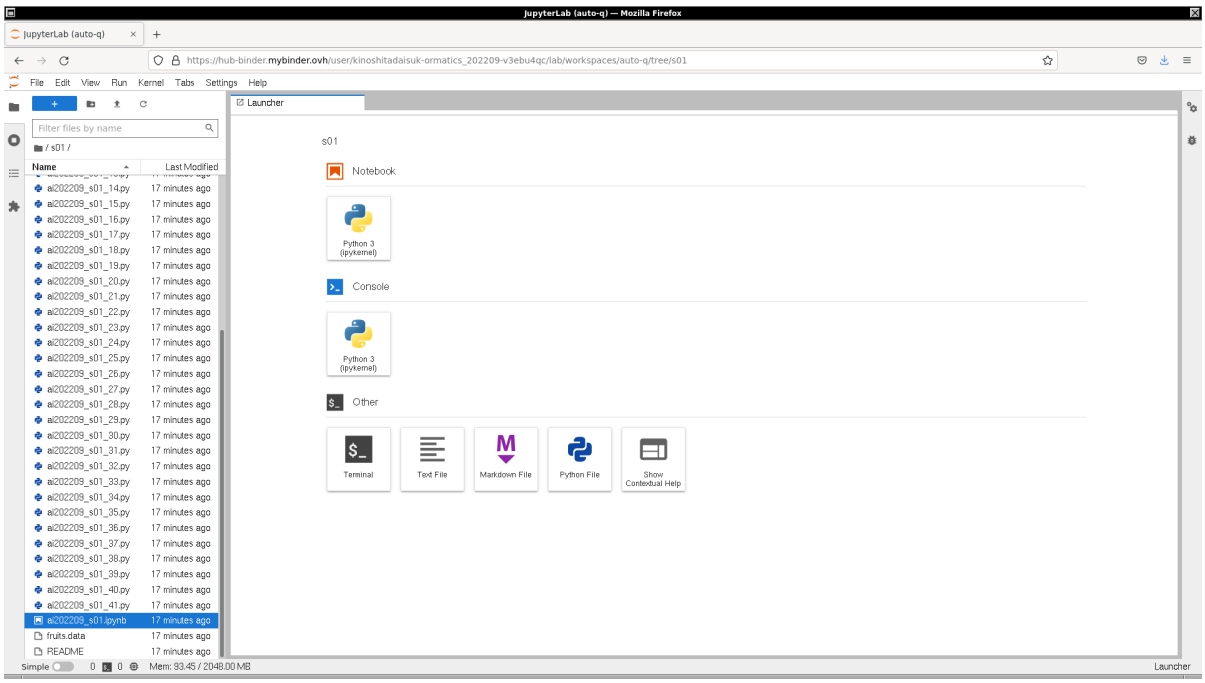

Figure 5: Using Binder to execute sample Python scripts for this session.

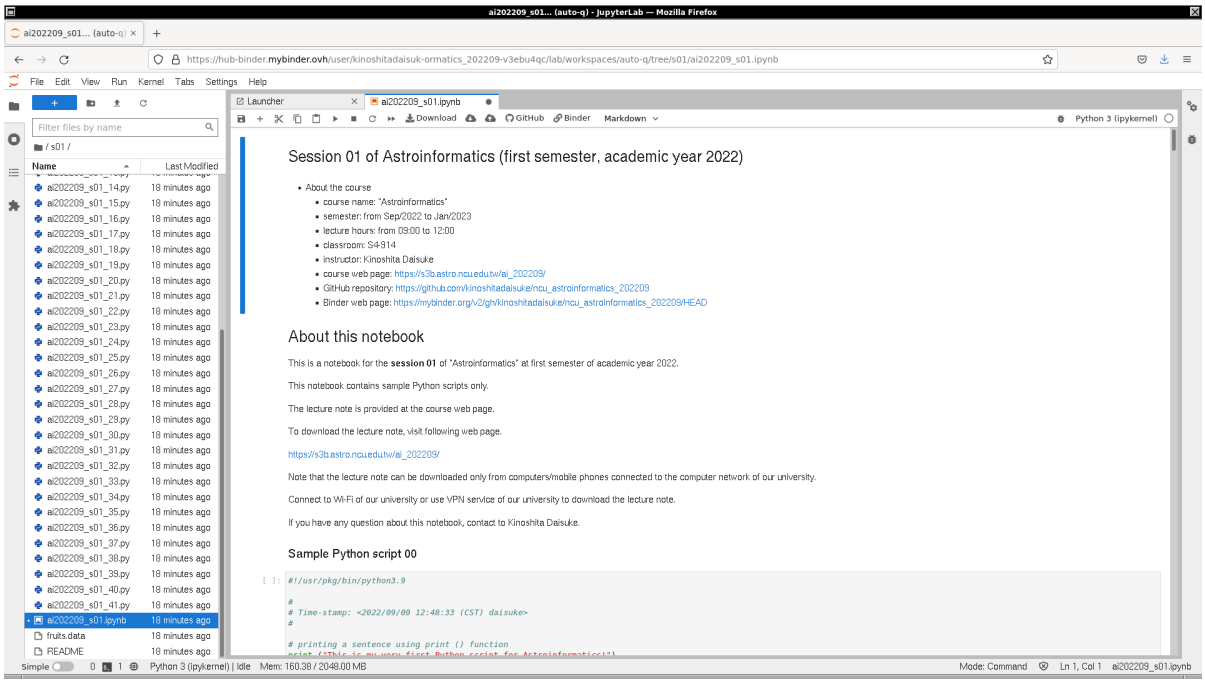

Figure 6: Using Binder to execute sample Python scripts for this session.

```
u_day = astropy.units.day

#
# parameters for synthetic data generation
#

# amplitude (mag)
A = 0.5

# period (hr)
P = 2.0 * u_hr

# phase
delta = 2.0 * pi * 0.25

# average magnitude
mag_mean = 20.0

# start of observation
date_start = '2022-12-01T12:00:00'
t_start    = astropy.time.Time (date_start, scale='utc', format='isot')
mjd_start  = t_start.mjd

# end of observation
date_end = '2022-12-01T20:00:00'
t_end    = astropy.time.Time (date_end, scale='utc', format='isot')
mjd_end  = t_end.mjd

# exposure time (sec)
exptime = 120.0 * u_sec

# overhead time (sec)
overhead = 10.0 * u_sec
```

```
# interval between exposures (sec)
interval = exptime + overhead

# printing input parameters
print (f"#")
print (f"#")
print (f"# Synthetic data for period search")
print (f"#")
print (f"#")
print (f"# input parameters")
print (f"#")
print (f"#  start of obs. = {t_start} = MJD {t_start.mjd}")
print (f"#  end of obs.   = {t_end} = MJD {t_end.mjd}")
print (f"#  amplitude     = {A} mag")
print (f"#  period        = {P}")
print (f"#  delta         = {delta}")
print (f"#  average mag.  = {mag_mean} mag")
print (f"#  exposure time = {exptime}")
print (f"#  overhead      = {overhead}")

# function for sine curve
def sine_curve (datetime, A, P, delta, mag_mean):
    # date/time
    t = astropy.time.Time (datetime)
    # calculation of magnitude at given time t
    mag = A * numpy.sin (2.0 * pi * t.mjd / P.to (u_day).value + delta) \
        + mag_mean
    # returning magnitude
    return (mag)

#
# generation of synthetic data
#

# time between start date/time and end date/time
n_data = int ( (mjd_end - mjd_start) * u_day / interval) + 1

# generating a numpy array for date/time
data_t = t_start + numpy.linspace (0.0, interval * (n_data - 1), n_data)

# generating a numpy array for magnitude
data_mag = sine_curve (data_t, A, P, delta, mag_mean)

# printing generated synthetic data
print (f"#")
print (f"# date/time, MJD, magnitude")
print (f"#")
for i in range (len (data_t)):
    print (f"{data_t[i]} {data_t[i].mjd:15.9f} {data_mag[i]:9.6f}")
```

Execute above script to generate a set of synthetic data for period search.

```
% chmod a+x ai202209_s11_00_00.py
% ./ai202209_s11_00_00.py
#
#
# Synthetic data for period search
#
```

```
#
# input parameters
#
#  start of obs. = 2022-12-01T12:00:00.000 = MJD 59914.5
#  end of obs.   = 2022-12-01T20:00:00.000 = MJD 59914.833333333336
#  amplitude     = 0.5 mag
#  period        = 2.0 h
#  delta         = 1.5707963267948966
#  average mag.  = 20.0 mag
#  exposure time = 120.0 s
#  overhead      = 10.0 s
#
# date/time, MJD, magnitude
#
2022-12-01T12:00:00.000 59914.500000000 20.500000
2022-12-01T12:02:10.000 59914.501504630 20.496786
2022-12-01T12:04:20.000 59914.503009259 20.487185
2022-12-01T12:06:30.000 59914.504513889 20.471321
2022-12-01T12:08:40.000 59914.506018519 20.449397

.....

2022-12-01T19:50:10.000 59914.826504630 20.435178
2022-12-01T19:52:20.000 59914.828009259 20.460252
2022-12-01T19:54:30.000 59914.829513889 20.479410
2022-12-01T19:56:40.000 59914.831018519 20.492404
2022-12-01T19:58:50.000 59914.832523148 20.499067
% ./ai202209_s11_00_00.py > ai202209_s11_00_00.data
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan 10042 Nov 27 17:15 ai202209_s11_00_00.data
% head -20 ai202209_s11_00_00.data
#
#
# Synthetic data for period search
#
#
# input parameters
#
#  start of obs. = 2022-12-01T12:00:00.000 = MJD 59914.5
#  end of obs.   = 2022-12-01T20:00:00.000 = MJD 59914.833333333336
#  amplitude     = 0.5 mag
#  period        = 2.0 h
#  delta         = 1.5707963267948966
#  average mag.  = 20.0 mag
#  exposure time = 120.0 s
#  overhead      = 10.0 s
#
# date/time, MJD, magnitude
#
2022-12-01T12:00:00.000 59914.500000000 20.500000
2022-12-01T12:02:10.000 59914.501504630 20.496786
```

Try following practice.

Practice 11-01

Generate a set of synthetic data for period search using cosine function.

2.2 Visualising time-series data

Make a Python script to visualise time-series data using Matplotlib. Here is an example.

Python Code 2: ai202209_s11_00_01.py

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

#
# Time-stamp: <2022/11/27 17:35:03 (CST) daisuke>
#

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

# importing numpy module
import numpy

# data file name
file_data = 'ai202209_s11_00_00.data'

# output file name
file_output = 'ai202209_s11_00_01.png'

# MJD offset
mjd_offset = 59000.0

# empty numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])

# opening file
with open (file_data, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)

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

# labels
ax.set_xlabel ('MJD - 59000 [day]')
ax.set_ylabel ('Apparent Magnitude [mag]')

# range
ax.set_ylim (19.3, 20.7)
```

```
ax.invert_yaxis ()

# plotting data
ax.plot (data_mjd - mjd_offset, data_mag, \
         linestyle='None', marker='o', markersize=3, color='blue', \
         label='synthetic time-series data')
ax.legend ()

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

Execute above script to make a plot of time-series data.

```
% chmod a+x ai202209_s11_00_01.py
% ./ai202209_s11_00_01.py
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan 69956 Nov 27 17:17 ai202209_s11_00_01.png
```

Display PNG file. (7)

```
% feh -dF ai202209_s11_00_01.png
```

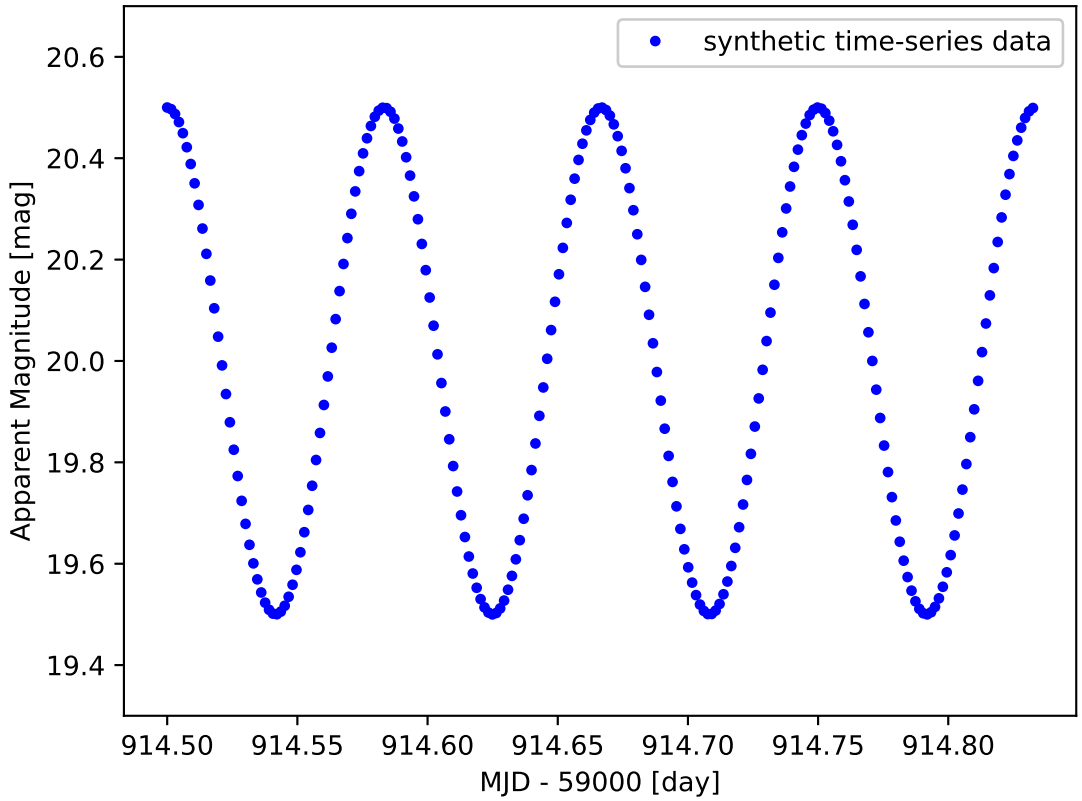

Figure 7: The visualisation of synthetic time-series data.

Try following practice.

Practice 11-02

Change the colour of data point to green, and make a plot again.

2.3 Folding time-series data using a trial period

Choose a trial period and fold the time-series data to construct a lightcurve. First, try $P = 1.0$ hr as a trial period.

Python Code 3: ai202209_s11_00_02.py

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

#
# Time-stamp: <2022/11/27 17:35:42 (CST) daisuke>
#

# importing numpy module
import numpy

# importing astropy module
import astropy.units

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

# units
u_hr  = astropy.units.hr
u_day = astropy.units.day

# data file name
file_data = 'ai202209_s11_00_00.data'

# output file name
file_output = 'ai202209_s11_00_02.png'

# trial period
p_hr = 1.0 * u_hr
p_day = p_hr.to (u_day)

# empty numpy arrays for storing data
data_mjd   = numpy.array ([])
data_phase = numpy.array ([])
data_mag   = numpy.array ([])

# opening file
with open (file_data, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str) = line.split ()
        # conversion from string into float
        mjd   = float (mjd_str)
        mag   = float (mag_str)
        # calculation of phase of variability using a trial period
```

```
        phase = mjd / p_day.value - int (mjd / p_day.value)
        # appending the data at the end of numpy arrays
        data_mjd   = numpy.append (data_mjd, mjd)
        data_phase = numpy.append (data_phase, phase)
        data_mag   = numpy.append (data_mag, mag)

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

# labels
ax.set_xlabel ('Phase')
ax.set_ylabel ('Apparent Magnitude [mag]')

# range
ax.set_ylim (19.3, 20.7)
ax.invert_yaxis ()

# plotting data
ax.plot (data_phase, data_mag, \
         linestyle='None', marker='o', markersize=3, color='blue', \
         label='lightcurve constructed from trial period $P=1$-hr')
ax.legend ()

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

Execute above script to hold the time-series data to construct a lightcurve using trial period of $P = 1.0$ hr.

```
% chmod a+x ai202209_s11_00_02.py
% ./ai202209_s11_00_02.py
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan 69956 Nov 27 17:17 ai202209_s11_00_01.png
-rw-r--r-- 1 daisuke taiwan 58886 Nov 27 17:18 ai202209_s11_00_02.png
```

Display PNG file. (8)

```
% feh -dF ai202209_s11_00_02.png
```

Constructed lightcurve does not look good, and a trial period of $P = 1.0$ hr is not the true period.
Try following practice.

Practice 11-03

Fold the time-series data to construct a lightcurve using a trial period of $P = 1.2$ hr.

Fold the time-series data using trial periods of $P = 1.0, 1.5, 2.0, 2.5, 3.0$, and 3.5 hr. Make a plot of 2×3 panels using "`.add_subplot ()`" method.

Python Code 4: ai202209_s11_00_03.py

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

#
# Time-stamp: <2022/11/27 17:43:35 (CST) daisuke>
#
```

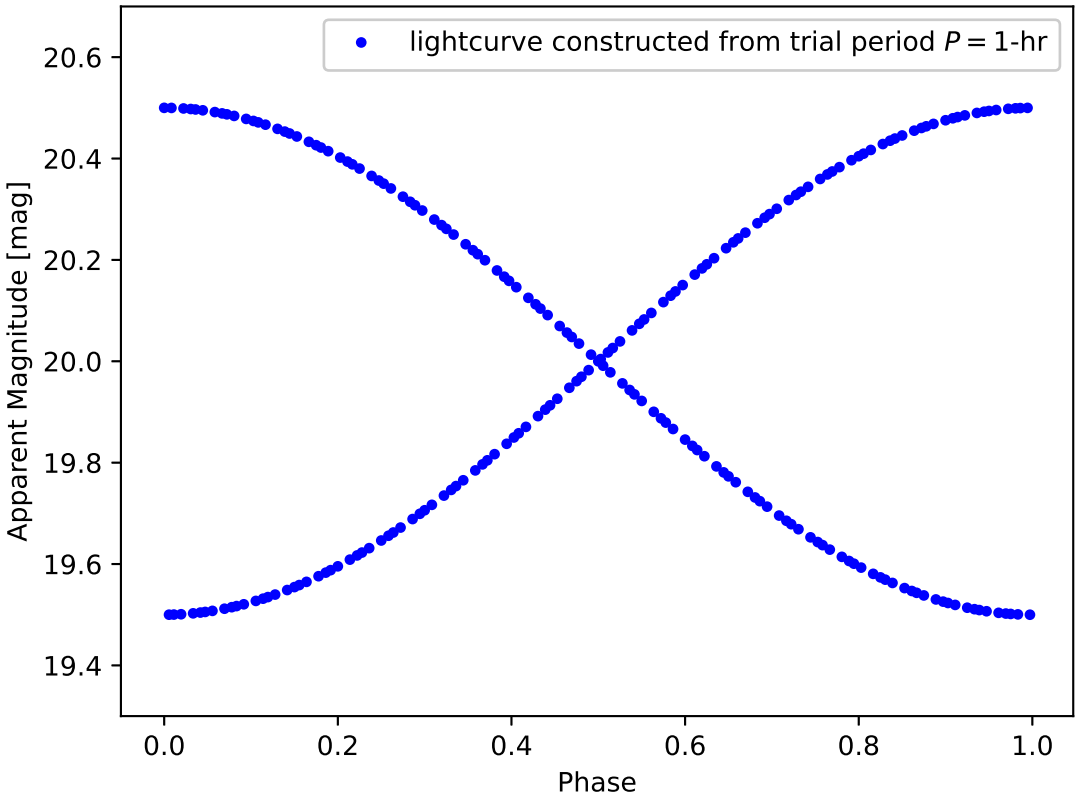

Figure 8: Constructed lightcurve using trial period of $P = 1.0$ hr.

```
# importing numpy module
import numpy

# importing astropy module
import astropy.units

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

# units
u_hr  = astropy.units.hr
u_day = astropy.units.day

# data file name
file_data = 'ai202209_s11_00_00.data'

# output file name
file_output = 'ai202209_s11_00_03.png'

# trial period
p_hr = numpy.array ([1.0, 1.5, 2.0, 2.5, 3.0, 3.5]) * u_hr
p_day = p_hr.to (u_day)

# empty numpy arrays for storing data
data_mjd   = numpy.array ([])
data_phase = numpy.array ([])
data_mag   = numpy.array ([])

# opening file
with open (file_data, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str) = line.split ()
        # conversion from string into float
        mjd   = float (mjd_str)
        mag   = float (mag_str)
        # appending the data at the end of numpy arrays
        data_mjd   = numpy.append (data_mjd, mjd)
        data_mag   = numpy.append (data_mag, mag)

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

# making a plot of 2x3 panels
for i in range (len (p_day)):
    # making a subplot
    subplot = 320 + int (i) + 1
    ax = fig.add_subplot (subplot)

    # calculation of phase
    p = p_day[i].value
```

```
    data_phase = numpy.array ([])
    for mjd in data_mjd:
        phase = mjd / p - int (mjd / p)
        data_phase = numpy.append (data_phase, phase)

    # labels
    ax.set_xlabel ('Phase')
    ax.set_ylabel ('Mag [mag]')

    # range
    ax.set_xlim (0.0, 1.0)
    ax.set_ylim (18.5, 20.7)
    ax.invert_yaxis ()

    # plotting data
    label = f"trial period of {p_hr[i]:3.1f}"
    ax.plot (data_phase, data_mag, \
             linestyle='None', marker='o', markersize=3, color='blue', \
             label=label)
    ax.legend ()

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

Execute above script to hold the time-series data to construct lightcurves using trial periods of $P = 1.0$, 1.5, 2.0, 2.5, 3.0, and 3.5 hr.

```
% chmod a+x ai202209_s11_00_03.py
% ./ai202209_s11_00_03.py
```

Display PNG file. (9)

```
% feh -dF ai202209_s11_00_03.png
```

Constructed lightcurve using a trial period of $P = 2.0$ hr looks nice, and $P = 2.0$ hr seems to be the true period. Try following practice.

Practice 11-04

Fold the time-series data using trial periods of $P = 1.00, 1.25, 1.50, 1.75, 2.00, 2.25, 2.50, 2.75$, and 3.00 hr. Make a plot of 3×3 panels using "`.add_subplot ()`" method.

2.4 Period search using PDM

PDM (Phase Dispersion Minimisation) is a period search technique for time-series data. This technique can be applied to unevenly spaced data. Note that intervals between exposures for astronomical observations are not the same. It works for poorly sampled astronomical data with large gaps due to day and night cycles. The data points of folded lightcurve using assumed period are divided into bins, and the sum of variance for each bin are calculated to estimate the quality of folded lightcurve to find the true period.

For the details of PDM, read following documents.

- "Period determination using phase dispersion minimization"
  - Stellingwerf, R. F., 1978, ApJ, 224, 953.
  - `https://ui.adsabs.harvard.edu/abs/1978ApJ...224..953S/abstract`
- PDM2: Phase Dispersion Minimization: `https://ascl.net/2105.002`

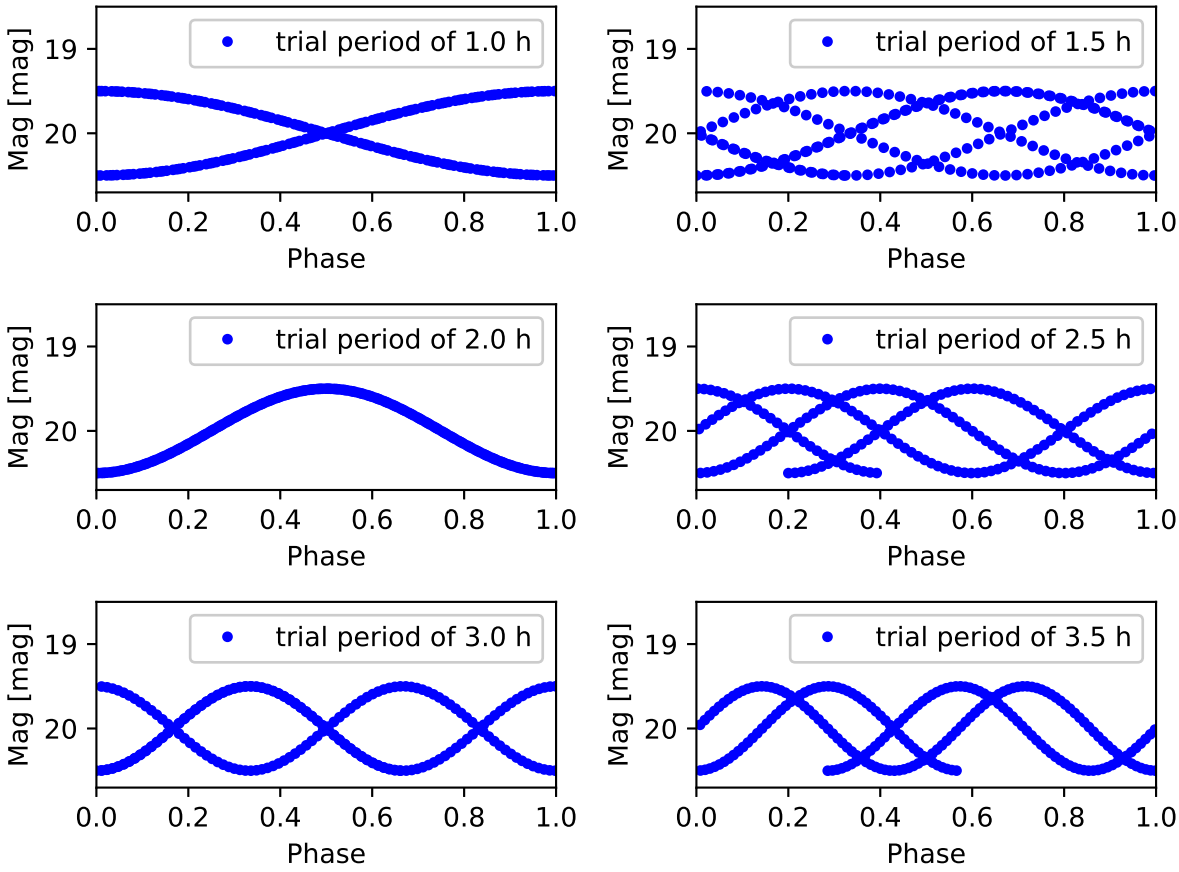

Figure 9: Constructed lightcurves using trial periods of $P = 1.0$, 1.5, 2.0, 2.5, 3.0, and 3.5 hr.

- PyAstronomy: `https://pyastronomy.readthedocs.io/`
  - The Phase Dispersion Minimization (PDM) analysis
    - `https://pyastronomy.readthedocs.io/en/latest/pyTimingDoc/pyPDMDoc/pdm.html`

Make a Python script to carry out PDM technique to find a period from time-series data. Here is an example.

Python Code 5: ai202209_s11_00_04.py

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

#
# Time-stamp: <2022/11/27 18:24:52 (CST) daisuke>
#

# importing numpy module
import numpy

# input file name
file_input = 'ai202209_s11_00_00.data'

# output file name
file_output = 'ai202209_s11_00_04.data'

# shortest trial period in minute and in day
period_min_min = 10.0
period_min_day = period_min_min / (60.0 * 24.0)

# longest trial period in minute and in day
period_max_min = 300.0
period_max_day = period_max_min / (60.0 * 24.0)

# step size of trial period in minute and in day
step_min = 0.1
step_day = step_min / (60.0 * 24.0)

# number of bins for PDM analysis
n_bins = 10

# empty numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading data line-by-line
    for line in fh_in:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
```

```
# opening file for writing
with open (file_output, 'w') as fh_out:
    # writing header to output file
    header  = f"#\n"
    header += f"# parameters for PDM analysis\n"
    header += f"#\n"
    header += f"# input file = {file_input}\n"
    header += f"# output file = {file_output}\n"
    header += f"# shortest trial period = {period_min_min} min\n"
    header += f"# longest trial period = {period_max_min} min\n"
    header += f"# step size of trial period = {step_min} min\n"
    header += f"# number of bins = {n_bins}\n"
    header += f"#\n"
    header += f"# results of PDM analysis\n"
    header += f"#\n"
    header += f"# trial period (day), trial period (hr), trial period (min), "
    header += f"total variance\n"
    header += f"#\n"
    fh_out.write (header)

    # initial value of trial period
    period_day = period_min_day

    # period search
    while (period_day < period_max_day):
        # calculation of phase with assumed period
        data_phase = numpy.array ([])
        for i in range ( len (data_mjd) ):
            phase = data_mjd[i] / period_day - int (data_mjd[i] / period_day)
            data_phase = numpy.append (data_phase, phase)

        # initialization of parameters
        total_variance = 0.0

        # calculation of variance
        for i in range (n_bins):
            # range of bin
            bin_min = i / n_bins
            bin_max = (i + 1) / n_bins

            # finding data within the bin
            data_bin = numpy.array ([])
            for j in range ( len (data_phase) ):
                if ( (data_phase[j] >= bin_min) and (data_phase[j] < bin_max) ):
                    data_bin = numpy.append (data_bin, data_mag[j])

            # if no data in the bin, then we skip.
            if (len (data_bin) == 0):
                continue

            # variance
            variance_in_bin = numpy.var (data_bin)
            # sum of variance
            total_variance += variance_in_bin

        # writing data to file
        output = f"{period_day:12.10f} {period_day * 24.0:12.8f} " \
            + f"{period_day * 24.0 * 60.0:12.6f} {total_variance:10.6f}\n"
```

```
        fh_out.write (output)

        # next trial period
        period_day += step_day
```

Execute above script to find the period of variability.

```
% chmod a+x ai202209_s11_00_04.py
% ./ai202209_s11_00_04.py
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan  11492 Nov 27 17:23 ai202209_s11_00_00.data
-rw-r--r-- 1 daisuke taiwan 145351 Nov 27 18:25 ai202209_s11_00_04.data
% head -20 ai202209_s11_00_04.data
#
# parameters for PDM analysis
#
# input file = ai202209_s11_00_00.data
# output file = ai202209_s11_00_04.data
# shortest trial period = 10.0 min
# longest trial period = 300.0 min
# step size of trial period = 0.1 min
# number of bins = 10
#
# results of PDM analysis
#
# trial period (day), trial period (hr), trial period (min), total variance
#
0.0069444444   0.16666667    10.000000   1.241321
0.0070138889   0.16833333    10.100000   1.246646
0.0070833333   0.17000000    10.200000   1.242564
0.0071527778   0.17166667    10.300000   1.247336
0.0072222222   0.17333333    10.400000   1.246411
0.0072916667   0.17500000    10.500000   1.248245
```

Try following practice.

Practice 11-05

Carry out PDM analysis from trial period of 5 min to trial period of 500 min.

Visualise the result of PDM analysis using Matplotlib.

Python Code 6: ai202209_s11_00_05.py

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

#
# Time-stamp: <2022/11/27 18:37:27 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'ai202209_s11_00_04.data'
```

```
# output file name
file_output = 'ai202209_s11_00_05.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', color='blue', linewidth=3, \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise result of PDM analysis.

```
% chmod a+x ai202209_s11_00_05.py
% ./ai202209_s11_00_05.py
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan  75938 Nov 27 17:34 ai202209_s11_00_01.png
-rw-r--r-- 1 daisuke taiwan  74066 Nov 27 17:35 ai202209_s11_00_02.png
-rw-r--r-- 1 daisuke taiwan 118759 Nov 27 17:43 ai202209_s11_00_03.png
-rw-r--r-- 1 daisuke taiwan  70834 Nov 27 18:32 ai202209_s11_00_05.png
```

Display PNG file. (10)

```
% feh -dF ai202209_s11_00_05.png
```

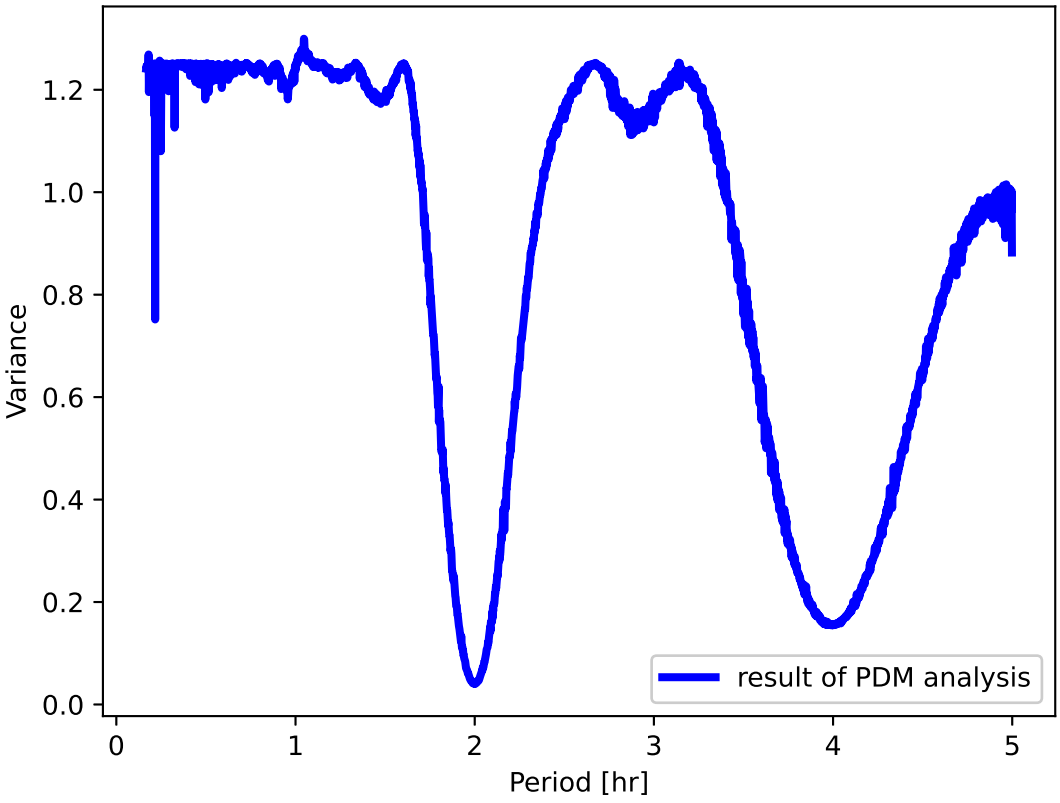

Figure 10: Result of PDM analysis.

There is a local minimum at $P = 2$ hr.
Try following practice.

Practice 11-06

Change the line style to dashed line and make the plot again.

Enlarge a local minimum at $P \sim 2$ hr, and make a plot again.

Python Code 7: ai202209_s11_00_06.py

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

#
# Time-stamp: <2022/11/27 18:43:56 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'ai202209_s11_00_04.data'

# output file name
file_output = 'ai202209_s11_00_06.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')
```

```
# axes
ax.set_xlim (1.5, 2.5)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', color='blue', linewidth=3, \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to make a plot around $P \sim 2$ hr.

```
% chmod a+x ai202209_s11_00_06.py
% ./ai202209_s11_00_06.py
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan  75938 Nov 27 17:34 ai202209_s11_00_01.png
-rw-r--r-- 1 daisuke taiwan  74066 Nov 27 17:35 ai202209_s11_00_02.png
-rw-r--r-- 1 daisuke taiwan 118759 Nov 27 17:43 ai202209_s11_00_03.png
-rw-r--r-- 1 daisuke taiwan  70834 Nov 27 18:32 ai202209_s11_00_05.png
-rw-r--r-- 1 daisuke taiwan  64206 Nov 27 18:41 ai202209_s11_00_06.png
```

Display PNG file. (11)

```
% feh -dF ai202209_s11_00_06.png
```

The local minimum at $P \sim 2$ hr seems to be fitted by a quadratic function.
Try following practice.

Practice 11-07

Change the line width to 2, and make the plot again.

Fit the local minimum at $P \sim 2$ hr of the result of PDM analysis by a quadratic function using least-squares method of SciPy. Here is an example.

Python Code 8: ai202209_s11_00_07.py

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

#
# Time-stamp: <2022/11/27 19:02:43 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize

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

# input data file
file_input = 'ai202209_s11_00_04.data'
```

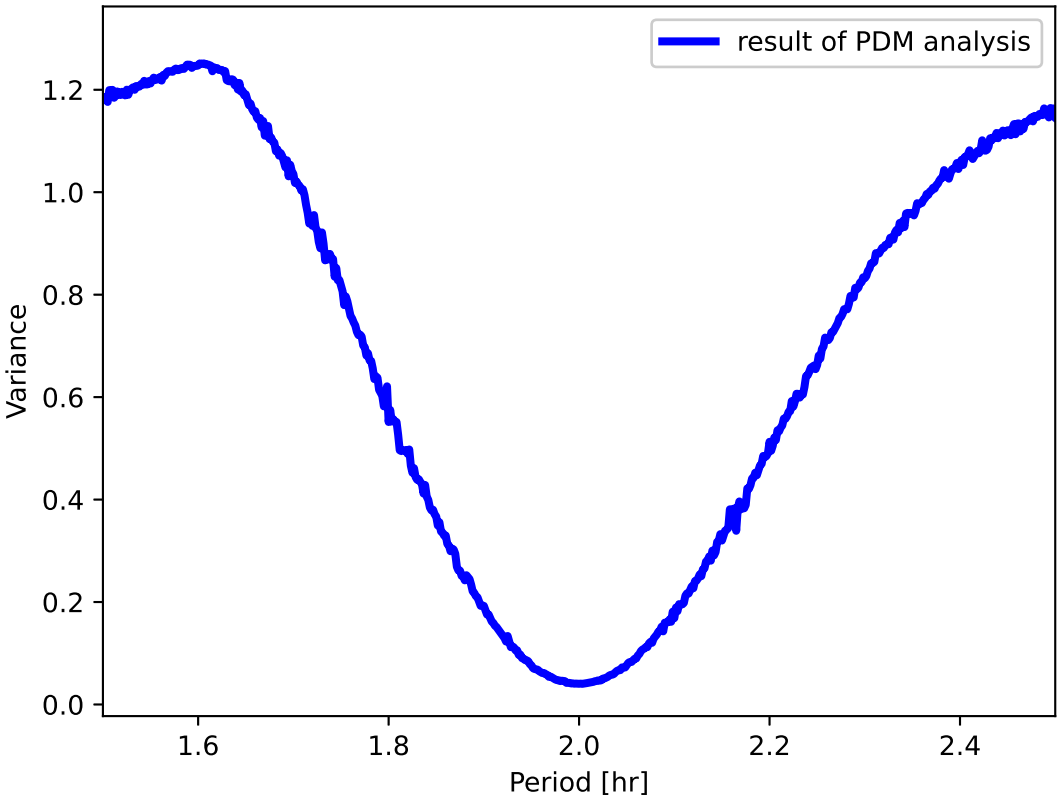

Figure 11: Result of PDM analysis at $P \sim 2$ hr.

```
# output figure file
file_output = 'ai202209_s11_00_07.png'

# range of data for fitting
x_min_fit = 1.9
x_max_fit = 2.1

# range of data for plotting
x_min_plot = 1.7
x_max_plot = 2.3

# empty numpy array for storing data
data_all_per = numpy.array ([])
data_all_var = numpy.array ([])
data_fit_per = numpy.array ([])
data_fit_var = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_all_per = numpy.append (data_all_per, per_hr)
        data_all_var = numpy.append (data_all_var, var)
        if ( (per_hr >= x_min_fit) and (per_hr <= x_max_fit) ):
            data_fit_per = numpy.append (data_fit_per, per_hr)
            data_fit_var = numpy.append (data_fit_var, var)

# initial values of coefficients of fitted function
a = 1.0
b = 1.0
c = 1.0

# function to be used for least-squares fitting
def func (x, a, b, c):
    y = a * (x - b)**2 + c
    return y

# least-squares fitting using scipy.optimize.curve_fit
popt, pcov = scipy.optimize.curve_fit (func, data_fit_per, data_fit_var)

# fitted coefficients
print ("popt:")
print (popt)

# covariance matrix
print ("pcov:")
print (pcov)
```

```
# fitted a and b
a_fitted = popt[0]
b_fitted = popt[1]
c_fitted = popt[2]

# degree of freedom
dof = len (data_fit_per) - 3
print (f"dof = {dof}")

# residual
residual     = data_fit_var - func (data_fit_per, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f"reduced chi^2 = {reduced_chi2}")

# errors of a, b, and c
a_err = numpy.sqrt (pcov[0][0])
b_err = numpy.sqrt (pcov[1][1])
c_err = numpy.sqrt (pcov[2][2])
print (f"a = {a_fitted:8.5f} +/- {a_err:8.5f} ({a_err/a_fitted*100:6.2f} %)")
print (f"b = {b_fitted:8.5f} +/- {b_err:8.5f} ({b_err/b_fitted*100:6.2f} %)")
print (f"c = {c_fitted:8.5f} +/- {c_err:8.5f} ({c_err/c_fitted*100:6.2f} %)")

# fitted line
fitted_x = numpy.linspace (x_min_plot, x_max_plot, 10**3)
fitted_y = func (fitted_x, a_fitted, b_fitted, c_fitted)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# range of plot
ax.set_xlim (x_min_plot, x_max_plot)

# plotting a figure
ax.plot (data_all_per, data_all_var, \
         linestyle='-', color='blue', linewidth=5, \
         label='result of PDM analysis')
ax.plot (fitted_x, fitted_y, \
         linestyle=':', color='red', linewidth=3, \
         label='fitted curve by curve_fit')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=225)
```

Execute above script to find the best fit period using least-squares method.

```
% chmod a+x ai202209_s11_00_07.py
% ./ai202209_s11_00_07.py
popt:
[14.33806225  2.00071624  0.04095186]
pcov:
[[ 7.34738928e-03 -3.67028123e-07 -2.48957167e-05]
 [-3.67028123e-07  2.42370134e-08  7.46193919e-10]
```

```
 [-2.48957167e-05  7.46193919e-10  1.51857704e-07]]
dof = 118
reduced chi^2 = 8.166464293385666e-06
a = 14.33806 +/-  0.08572 (  0.60 %)
b =  2.00072 +/-  0.00016 (  0.01 %)
c =  0.04095 +/-  0.00039 (  0.95 %)
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan  75938 Nov 27 17:34 ai202209_s11_00_01.png
-rw-r--r-- 1 daisuke taiwan  74066 Nov 27 17:35 ai202209_s11_00_02.png
-rw-r--r-- 1 daisuke taiwan 118759 Nov 27 17:43 ai202209_s11_00_03.png
-rw-r--r-- 1 daisuke taiwan  70834 Nov 27 18:32 ai202209_s11_00_05.png
-rw-r--r-- 1 daisuke taiwan  64206 Nov 27 18:41 ai202209_s11_00_06.png
-rw-r--r-- 1 daisuke taiwan  86933 Nov 27 19:02 ai202209_s11_00_07.png
```

Display PNG file. (12)

```
% feh -dF ai202209_s11_00_07.png
```

The best fit location of the local minimum is found to be $P = 2.0007 \pm 0.0002$ hr. Hence, the best fit period is $P = 2.0007 \pm 0.0002$ hr.

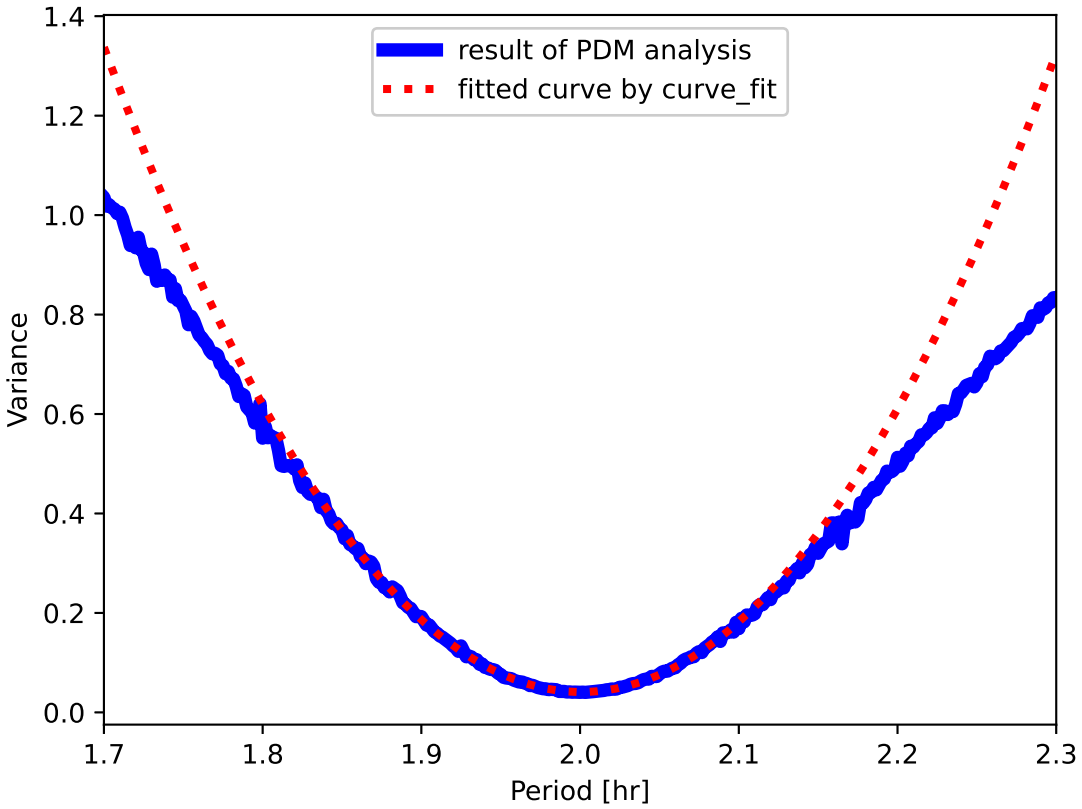

Figure 12: The result of least-squares method for finding the location of the local minimum at $P \sim 2$ hr.

Try following practice.

Practice 11-08

Change the initial values of fitting coefficients and execute the script again.

2.5 Folding time-series data using the best fit period

Make a Python script to fold the time-series data using the best fit period of $P = 2.0007 \pm 0.0002$ hr and construct a lightcurve. Here is an example.

Python Code 9: ai202209_s11_00_08.py

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

#
# Time-stamp: <2022/11/27 19:18:47 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'ai202209_s11_00_00.data'

# output file name
file_output = 'ai202209_s11_00_08.png'

# best fit period (day)
p_best = 2.0007 / 24

# empty numpy arrays for storing data
data_mjd   = numpy.array ([])
data_phase = numpy.array ([])
data_mag   = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str) = line.split ()
        # conversion from string into float
        mjd   = float (mjd_str)
        mag   = float (mag_str)
        phase = mjd / p_best - int (mjd / p_best)
        # appending the data at the end of numpy arrays
        data_mjd   = numpy.append (data_mjd, mjd)
        data_phase = numpy.append (data_phase, phase)
        data_mag   = numpy.append (data_mag, mag)

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

# labels
ax.set_xlabel ('Phase')
```

```
ax.set_ylabel ('Apparent Magnitude [mag]')

# range
ax.set_ylim (19.3, 20.7)
ax.invert_yaxis ()

# plotting data
ax.plot (data_phase, data_mag, \
         linestyle='None', marker='o', markersize=3, color='blue', \
         label='lightcurve constructed by the best fit period $P = 2.0007$ hr')
ax.legend ()

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

Execute above script to fold the time-series data to construct a lightcurve.

```
% chmod a+x ai202209_s11_00_08.py
% ./ai202209_s11_00_08.py
% ls -lF *.png
-rw-r--r-- 1 daisuke taiwan  75938 Nov 27 17:34 ai202209_s11_00_01.png
-rw-r--r-- 1 daisuke taiwan  74066 Nov 27 17:35 ai202209_s11_00_02.png
-rw-r--r-- 1 daisuke taiwan 118759 Nov 27 17:43 ai202209_s11_00_03.png
-rw-r--r-- 1 daisuke taiwan  70834 Nov 27 18:32 ai202209_s11_00_05.png
-rw-r--r-- 1 daisuke taiwan  64206 Nov 27 18:41 ai202209_s11_00_06.png
-rw-r--r-- 1 daisuke taiwan  86933 Nov 27 19:05 ai202209_s11_00_07.png
-rw-r--r-- 1 daisuke taiwan  71725 Nov 27 19:15 ai202209_s11_00_08.png
```

Display PNG file. (13)

```
% feh -dF ai202209_s11_00_08.png
```

Try following practice.

Practice 11-09

Change the colour of data point to red, and make the plot again.

3 Dealing with more realistic data

Real data always have error. Astronomical time-series photometric data are unevenly spaced data. The intervals between exposures are not always the same. There may be larger gaps in time-series data due to acquisition of calibration data. There are even larger gaps due to day and night cycles.

3.1 Generating synthetic data

Simulate astronomical optical photometric observations, and generate a set of realistic time-series measurements.

Python Code 10: ai202209_s11_01_00.py

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

#
# Time-stamp: <2022/11/27 20:09:09 (CST) daisuke>
#

# importing numpy module
```

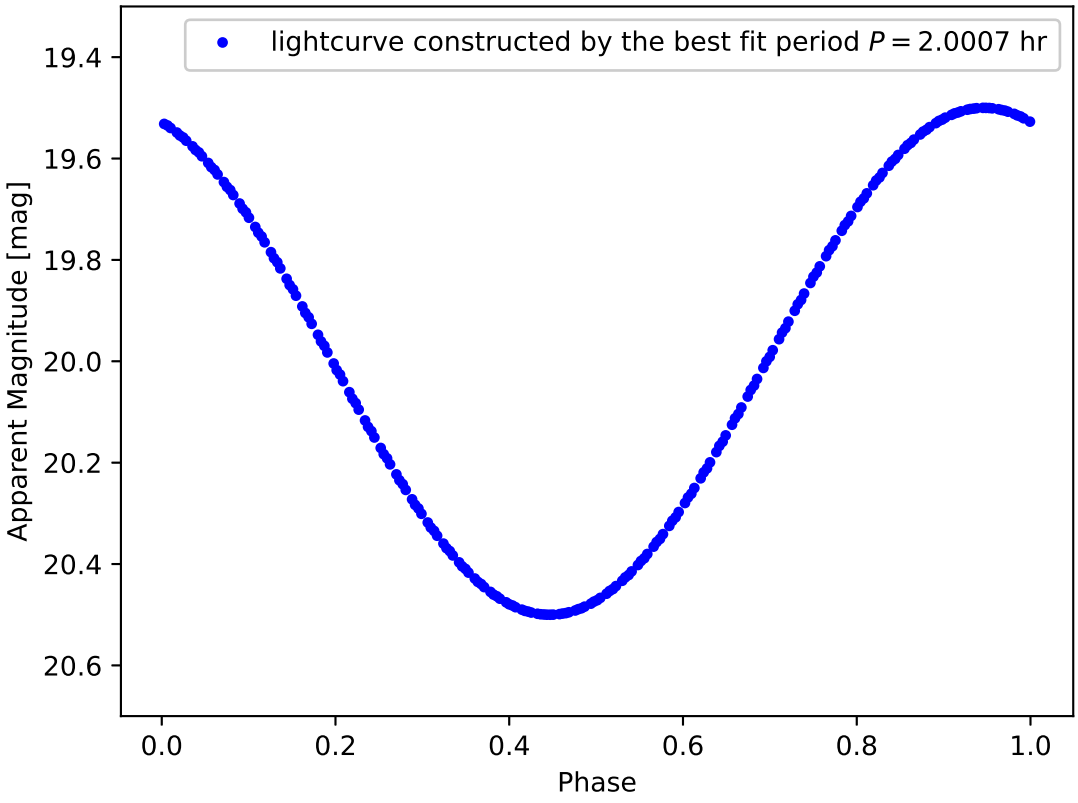

Figure 13: Folded lightcurve using the best fit period of $P \sim 2$ hr.

```
import numpy

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

# constant
pi = numpy.pi

# units
u_sec = astropy.units.s
u_hr  = astropy.units.hr
u_day = astropy.units.day

# random number generator
rng = numpy.random.default_rng ()

#
# parameters for synthetic data generation
#

# amplitude (mag)
A = 0.3

# period (hr)
P = 7.5 * u_hr

# phase
delta = 2.0 * pi * 0.35

# average magnitude
mag_mean = 21.0

# observable time
start_night = (20 - 8) / 24.0 # 12:00 UT
end_night   = (28 - 8) / 24.0 # 20:00 UT

# time for calibration data
start_calib = (24 - 8) / 24.0 # 16:00 UT
end_calib   = (25 - 8) / 24.0 # 17:00 UT

# start of observation
date_start = '2022-12-01T12:00:00'
t_start    = astropy.time.Time (date_start, scale='utc', format='isot')
mjd_start  = t_start.mjd

# end of observation
date_end = '2022-12-04T20:00:00'
t_end    = astropy.time.Time (date_end, scale='utc', format='isot')
mjd_end  = t_end.mjd

# exposure time (sec)
exptime = 180.0 * u_sec

# overhead time (sec)
overhead = 30.0 * u_sec

# interval between exposures (sec)
interval = exptime + overhead
```

```
# error
error_mean  = 0.00
error_sigma = 0.03

# printing input parameters
print (f"#")
print (f"#")
print (f"# Synthetic data for period search")
print (f"#")
print (f"#")
print (f"# input parameters")
print (f"#")
print (f"#  start of obs. = {t_start} = MJD {t_start.mjd}")
print (f"#  end of obs.   = {t_end} = MJD {t_end.mjd}")
print (f"#  amplitude     = {A} mag")
print (f"#  period        = {P}")
print (f"#  delta         = {delta}")
print (f"#  average mag.  = {mag_mean} mag")
print (f"#  exposure time = {exptime}")
print (f"#  overhead      = {overhead}")

# function for sine curve
def sine_curve (mjd, A, P, delta, mag_mean):
    # calculation of magnitude at given time t
    mag = A * numpy.sin (2.0 * pi * mjd / P.to (u_day).value + delta) \
        + mag_mean
    # returning magnitude
    return (mag)

#
# generation of synthetic data
#

print (f"#")
print (f"# date/time, MJD, magnitude, error")
print (f"#")
t = t_start
while (t <= t_end):
    # MJD
    mjd = t.mjd
    # fractional day
    fractional_day = mjd - int (mjd)
    # error
    err = rng.normal (loc=error_mean, scale=error_sigma)
    # apparent magnitude
    mag = sine_curve (mjd, A, P, delta, mag_mean) + err
    # printing data
    if ( ( (fractional_day >= start_night) \
           and (fractional_day <= end_night) ) \
         and ( (fractional_day < start_calib) \
               or (fractional_day > end_calib) ) ):
        print (f"{t} {mjd:15.9f} {mag:9.6f} {abs (err):9.6f}")

    # calculation of time of next exposure
    t += interval + rng.uniform (0.0, 60.0) * u_sec
```

Execute above script to generate synthetic data for period search.

```
% chmod a+x ai202209_s11_01_00.py
% ./ai202209_s11_01_00.py > ai202209_s11_01_00.data
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan  11492 Nov 27 17:23 ai202209_s11_00_00.data
-rw-r--r-- 1 daisuke taiwan 145351 Nov 27 18:25 ai202209_s11_00_04.data
-rw-r--r-- 1 daisuke taiwan  25658 Nov 27 20:09 ai202209_s11_01_00.data
% head -20 ai202209_s11_01_00.data
#
#
# Synthetic data for period search
#
#
# input parameters
#
#  start of obs. = 2022-12-01T12:00:00.000 = MJD 59914.5
#  end of obs.   = 2022-12-04T20:00:00.000 = MJD 59917.833333333336
#  amplitude     = 0.3 mag
#  period        = 7.5 h
#  delta         = 2.199114857512855
#  average mag.  = 21.0 mag
#  exposure time = 180.0 s
#  overhead      = 30.0 s
#
# date/time, MJD, magnitude, error
#
2022-12-01T12:00:00.000 59914.500000000 20.680695  0.019305
2022-12-01T12:04:27.899 59914.503100688 20.715048  0.014466
```

Try following practice.

Practice 11-10

Change the period to $P = 8.5$ hr, and generate a set of synthetic data.

3.2 Visualising synthetic data

Make a Python script to visualise generated synthetic data. Here is an example.

Python Code 11: ai202209_s11_01_01.py

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

#
# Time-stamp: <2022/11/27 20:13:07 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'ai202209_s11_01_00.data'

# output file name
file_output = 'ai202209_s11_01_01.png'
```

```
# empty numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])
data_err = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
        data_err = numpy.append (data_err, err)

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

# labels
ax.set_xlabel ('MJD - 59000 [day]')
ax.set_ylabel ('Apparent Magnitude [mag]')

# range
ax.set_ylim (20.4, 21.5)
ax.invert_yaxis ()

# plotting data
ax.errorbar (data_mjd - 59000, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='blue', \
             ecolor='black', capsize=5, \
             label='synthetic time-series data')
ax.legend ()

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

Execute above script to visualise synthetic data.

```
% chmod a+x ai202209_s11_01_01.py
% ./ai202209_s11_01_01.py
```

Display PNG file. (14)

```
% feh -dF ai202209_s11_01_01.png
```

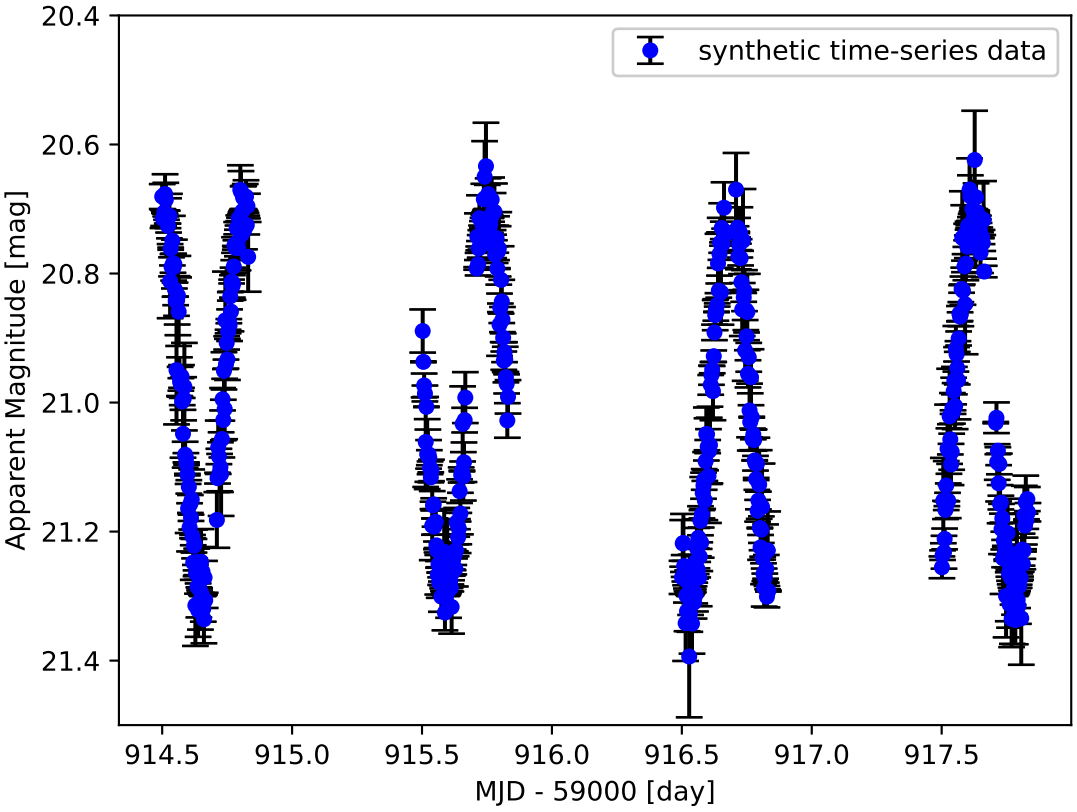

Figure 14: Synthetic time-series data simulating optical astronomical photometry.

There are gaps during the night, and there are even larger gaps due to day and night cycles.
Try following practice.

Practice 11-11

Change the marker size of data points and make the plot again.

3.3 Period search using PDM

Make a Python script to carry out period search using PDM. Here is an example.

Python Code 12: ai202209_s11_01_02.py

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

#
# Time-stamp: <2022/11/27 20:32:59 (CST) daisuke>
#

# importing numpy module
import numpy

# input file name
file_input = 'ai202209_s11_01_00.data'

# output file name
file_output = 'ai202209_s11_01_02.data'

# shortest trial period in minute and in day
period_min_min = 10.0
period_min_day = period_min_min / (60.0 * 24.0)

# longest trial period in minute and in day
period_max_min = 1000.0
period_max_day = period_max_min / (60.0 * 24.0)

# step size of trial period in minute and in day
step_min = 0.1
step_day = step_min / (60.0 * 24.0)

# number of bins
n_bins = 10

# empty numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh_in:
    # reading data line-by-line
    for line in fh_in:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
```

```
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
        data_err = numpy.append (data_err, err)

# opening file for writing
with open (file_output, 'w') as fh_out:
    # writing header to output file
    header  = f"#\n"
    header += f"# parameters for PDM analysis\n"
    header += f"#\n"
    header += f"# input file = {file_input}\n"
    header += f"# output file = {file_output}\n"
    header += f"# shortest trial period = {period_min_min} min\n"
    header += f"# longest trial period = {period_max_min} min\n"
    header += f"# step size of trial period = {step_min} min\n"
    header += f"# number of bins = {n_bins}\n"
    header += f"#\n"
    header += f"# results of PDM analysis\n"
    header += f"#\n"
    header += f"# trial period (day), trial period (hr), trial period (min), "
    header += f"total variance\n"
    header += f"#\n"
    fh_out.write (header)

    # initial value of trial period
    period_day = period_min_day

    # period search
    while (period_day < period_max_day):
        # calculation of phase with assumed period
        data_phase = numpy.array ([])
        for i in range ( len (data_mjd) ):
            phase = data_mjd[i] / period_day - int (data_mjd[i] / period_day)
            data_phase = numpy.append (data_phase, phase)

        # initialization of parameters
        total_variance = 0.0

        # calculation of variance
        for i in range (n_bins):
            # range of bin
            bin_min = i / n_bins
            bin_max = (i + 1) / n_bins

            # finding data within the bin
            data_bin = numpy.array ([])
            for j in range ( len (data_phase) ):
                if ( (data_phase[j] >= bin_min) and (data_phase[j] < bin_max) ):
                    data_bin = numpy.append (data_bin, data_mag[j])

            # if no data in the bin, then we skip.
            if (len (data_bin) == 0):
                continue

            # variance
```

```
            variance_in_bin = numpy.var (data_bin)
            # sum of variance
            total_variance += variance_in_bin

        # writing data to file
        output = f"{period_day:12.10f} {period_day * 24.0:12.8f} " \
            + f"{period_day * 24.0 * 60.0:12.6f} {total_variance:10.6f}\n"
        fh_out.write (output)

        # next trial period
        period_day += step_day
```

Execute above script to carry out period search using PDM.

```
% chmod a+x ai202209_s11_01_02.py
% ./ai202209_s11_01_02.py
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan  11492 Nov 27 17:23 ai202209_s11_00_00.data
-rw-r--r-- 1 daisuke taiwan 145351 Nov 27 18:25 ai202209_s11_00_04.data
-rw-r--r-- 1 daisuke taiwan  25658 Nov 27 20:09 ai202209_s11_01_00.data
-rw-r--r-- 1 daisuke taiwan 495352 Nov 27 20:24 ai202209_s11_01_02.data
% head -20 ai202209_s11_01_02.data
#
# parameters for PDM analysis
#
# input file = ai202209_s11_01_00.data
# output file = ai202209_s11_01_02.data
# shortest trial period = 10.0 min
# longest trial period = 1000.0 min
# step size of trial period = 0.1 min
# number of bins = 10
#
# results of PDM analysis
#
# trial period (day), trial period (hr), trial period (min), total variance
#
0.0069444444   0.16666667    10.000000   0.468314
0.0070138889   0.16833333    10.100000   0.469464
0.0070833333   0.17000000    10.200000   0.473535
0.0071527778   0.17166667    10.300000   0.473001
0.0072222222   0.17333333    10.400000   0.468183
0.0072916667   0.17500000    10.500000   0.473022
```

Try following practice.

Practice 11-12

Change the trial period range and carry out PDM again.

3.4 Visualising result of PDM analysis

Make a Python script to visualise result of PDM analysis. Here is an example.

Python Code 13: ai202209_s11_01_03.py

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

#
# Time-stamp: <2022/11/27 20:39:18 (CST) daisuke>
```

```
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_01_02.data'

# output file name
file_output = 'ai202209_s11_01_03.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise result of PDM analysis.

```
% chmod a+x ai202209_s11_01_03.py
% ./ai202209_s11_01_03.py
```

Display PNG file. (15)

```
% feh -dF ai202209_s11_01_03.png
```

A local minimum can be found at $P \sim 7.5$ hr.

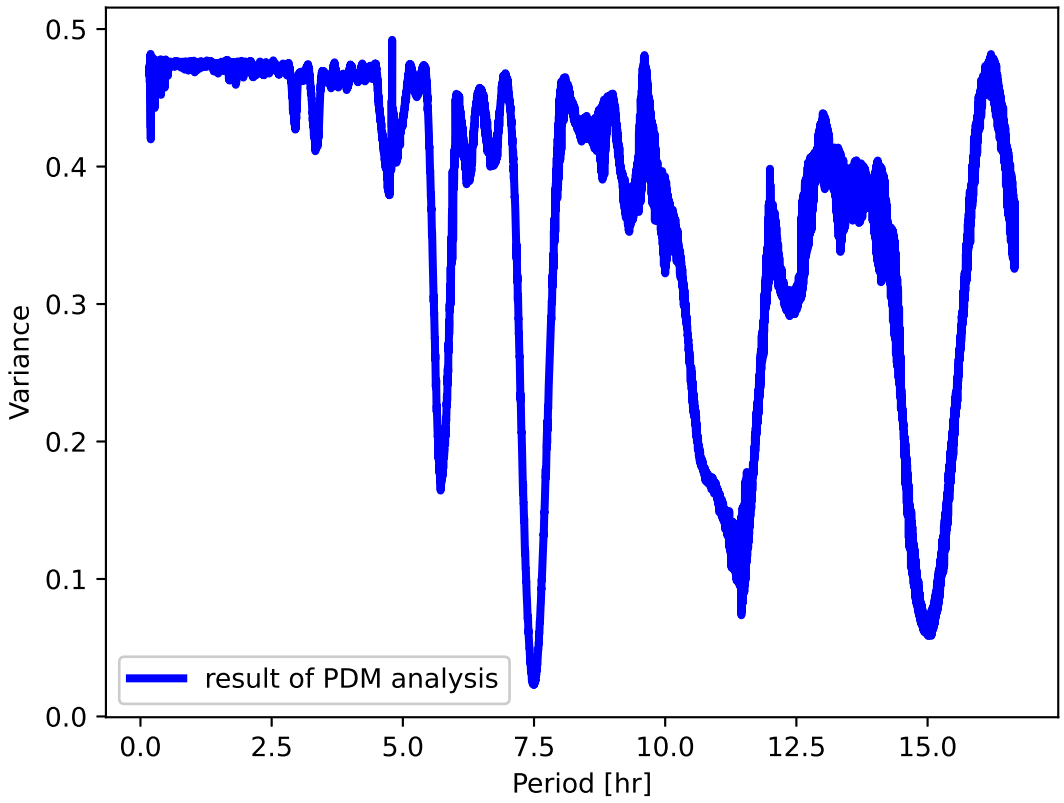

Figure 15: The result of PDM analysis.

Try following practice.

Practice 11-13

Change the line width and colour and make the plot again.

Enlarge the part around $P = 7.5$ hr, and make a plot again. Here is an example.

Python Code 14: ai202209_s11_01_04.py

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

#
# Time-stamp: <2022/11/27 20:50:38 (CST) daisuke>
#

# importing numpy module
import numpy

# importing matplotlib module
import matplotlib.figure
```

```
import matplotlib.backends.backend_agg

# input file name
file_input = 'ai202209_s11_01_02.data'

# output file name
file_output = 'ai202209_s11_01_04.png'

# range of x-value to plot
x_min = 6.5
x_max = 8.5

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (x_min, x_max)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to make a plot around $P = 7.5$ hr.

```
% chmod a+x ai202209_s11_01_04.py
% ./ai202209_s11_01_04.py
```

Display PNG file. (16)

```
% feh -dF ai202209_s11_01_04.png
```

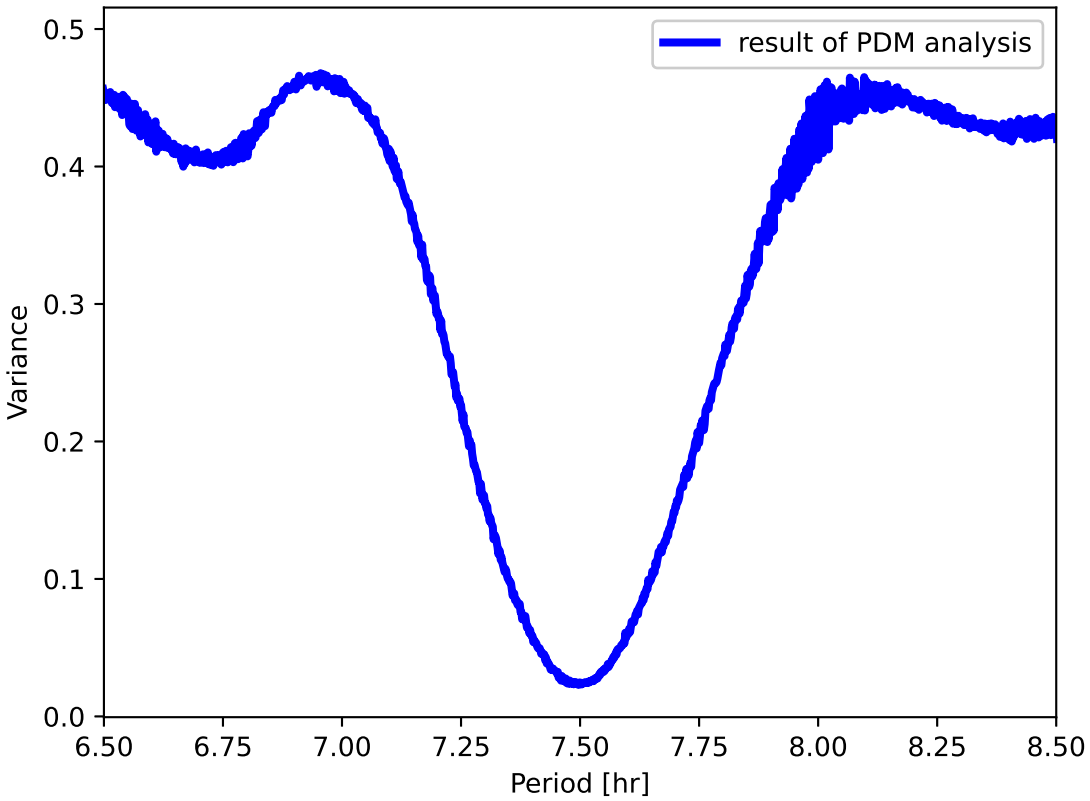

Figure 16: The result of PDM analysis around $P = 7.5$ hr.

Try following practice.

Practice 11-14

Change the range of X-axis, and re-generate the plot.

3.5 Finding the best fit period

Use least-squares method to find the best fit period. Here is an example.

Python Code 15: ai202209_s11_01_05.py

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

#
# Time-stamp: <2022/11/28 03:15:58 (CST) daisuke>
#

# importing numpy module
import numpy
```

```
# importing scipy module
import scipy.optimize

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

# input data file
file_input = 'ai202209_s11_01_02.data'

# output figure file
file_output = 'ai202209_s11_01_05.png'

# range of data for fitting
x_min = 7.3
x_max = 7.7

# empty numpy array for storing data
data_all_per = numpy.array ([])
data_all_var = numpy.array ([])
data_fit_per = numpy.array ([])
data_fit_var = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_all_per = numpy.append (data_all_per, per_hr)
        data_all_var = numpy.append (data_all_var, var)
        if ( (per_hr >= x_min) and (per_hr <= x_max) ):
            data_fit_per = numpy.append (data_fit_per, per_hr)
            data_fit_var = numpy.append (data_fit_var, var)

# initial values of coefficients of fitted function
a = 1.0
b = 1.0
c = 1.0

# function to be used for least-squares fitting
def func (x, a, b, c):
    y = a * (x - b)**2 + c
    return y

# least-squares fitting using scipy.optimize.curve_fit
popt, pcov = scipy.optimize.curve_fit (func, data_fit_per, data_fit_var)

# fitted coefficients
print ("popt:")
```

```
print (popt)

# covariance matrix
print ("pcov:")
print (pcov)

# fitted a and b
a_fitted = popt[0]
b_fitted = popt[1]
c_fitted = popt[2]

# degree of freedom
dof = len (data_fit_per) - 3
print (f"dof = {dof}")

# residual
residual = data_fit_var - func (data_fit_per, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f"reduced chi^2 = {reduced_chi2}")

# errors of a and b
a_err = numpy.sqrt (pcov[0][0])
b_err = numpy.sqrt (pcov[1][1])
c_err = numpy.sqrt (pcov[2][2])
print (f"a = {a_fitted:8.5f} +/- {a_err:8.5f} ({a_err/a_fitted*100:6.2f} %)")
print (f"b = {b_fitted:8.5f} +/- {b_err:8.5f} ({b_err/b_fitted*100:6.2f} %)")
print (f"c = {c_fitted:8.5f} +/- {c_err:8.5f} ({c_err/c_fitted*100:6.2f} %)")

# fitted line
fitted_x = numpy.linspace (x_min, x_max, 10**3)
fitted_y = func (fitted_x, a_fitted, b_fitted, c_fitted)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# range of plot
ax.set_xlim (x_min - 0.1, x_max + 0.1)
ax.set_ylim (0, func (x_max, a_fitted, b_fitted, c_fitted) * 1.5 )

# plotting a figure
ax.plot (data_all_per, data_all_var, \
         linestyle='-', linewidth=5, color='blue', \
         label='result of PDM analysis')
ax.plot (fitted_x, fitted_y, \
         linestyle=':', linewidth=3, color='red', \
         label='fitted curve by curve_fit')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=225)
```

Execute above script to carry out least-squares method to find the best fit period.

```
% chmod a+x ai202209_s11_01_05.py
% ./ai202209_s11_01_05.py
popt:
[3.2924525  7.49967918 0.02498009]
pcov:
[[ 1.56697970e-04  1.52689990e-08 -2.10670104e-06]
 [ 1.52689990e-08  3.88679282e-08 -1.23186892e-10]
 [-2.10670104e-06 -1.23186892e-10  5.09811253e-08]]
dof = 238
reduced chi^2 = 5.408568029784357e-06
a =  3.29245 +/-  0.01252 (  0.38 %)
b =  7.49968 +/-  0.00020 (  0.00 %)
c =  0.02498 +/-  0.00023 (  0.90 %)
```

Display PNG file. (17)

```
% feh -dF ai202209_s11_01_05.png
```

The best fit period is found to be $P = 7.4997 \pm 0.0002$ hr.

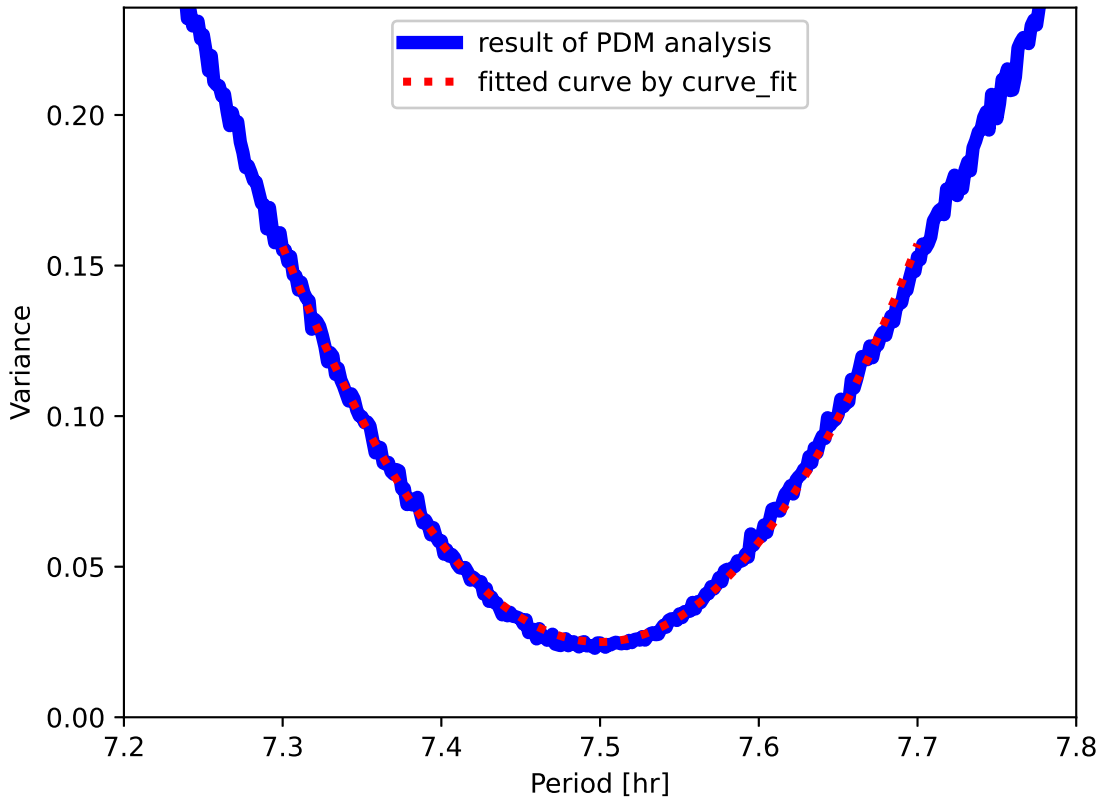

Figure 17: The result of PDM analysis around $P = 7.5$ hr.

Try following practice.

Practice 11-15

Change the range of the fitting and carry out least-squares method again.

3.6 Folding time-series data using the best fit period

Fold the time-series data using the best fit period of $P = 7.4997 \pm 0.0002$ hr. Here is an example.

Python Code 16: ai202209_s11_01_06.py

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

#
# Time-stamp: <2022/11/27 21:16:25 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'ai202209_s11_01_00.data'

# output file name
file_output = 'ai202209_s11_01_06.png'

# best fit period (day)
p_best = 7.4997 / 24.0

# empty numpy arrays for storing data
data_mjd   = numpy.array ([])
data_phase = numpy.array ([])
data_mag   = numpy.array ([])
data_err   = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (datetime, mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        phase = mjd / p_best - int (mjd / p_best)
        # appending the data at the end of numpy arrays
        data_mjd   = numpy.append (data_mjd, mjd)
        data_phase = numpy.append (data_phase, phase)
        data_mag   = numpy.append (data_mag, mag)
        data_err   = numpy.append (data_err, err)

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

```
# labels
ax.set_xlabel ('Phase')
ax.set_ylabel ('Apparent Magnitude [mag]')

# axes
ax.invert_yaxis ()

# plotting data
ax.errorbar (data_phase, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='blue', \
             ecolor='black', capsize=5, \
             label='folded lightcurve')
ax.errorbar (data_phase + 1, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='blue', \
             ecolor='black', capsize=5)
ax.legend ()

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

Execute above script to fold the time-series data to construct a lightcurve.

```
% chmod a+x ai202209_s11_01_06.py
% ./ai202209_s11_01_06.py
```

Display PNG file. (18)

```
% feh -dF ai202209_s11_01_06.png
```

Try following practice.

Practice 11-16

Change the marker size of data points and make the plot again.

4 Rotational lightcurve of a trans-Neptunian object

We try to find the period of the rotational lightcurve of a trans-Neptunian object.
Read following paper.

- "Physical properties of Trans-Neptunian Object (20000) Varuna"
  - Jewitt, D., Sheppard, S., AJ, 2002, 123, 2110.
  - `https://iopscience.iop.org/article/10.1086/339557/fulltext/`

4.1 Downloading data

Table 2 of above paper has time-series photometric measurement of a trans-Neptunian object Varuna. Download the electric file of Table 2.
Now, you should have the file named "`201498.tb2.txt`".

```
% ls -lF *.txt
-rw-r--r-- 1 daisuke taiwan 4872 Nov 27 22:26 201498.tb2.txt
% head 201498.tb2.txt

2019    Feb 17.2447     1,957.49983     19.913  3.450
```

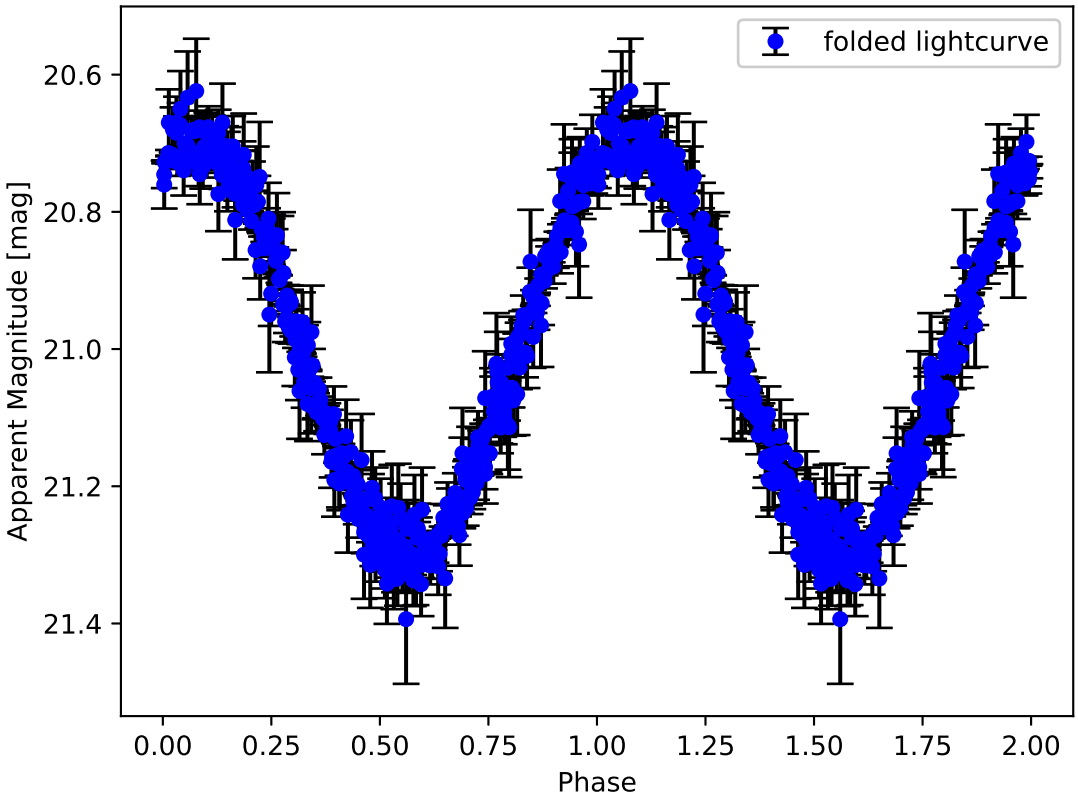

Figure 18: Folded lightcurve using the best fit period of $P = 7.4997$ hr.

```
2020    Feb 17.2484     1,957.50464     19.914  3.451
2021    Feb 17.2547     1,957.51099     19.898  3.435
2022    Feb 17.2592     1,957.51550     19.880  3.417
2023    Feb 17.2627     1,957.51904     19.860  3.397
2024    Feb 17.2662     1,957.52246     19.785  3.322
2025    Feb 17.2698     1,957.52600     19.776  3.313
2026    Feb 17.2733     1,957.52954     19.732  3.269
2027    Feb 17.2769     1,957.53320     19.689  3.226
```

4.2 Intra-night time-series photometry

Make a Python script to make intra-night time-series photometric measurements of (20000) Varuna on 17 Feb. 2001. Here is an example.

Python Code 17: ai202209_s11_02_00.py

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

#
# Time-stamp: <2022/11/27 22:44:43 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file
file_input = '201498.tb2.txt'

# output file
file_output = 'ai202209_s11_02_00.png'

# numpy arrays to store data
data_jd      = numpy.array ([])
data_mag_app = numpy.array ([])
data_mag_abs = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # skipping line if the line does not start with digits
        if not (line[0].isdigit):
            continue
        # skipping line if it is empty
        if (line.strip () == ''):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (frame_id_str, month_str, day_str, jd_str, mag_app_str, mag_abs_str) \
            = line.split ()
        # conversion from string into float
```

```
        frame_id = float (frame_id_str)
        day = float (day_str)
        jd_str = jd_str.replace (',', '')
        jd = float (jd_str)
        mag_app = float (mag_app_str)
        mag_abs = float (mag_abs_str)

        # only the data taken on 17 Feb are used for plotting
        if not ( (month_str == 'Feb') and (day_str[0:2] == '17') ):
            continue

        # appending the data at the end of numpy arrays
        data_jd      = numpy.append (data_jd, jd)
        data_mag_app = numpy.append (data_mag_app, mag_app)
        data_mag_abs = numpy.append (data_mag_abs, mag_abs)

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

# labels
ax.set_xlabel ('JD - 2451950')
ax.set_ylabel ('R-band Absolute Magnitude [mag]')

# axes
ax.invert_yaxis ()

# plotting a figure
ax.plot (data_jd - 1950.0, data_mag_app, \
         linestyle='None', marker='o', markersize=3, color='red', \
         label='(20000) Varuna')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=225)
```

Execute above script to fold the time-series data to construct a lightcurve.

```
% chmod a+x ai202209_s11_02_00.py
% ./ai202209_s11_02_00.py
```

Display PNG file. (19)

```
% feh -dF ai202209_s11_02_00.png
```

The brightness change is clearly seen. Compare the plot you have made with Fig. 1 of Jewitt and Sheppard (2002). Try following practice.

Practice 11-17

Change the marker shape, size, and colour and make the plot again.

4.3 Period search using PDM

Make a Python script to carry out period search using PDM. Here is an example.

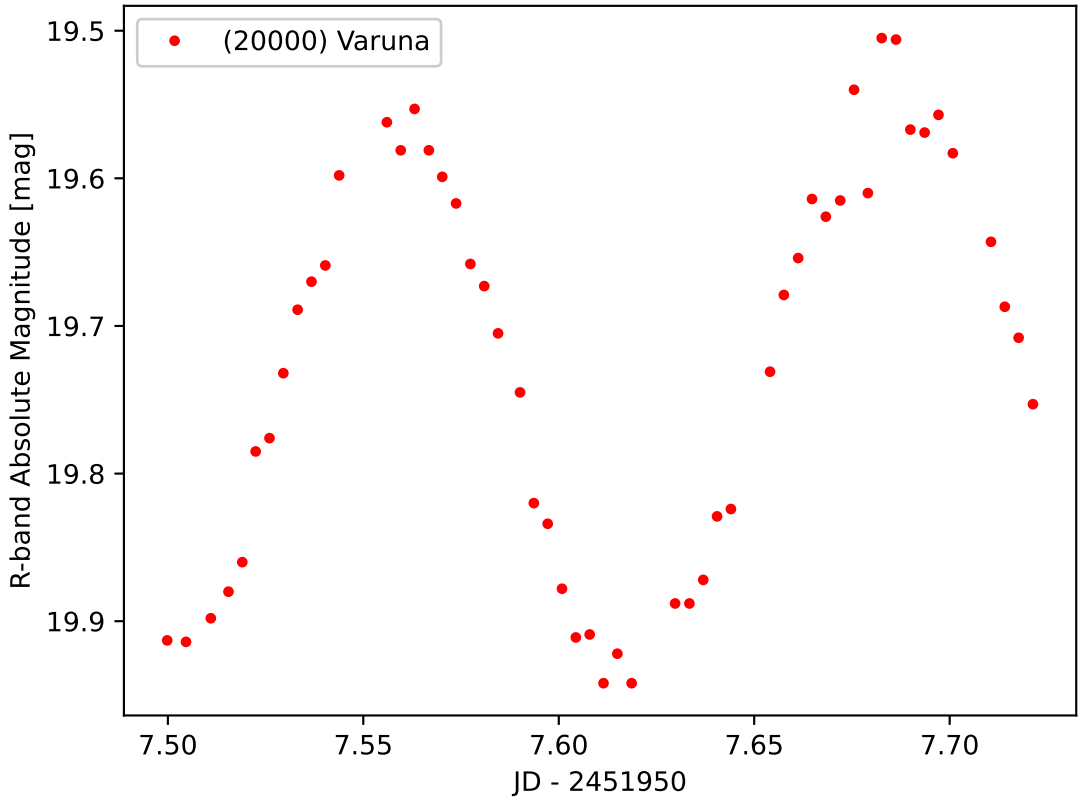

Figure 19: Intra-night time-series photometric measurements of (20000) Varuna on 17 Feb. 2001.

Python Code 18: ai202209_s11_02_01.py

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

#
# Time-stamp: <2022/11/27 22:54:34 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = '201498.tb2.txt'

# output file name
file_output = 'ai202209_s11_02_01.data'

# shortest trial period in minute and in day
period_min_min = 10.0
period_min_day = period_min_min / (60.0 * 24.0)

# longest trial period in minute and in day
period_max_min = 1000.0
period_max_day = period_max_min / (60.0 * 24.0)

# step size of trial period in minute and in day
step_min = 0.01
step_day = step_min / (60.0 * 24.0)

# number of bins
n_bins = 10

# numpy arrays to store data
data_jd      = numpy.array ([])
data_mag_app = numpy.array ([])
data_mag_abs = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # skipping line if the line does not start with digits
        if not (line[0].isdigit):
            continue
        # skipping line if it is empty
        if (line.strip () == ''):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (frame_id_str, month_str, day_str, jd_str, mag_app_str, mag_abs_str) \
            = line.split ()
        # conversion from string into float
```

```
        frame_id = float (frame_id_str)
        day = float (day_str)
        jd_str = jd_str.replace (',', '')
        jd = float (jd_str)
        mag_app = float (mag_app_str)
        mag_abs = float (mag_abs_str)

        # appending the data at the end of numpy arrays
        data_jd      = numpy.append (data_jd, jd)
        data_mag_app = numpy.append (data_mag_app, mag_app)
        data_mag_abs = numpy.append (data_mag_abs, mag_abs)

# opening file for writing
with open (file_output, 'w') as fh_out:
    # writing header to output file
    header  = f"#\n"
    header += f"# parameters for PDM analysis\n"
    header += f"#\n"
    header += f"# input file = {file_input}\n"
    header += f"# output file = {file_output}\n"
    header += f"# shortest trial period = {period_min_min} min\n"
    header += f"# longest trial period = {period_max_min} min\n"
    header += f"# step size of trial period = {step_min} min\n"
    header += f"# number of bins = {n_bins}\n"
    header += f"#\n"
    header += f"# results of PDM analysis\n"
    header += f"#\n"
    header += f"# trial period (day), trial period (hr), trial period (min), "
    header += f"total variance\n"
    header += f"#\n"
    fh_out.write (header)

    # initial value of trial period
    period_day = period_min_day

    # period search
    while (period_day < period_max_day):
        # calculation of phase with assumed period
        data_phase = numpy.array ([])
        for i in range ( len (data_jd) ):
            phase = (data_jd[i] - data_jd[0]) / period_day
            phase -= int (phase)
            data_phase = numpy.append (data_phase, phase)

        # initialization of parameter
        total_variance = 0.0

        # calculation of variance
        for i in range (n_bins):
            # range of bin
            bin_min = i / n_bins
            bin_max = (i + 1) / n_bins

            # finding data within the bin
            data_bin = numpy.array ([])
            for j in range ( len (data_phase) ):
                if ( (data_phase[j] >= bin_min) and (data_phase[j] < bin_max) ):
                    data_bin = numpy.append (data_bin, data_mag_abs[j])
```

```
            # if no data in the bin, then we skip.
            if (len (data_bin) == 0):
                continue

            # variance
            variance_in_bin = numpy.var (data_bin)
            # sum of variance
            total_variance += variance_in_bin

        # writing data to file
        output = f"{period_day:12.10f} {period_day * 24.0:12.8f} " \
            + f"{period_day * 24.0 * 60.0:12.6f} {total_variance:10.6f}\n"
        fh_out.write (output)

        # next trial period
        period_day += step_day
```

Execute above script to carry out period search using PDM.

```
% chmod a+x ai202209_s11_02_01.py
% ./ai202209_s11_02_01.py
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan   11492 Nov 27 17:23 ai202209_s11_00_00.data
-rw-r--r-- 1 daisuke taiwan  145351 Nov 27 18:25 ai202209_s11_00_04.data
-rw-r--r-- 1 daisuke taiwan   25658 Nov 27 20:09 ai202209_s11_01_00.data
-rw-r--r-- 1 daisuke taiwan  495352 Nov 27 20:34 ai202209_s11_01_02.data
-rw-r--r-- 1 daisuke taiwan 4950344 Nov 27 22:58 ai202209_s11_02_01.data
% head -20 ai202209_s11_02_01.data
#
# parameters for PDM analysis
#
# input file = 201498.tb2.txt
# output file = ai202209_s11_02_01.data
# shortest trial period = 10.0 min
# longest trial period = 1000.0 min
# step size of trial period = 0.01 min
# number of bins = 10
#
# results of PDM analysis
#
# trial period (day), trial period (hr), trial period (min), total variance
#
0.0069444444   0.16666667    10.000000   0.165003
0.0069513889   0.16683333    10.010000   0.184543
0.0069583333   0.16700000    10.020000   0.187221
0.0069652778   0.16716667    10.030000   0.173198
0.0069722222   0.16733333    10.040000   0.183261
0.0069791667   0.16750000    10.050000   0.164212
```

Try following practice.

Practice 11-18

Change the number of bins and carry out PDM analysis again.

4.4 Visualising result of PDM analysis

Make a Python script to visualise the result of PDM analysis. Here is an example.

Python Code 19: ai202209_s11_02_02.py

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

#
# Time-stamp: <2022/11/27 23:26:25 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_02_01.data'

# output file name
file_output = 'ai202209_s11_02_02.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis.

```
% chmod a+x ai202209_s11_02_02.py
% ./ai202209_s11_02_02.py
```

Display PNG file. (20)

```
% feh -dF ai202209_s11_02_02.png
```

A local minimum can be found at $P \sim 3.2$ hr.

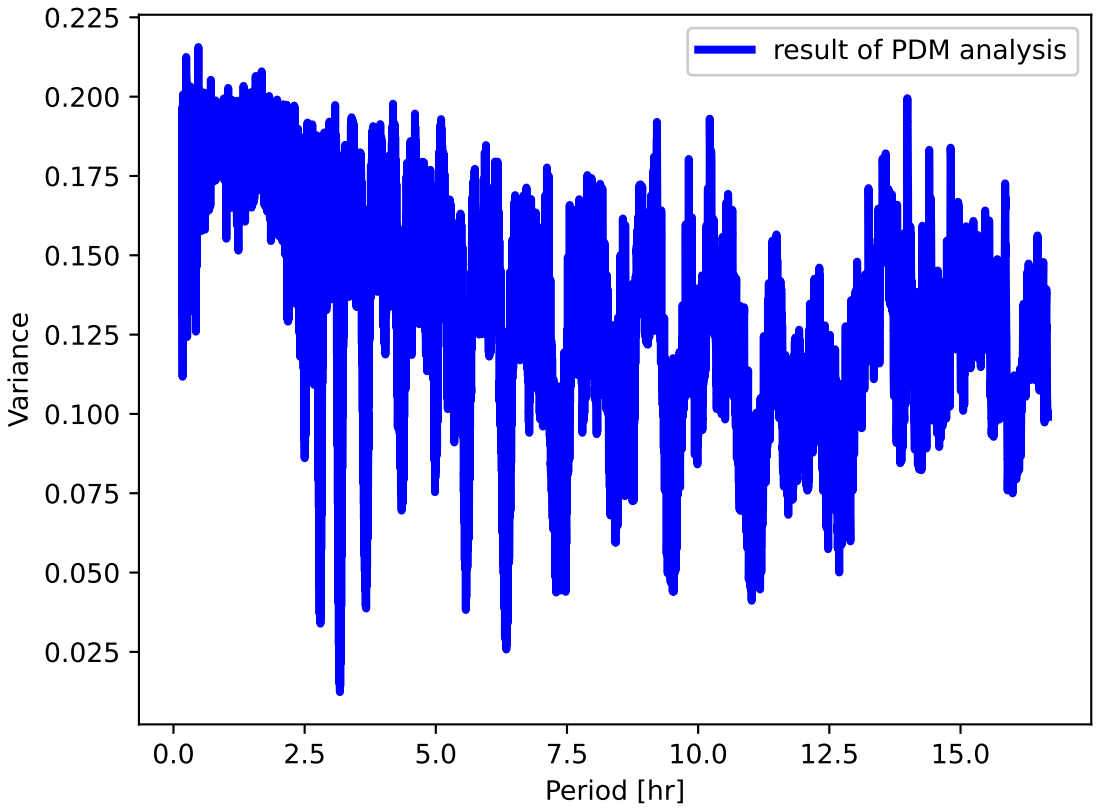

Figure 20: The result of PDM analysis for (20000) Varuna.

Try following practice.

Practice 11-19

Change the line width and make the plot again.

Make a plot again to show the region between $P = 2$ hr and $P = 4$ hr.

Python Code 20: ai202209_s11_02_03.py

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

#
```

```
# Time-stamp: <2022/11/27 23:29:27 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_02_01.data'

# output file name
file_output = 'ai202209_s11_02_03.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (2.0, 4.0)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis.

```
% chmod a+x ai202209_s11_02_03.py
% ./ai202209_s11_02_03.py
```

Display PNG file. (21)

```
% feh -dF ai202209_s11_02_03.png
```

A local minimum is at $P \sim 3.17$ hr.

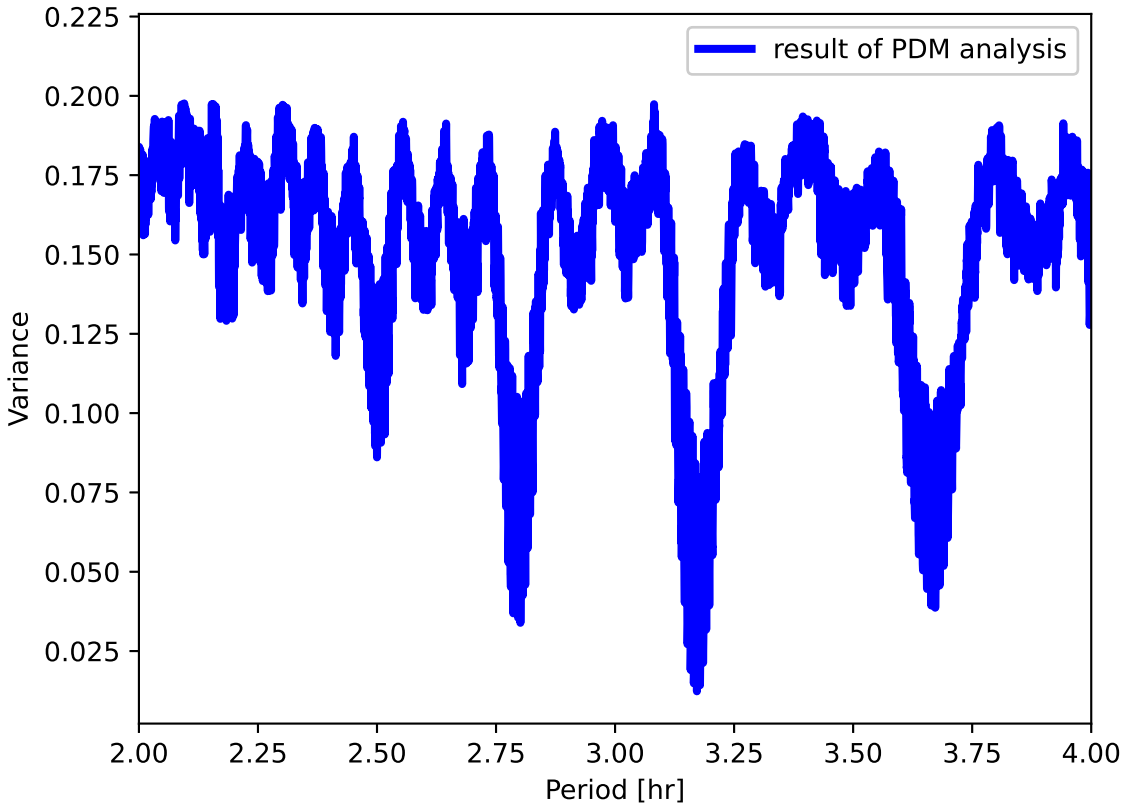

Figure 21: The result of PDM analysis for (20000) Varuna between $P = 2$ hr and $P = 4$ hr.

Try following practice.

Practice 11-20

Change the colour of the line and make the plot again.

Jewitt and Sheppard (2002) shows the frequency (cycle per day) on the horizontal axis. Make a similar plot. Here is an example Python script.

Python Code 21: ai202209_s11_02_04.py

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

#
```

```
# Time-stamp: <2022/11/27 23:41:53 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_02_01.data'

# output file name
file_output = 'ai202209_s11_02_04.png'

# range of x on the plot
x_min = 3.0
x_max = 10.0

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Frequency [cycle/day]')
ax.set_ylabel ('Variance')

# range
ax.set_xlim (x_min, x_max)

# plotting data
ax.plot (24.0 / data_per, data_var, 'b-', label='result of PDM analysis')
ax.legend ()

# saving the plot into a file
```

```
fig.savefig (file_output, dpi=225)
```

Execute above script to visualise the result of PDM analysis. Use cycle per day for X-axis.

```
% chmod a+x ai202209_s11_02_04.py
% ./ai202209_s11_02_04.py
```

Display PNG file. (22)

```
% feh -dF ai202209_s11_02_04.png
```

A local minimum is at $\sim$ 7.6 cycle per day.

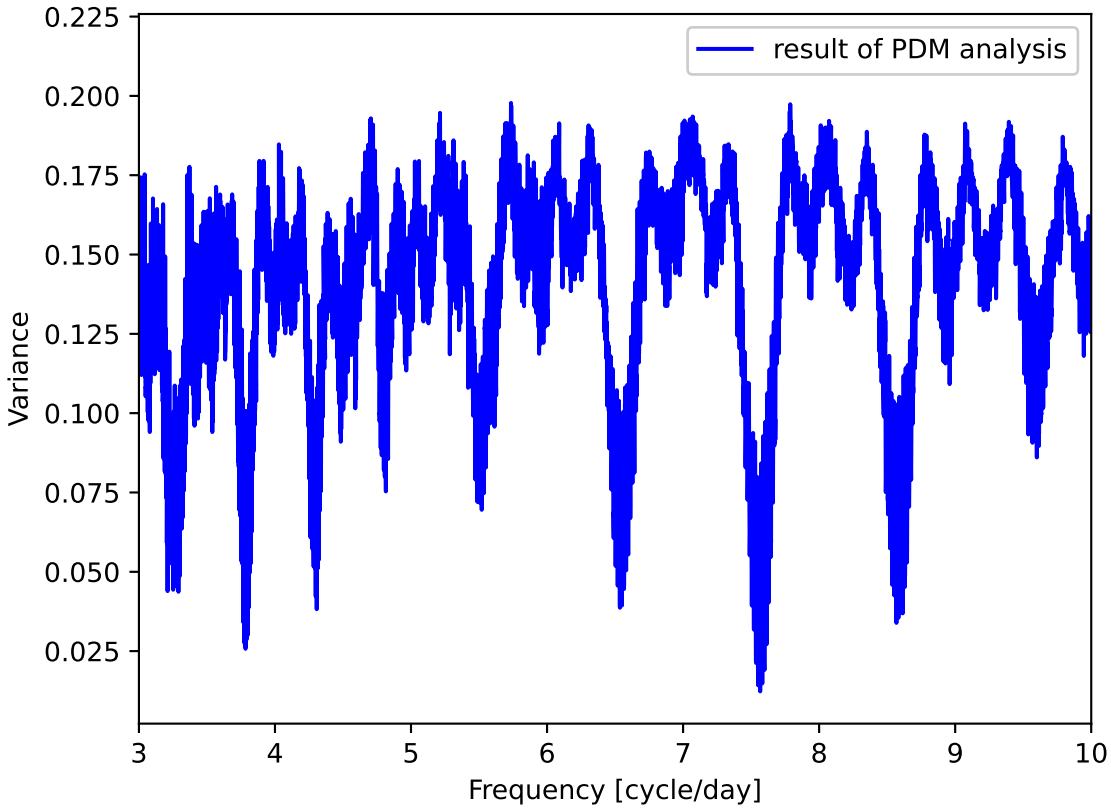

Figure 22: The result of PDM analysis for (20000) Varuna. The minimum can be found at $\sim$ 7.6 cycle per day.

Try following practice.

Practice 11-21

Change the line width and make the plot again.

4.5 Finding the best fit period

We enlarge the region around the local minimum at $P \sim 3.2$ hr.

Python Code 22: ai202209_s11_02_05.py

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

#
# Time-stamp: <2022/11/27 23:45:19 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_02_01.data'

# output file name
file_output = 'ai202209_s11_02_05.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (3.10, 3.25)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()
```

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

Execute above script to visualise the result of PDM analysis at $P \sim 3.2$ hr.

```
% chmod a+x ai202209_s11_02_05.py
% ./ai202209_s11_02_05.py
```

Display PNG file. (23)

```
% feh -dF ai202209_s11_02_05.png
```

A local minimum is at $P \sim 3.17$ hr.

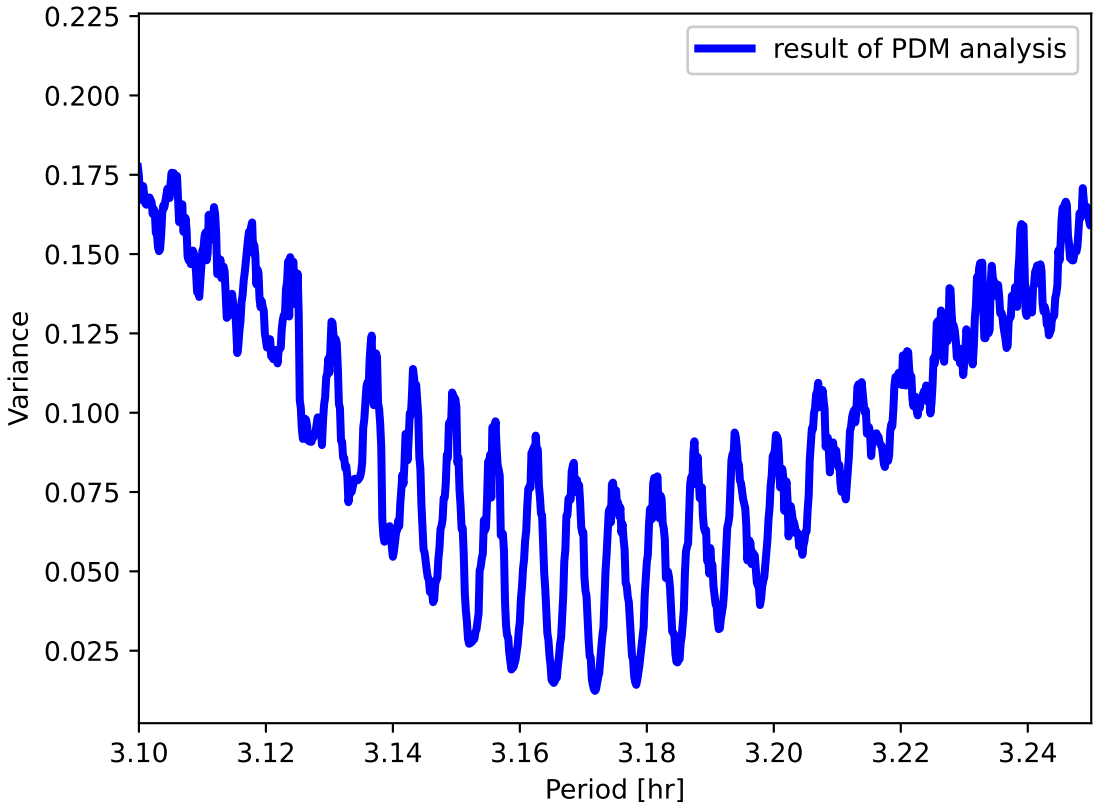

Figure 23: The result of PDM analysis for (20000) Varuna. The minimum can be found at $P \sim 3.17$ hr.

We further enlarge the region around the minimum.

Python Code 23: ai202209_s11_02_06.py

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

#
# Time-stamp: <2022/11/27 23:52:03 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_02_01.data'

# output file name
file_output = 'ai202209_s11_02_06.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (3.15, 3.19)
ax.set_ylim (0.00, 0.12)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis at $P \sim 3.17$ hr.

```
% chmod a+x ai202209_s11_02_06.py
% ./ai202209_s11_02_06.py
```

Display PNG file. (24)

```
% feh -dF ai202209_s11_02_06.png
```

A local minimum is at $P \sim 3.172$ hr.

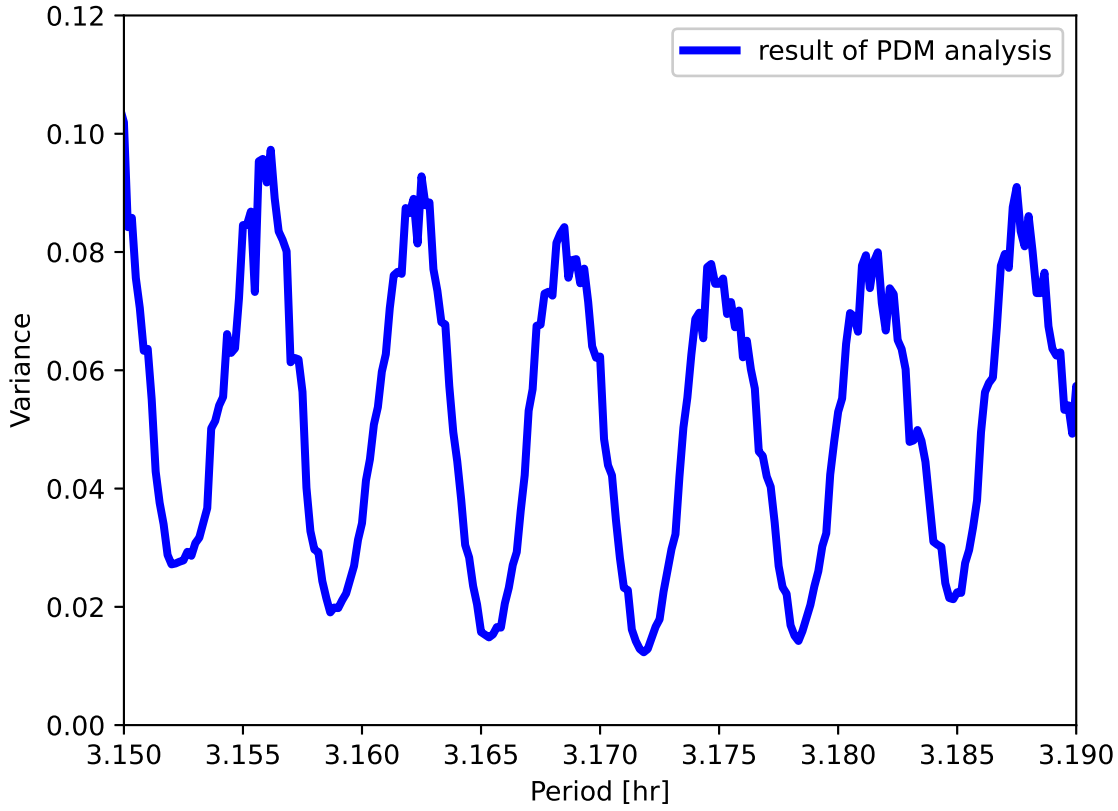

Figure 24: The result of PDM analysis for (20000) Varuna. The minimum can be found at $P \sim 3.172$ hr.

Make a Python script to find the best fit period using least-squares method. Here is an example.

Python Code 24: ai202209_s11_02_07.py

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

#
# Time-stamp: <2022/11/28 03:16:10 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
```

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

# input data file
file_input = 'ai202209_s11_02_01.data'

# output figure file
file_output = 'ai202209_s11_02_07.png'

# range of data for fitting
x_min = 3.1700
x_max = 3.1739

# empty numpy array for storing data
data_all_per = numpy.array ([])
data_all_var = numpy.array ([])
data_fit_per = numpy.array ([])
data_fit_var = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_all_per = numpy.append (data_all_per, per_hr)
        data_all_var = numpy.append (data_all_var, var)
        if ( (per_hr >= x_min) and (per_hr <= x_max) ):
            data_fit_per = numpy.append (data_fit_per, per_hr)
            data_fit_var = numpy.append (data_fit_var, var)

# initial values of coefficients of fitted function
a = 1.0
b = 1.0
c = 1.0

# function to be used for least-squares fitting
def func (x, a, b, c):
    y = a * (x - b)**2 + c
    return y

# least-squares fitting using scipy.optimize.curve_fit
popt, pcov = scipy.optimize.curve_fit (func, data_fit_per, data_fit_var)

# fitted coefficients
print ("popt:")
print (popt)

# covariance matrix
```

```
print ("pcov:")
print (pcov)

# fitted a and b
a_fitted = popt[0]
b_fitted = popt[1]
c_fitted = popt[2]

# degree of freedom
dof = len (data_fit_per) - 3
print (f"dof = {dof}")

# residual
residual = data_fit_var - func (data_fit_per, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f"reduced chi^2 = {reduced_chi2}")

# errors of a and b
a_err = numpy.sqrt (pcov[0][0])
b_err = numpy.sqrt (pcov[1][1])
c_err = numpy.sqrt (pcov[2][2])
print (f"a = {a_fitted:11.5f} +/- {a_err:9.5f} ({a_err/a_fitted*100:6.2f} %)")
print (f"b = {b_fitted:11.5f} +/- {b_err:9.5f} ({b_err/b_fitted*100:6.2f} %)")
print (f"c = {c_fitted:11.5f} +/- {c_err:9.5f} ({c_err/c_fitted*100:6.2f} %)")

# fitted line
fitted_x = numpy.linspace (x_min, x_max, 10**3)
fitted_y = func (fitted_x, a_fitted, b_fitted, c_fitted)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# range of plot
ax.set_xlim (x_min - 0.005, x_max + 0.005)
ax.set_ylim (0, func (x_max, a_fitted, b_fitted, c_fitted) * 1.5 )

# plotting a figure
ax.plot (data_all_per, data_all_var, \
         linestyle='-', linewidth=5, color='blue', \
         label='result of PDM analysis')
ax.plot (fitted_x, fitted_y, \
         linestyle=':', linewidth=3, color='red', \
         label='fitted curve by curve_fit')
ax.legend ()

# saving the figure to a file
fig.savefig (file_output, dpi=225)
```

Execute above script to find the best fit period.

```
% chmod a+x ai202209_s11_02_07.py
% ./ai202209_s11_02_07.py
popt:
```

```
[1.32892810e+04 3.17188575e+00 1.33210644e-02]
pcov:
[[ 9.53311847e+04  1.77848597e-04 -1.26829052e-01]
 [ 1.77848597e-04  1.43278388e-10 -1.42506499e-10]
 [-1.26829052e-01 -1.42506499e-10  3.03202562e-07]]
dof = 21
reduced chi^2 = 3.2257595883229347e-06
a = 13289.28104 +/- 308.75749 (  2.32 %)
b =     3.17189 +/-   0.00001 (  0.00 %)
c =     0.01332 +/-   0.00055 (  4.13 %)
```

Display PNG file. (25)

```
% feh -dF ai202209_s11_02_07.png
```

The best fit period is found to be $P = 3.17189 \pm 0.00001$ hr.

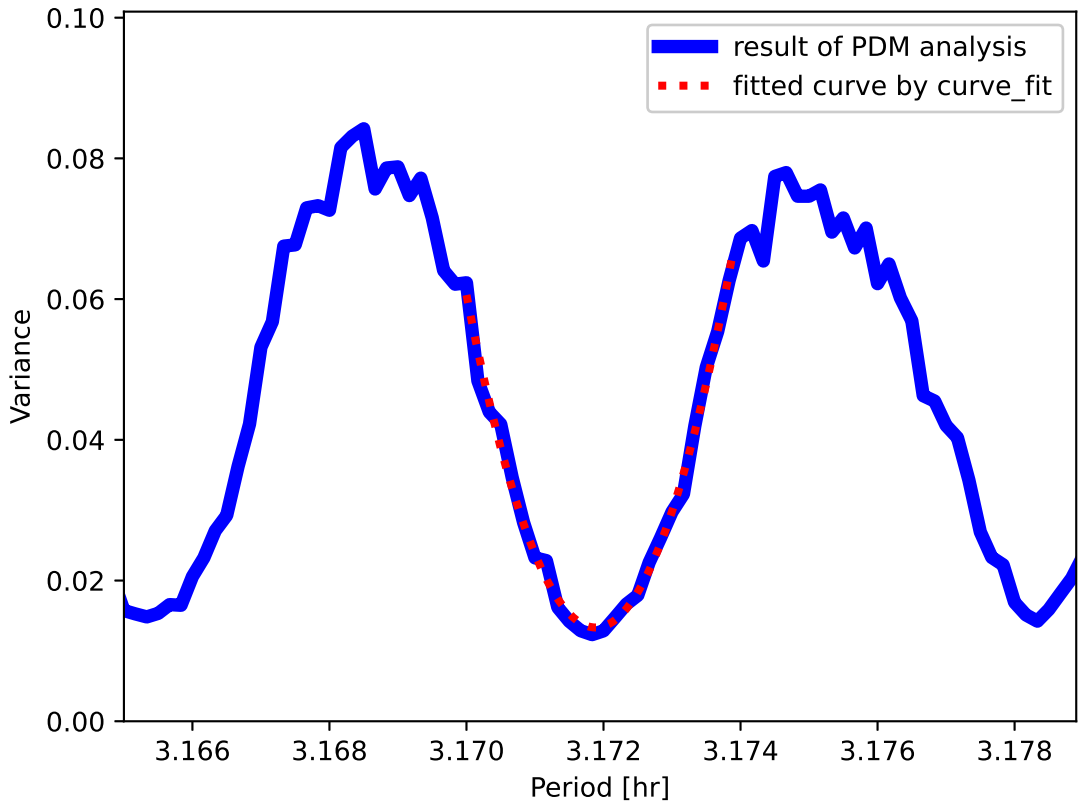

Figure 25: The best fit period is found to be $P = 3.17189 \pm 0.00001$ hr.

Try following practice.

Practice 11-22

Change the initial values of fitting coefficients and carry out least-squares method again.

4.6 Folding the time-series data using the best fit period

In general, an asteroid brightness variation is caused by a rotation of irregularly shaped body. A double of the best fit period is the rotation period of the asteroid. Therefore, the best fit rotation period of (20000) Varuna is $P = 6.34378 \pm 0.00002$ hr. We fold the time-series data using the best fit period of $P = 6.34378 \pm 0.00002$ hr.

Python Code 25: ai202209_s11_02_08.py

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

#
# Time-stamp: <2022/11/28 00:14:01 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = '201498.tb2.txt'

# output file name
file_output = 'ai202209_s11_02_08.png'

# best fit period (day)
p_best = 6.34378 / 24.0

# empty numpy arrays for storing data
data_jd      = numpy.array ([])
data_mag_app = numpy.array ([])
data_mag_abs = numpy.array ([])
data_phase   = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading data line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # skipping line if the line does not start with digits
        if not (line[0].isdigit):
            continue
        # skipping line if it is empty
        if (line.strip () == ''):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (frame_id_str, month_str, day_str, jd_str, mag_app_str, mag_abs_str) \
            = line.split ()
        # conversion from string into float
        frame_id = float (frame_id_str)
        day = float (day_str)
        jd_str = jd_str.replace (',', '')
        jd = float (jd_str)
        mag_app = float (mag_app_str)
        mag_abs = float (mag_abs_str)
```

```
        # appending the data at the end of numpy arrays
        data_jd      = numpy.append (data_jd, jd)
        data_mag_app = numpy.append (data_mag_app, mag_app)
        data_mag_abs = numpy.append (data_mag_abs, mag_abs)

        phase = (jd - data_jd[0]) / p_best
        phase -= int (phase)
        data_phase   = numpy.append (data_phase, phase)

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

# labels
ax.set_xlabel ('Phase')
ax.set_ylabel ('Absolute Magnitude [mag]')

# axes
ax.invert_yaxis ()

# plotting data
ax.plot (data_phase, data_mag_abs, \
         linestyle='None', marker='o', markersize=3, color='red', \
         label='folded lightcurve')
ax.plot (data_phase + 1, data_mag_abs, \
         linestyle='None', marker='o', markersize=3, color='red')
ax.legend ()

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

Execute above script to construct a lightcurve by folding the time-series data using the best fit period.

```
% chmod a+x ai202209_s11_02_08.py
% ./ai202209_s11_02_08.py
```

Display PNG file. (26)

```
% feh -dF ai202209_s11_02_08.png
```

Compare the plot with Fig. 4 of Jewitt and Sheppard (2002).
Try following practice.

Practice 11-23

Use the best fit period of $P = 3.17189$ hr, and construct a single-peaked lightcurve.

5 Variable star data

A set of variable star data can be found at following website.

- `http://faculty.washington.edu/ivezic/linear/PaperIII/PLV.html`

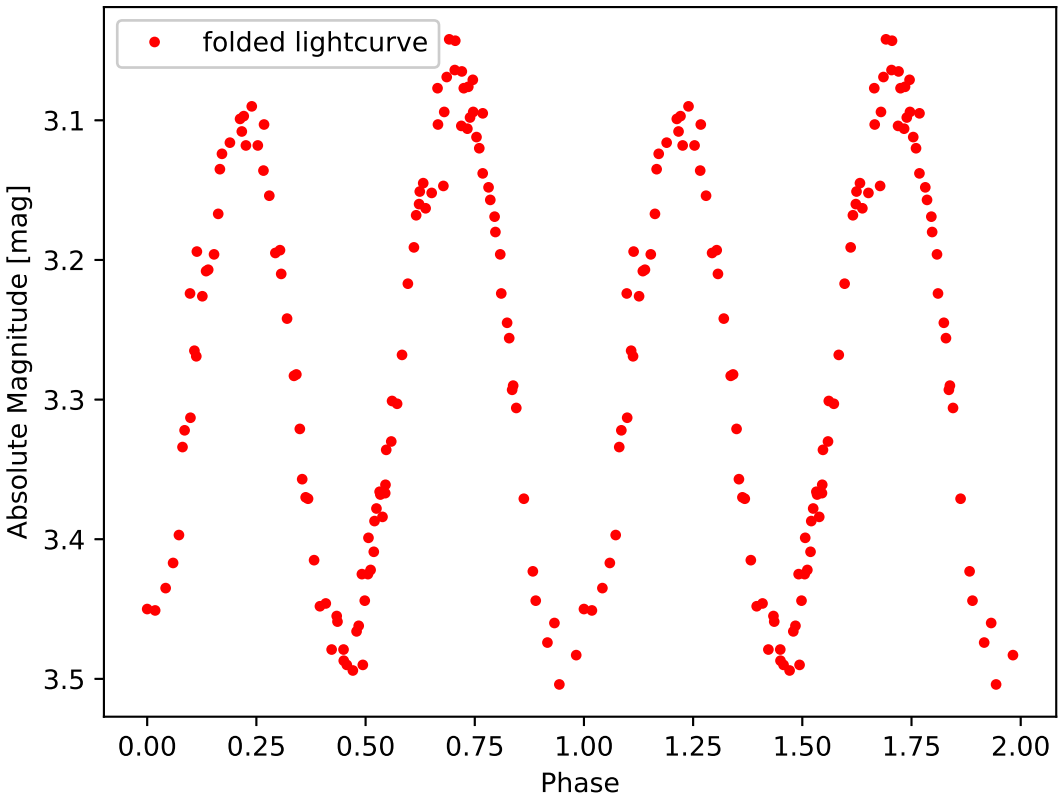

Figure 26: The folded lightcurve of (20000) Varuna constructed from time-series data and the best fit period of $P = 6.34378 \pm 0.00002$ hr.

5.1 Downloading variable star data

Make a Python script to download LINEAR variables data. Here is an example.

Python Code 26: ai202209_s11_03_00.py

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

#
# Time-stamp: <2022/11/28 00:19:53 (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 = 'http://faculty.washington.edu/ivezic/linear/PaperIII/allDAT.tar.gz'

# output file name
file_output = 'linear.tar.gz'

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

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

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

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

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

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

Execute above script to download the LINEAR variables data.

```
% chmod a+x ai202209_s11_03_00.py
% ./ai202209_s11_03_00.py
Now, fetching http://faculty.washington.edu/ivezic/linear/PaperIII/allDAT.tar.gz
...
Finished fetching http://faculty.washington.edu/ivezic/linear/PaperIII/allDAT.ta
r.gz!
Now, writing the data into file "linear.tar.gz"...
Finished writing the data into file "linear.tar.gz"!
% ls -lF *.tar.gz
-rw-r--r-- 1 daisuke taiwan 17403165 Nov 28 00:20 linear.tar.gz
```

5.2 Extracting the data

Extract the downloaded data.

```
% mkdir linear
% cd linear
% tar xzvf ../linear.tar.gz
./._10003298.dat
10003298.dat
./._10004892.dat
10004892.dat
./._10013411.dat
10013411.dat
./._10021274.dat
10021274.dat
./._10022663.dat
10022663.dat

.....

./._9984569.dat
9984569.dat
./._9987252.dat
9987252.dat
./._999498.dat
999498.dat
./._999528.dat
999528.dat
./._9996084.dat
9996084.dat
% cd ..
```

5.3 Visualising the photometry of the object 4672469

Find the data of the object 4672469.

```
% ls -lF linear/4672469.dat
-rw-r--r-- 1 daisuke taiwan 32636 Oct  8  2010 linear/4672469.dat
% head linear/4672469.dat
        52610.388566    16.137     0.068
        52610.388699    16.099     0.057
        52610.403695    15.920     0.044
        52610.403829    15.979     0.043
        52610.419000    15.769     0.031
        52610.434167    15.856     0.063
        52610.449282    15.887     0.085
        52614.378450    15.907     0.039
        52614.393550    15.962     0.041
        52614.423889    16.056     0.046
```

Visualise the time-series photometry of the object 4672469.

Python Code 27: ai202209_s11_03_01.py

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

#
# Time-stamp: <2022/11/28 00:36:25 (CST) daisuke>
```

```
#

# importing numpy module
import numpy

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

# data file name
file_input = 'linear/4672469.dat'

# output file name
file_output = 'ai202209_s11_03_01.png'

# empty numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])
data_err = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
        data_err = numpy.append (data_err, err)

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

# labels
ax.set_xlabel ('MJD [day]')
ax.set_ylabel ('Apparent Magnitude [mag]')

# axes
ax.invert_yaxis ()

# plotting data
ax.errorbar (data_mjd, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='green', \
             ecolor='black', capsize=5, \
             label='LINEAR time-series data for object 4672469')
ax.legend ()
```

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

Execute above script to visualise the time-series photometry of the object 4672469.

```
% chmod a+x ai202209_s11_03_01.py
% ./ai202209_s11_03_01.py
```

Display PNG file. (27)

```
% feh -dF ai202209_s11_03_01.png
```

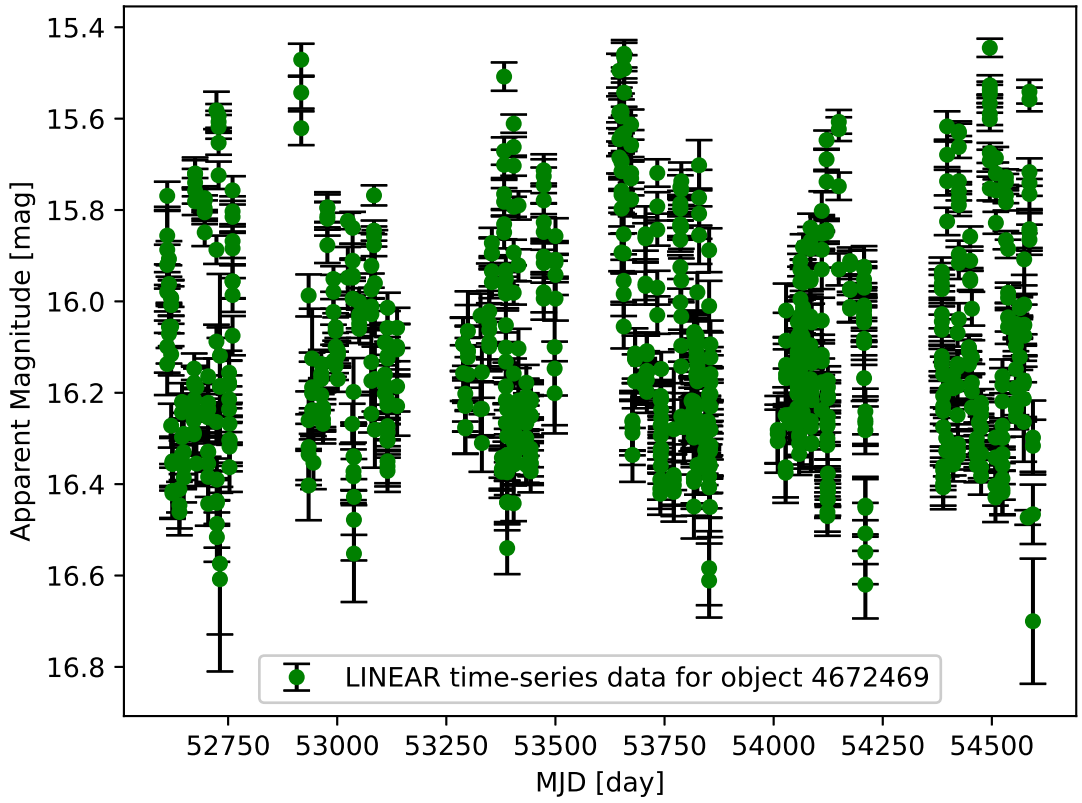

Figure 27: The time-series photometry of the object 4672469.

Try following practice.

Practice 11-24

Change marker shape, marker size, and marker colour, and make the plot again.

5.4 Period search using PDM

Make a Python script to carry out a period search using PDM for the object 4672469. Here is an example.

Python Code 28: ai202209_s11_03_02.py

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

#
# Time-stamp: <2022/11/28 00:41:54 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'linear/4672469.dat'

# output file name
file_output = 'ai202209_s11_03_02.data'

# shortest trial period
period_min_min = 10.0
period_min_day = period_min_min / (60.0 * 24.0)

# longest trial period
period_max_min = 1500.0
period_max_day = period_max_min / (60.0 * 24.0)

# step size of trial period
step_min = 0.1
step_day = step_min / (60.0 * 24.0)

# number of bins
n_bins = 40

# numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])
data_err = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
        data_err = numpy.append (data_err, err)
```

```
# opening file for writing
with open (file_output, 'w') as fh_out:
    # writing header to output file
    header  = f"#\n"
    header += f"# parameters for PDM analysis\n"
    header += f"#\n"
    header += f"# input file = {file_input}\n"
    header += f"# output file = {file_output}\n"
    header += f"# shortest trial period = {period_min_min} min\n"
    header += f"# longest trial period = {period_max_min} min\n"
    header += f"# step size of trial period = {step_min} min\n"
    header += f"# number of bins = {n_bins}\n"
    header += f"#\n"
    header += f"# results of PDM analysis\n"
    header += f"#\n"
    header += f"# trial period (day), trial period (hr), trial period (min), "
    header += f"total variance\n"
    header += f"#\n"
    fh_out.write (header)

    # initial value of trial period
    period_day = period_min_day

    # period search
    while (period_day < period_max_day):
        # calculation of phase with assumed period
        data_phase = numpy.array ([])
        for i in range ( len (data_mjd) ):
            phase = (data_mjd[i] - data_mjd[0]) / period_day
            phase -= int (phase)
            data_phase = numpy.append (data_phase, phase)

        # initialization of parameter
        total_variance = 0.0

        # calculation of variance
        for i in range (n_bins):
            # range of bin
            bin_min = i / n_bins
            bin_max = (i + 1) / n_bins

            # finding data within the bin
            data_bin = numpy.array ([])
            for j in range ( len (data_phase) ):
                if ( (data_phase[j] >= bin_min) and (data_phase[j] < bin_max) ):
                    data_bin = numpy.append (data_bin, data_mag[j])

            # if no data in the bin, then we skip.
            if (len (data_bin) == 0):
                continue

            # variance
            variance_in_bin = numpy.var (data_bin)
            # sum of variance
            total_variance += variance_in_bin

        # writing data to file
        output = f"{period_day:12.10f} {period_day * 24.0:12.8f} " \
```

```
            + f"{period_day * 24.0 * 60.0:12.6f} {total_variance:10.6f}\n"
        fh_out.write (output)

        # next trial period
        period_day += step_day
```

Execute above script to carry out period search for the object 4672469 using PDM.

```
% chmod a+x ai202209_s11_03_02.py
% ./ai202209_s11_03_02.py
% ls -lF *.data
-rw-r--r-- 1 daisuke taiwan   11492 Nov 27 17:23 ai202209_s11_00_00.data
-rw-r--r-- 1 daisuke taiwan  145351 Nov 27 18:25 ai202209_s11_00_04.data
-rw-r--r-- 1 daisuke taiwan   25658 Nov 27 20:09 ai202209_s11_01_00.data
-rw-r--r-- 1 daisuke taiwan  495352 Nov 27 20:34 ai202209_s11_01_02.data
-rw-r--r-- 1 daisuke taiwan 4950344 Nov 27 22:58 ai202209_s11_02_01.data
-rw-r--r-- 1 daisuke taiwan  745347 Nov 28 00:46 ai202209_s11_03_02.data
% head -20 ai202209_s11_03_02.data
#
# parameters for PDM analysis
#
# input file = linear/4672469.dat
# output file = ai202209_s11_03_02.data
# shortest trial period = 10.0 min
# longest trial period = 1500.0 min
# step size of trial period = 0.1 min
# number of bins = 40
#
# results of PDM analysis
#
# trial period (day), trial period (hr), trial period (min), total variance
#
0.0069444444   0.16666667    10.000000   2.081896
0.0070138889   0.16833333    10.100000   2.060984
0.0070833333   0.17000000    10.200000   2.094747
0.0071527778   0.17166667    10.300000   2.073052
0.0072222222   0.17333333    10.400000   2.027556
0.0072916667   0.17500000    10.500000   2.088249
```

Try following practice.

Practice 11-25

Change some parameters and carry out PDM analysis again.

Visualise the result of PDM analysis.

Python Code 29: ai202209_s11_03_03.py

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

#
# Time-stamp: <2022/11/28 00:55:37 (CST) daisuke>
#

# importing numpy module
import numpy

# importing matplotlib module
import matplotlib.figure
```

```
import matplotlib.backends.backend_agg

# input file name
file_input = 'ai202209_s11_03_02.data'

# output file name
file_output = 'ai202209_s11_03_03.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (0.0, 20.0)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis for the object 4672469.

```
% chmod a+x ai202209_s11_03_03.py
% ./ai202209_s11_03_03.py
```

Display PNG file. (28)

```
% feh -dF ai202209_s11_03_03.png
```

A local minimum is found at $P \sim 13$ hr.

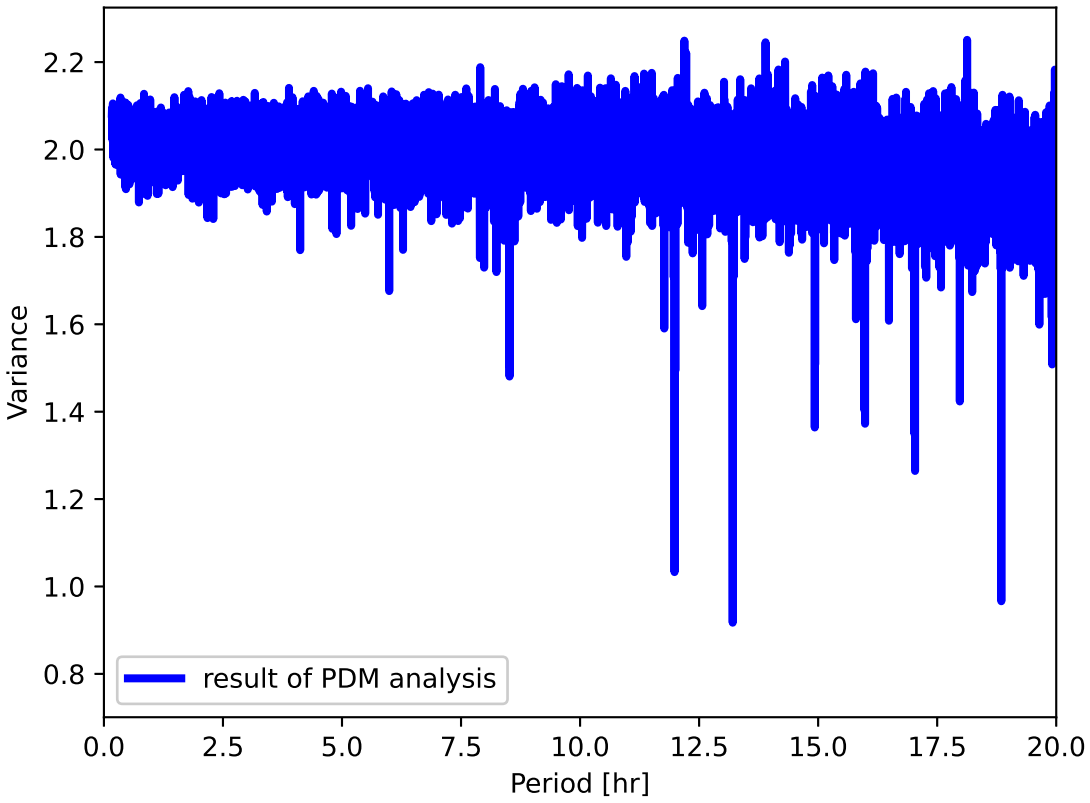

Figure 28: The result of PDM analysis for the object 4672469.

Enlarge the region around $P \sim 13$ hr.

Python Code 30: ai202209_s11_03_04.py

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

#
# Time-stamp: <2022/11/28 00:59:17 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_03_02.data'

# output file name
file_output = 'ai202209_s11_03_04.png'
```

```
# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (12.9, 13.5)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis for the object 4672469 around $P \sim 13$ hr.

```
% chmod a+x ai202209_s11_03_04.py
% ./ai202209_s11_03_04.py
```

Display PNG file. (29)

```
% feh -dF ai202209_s11_03_04.png
```

A local minimum is found at $P \sim 13.2$ hr.
Carry out PDM analysis again.

Python Code 31: ai202209_s11_03_05.py

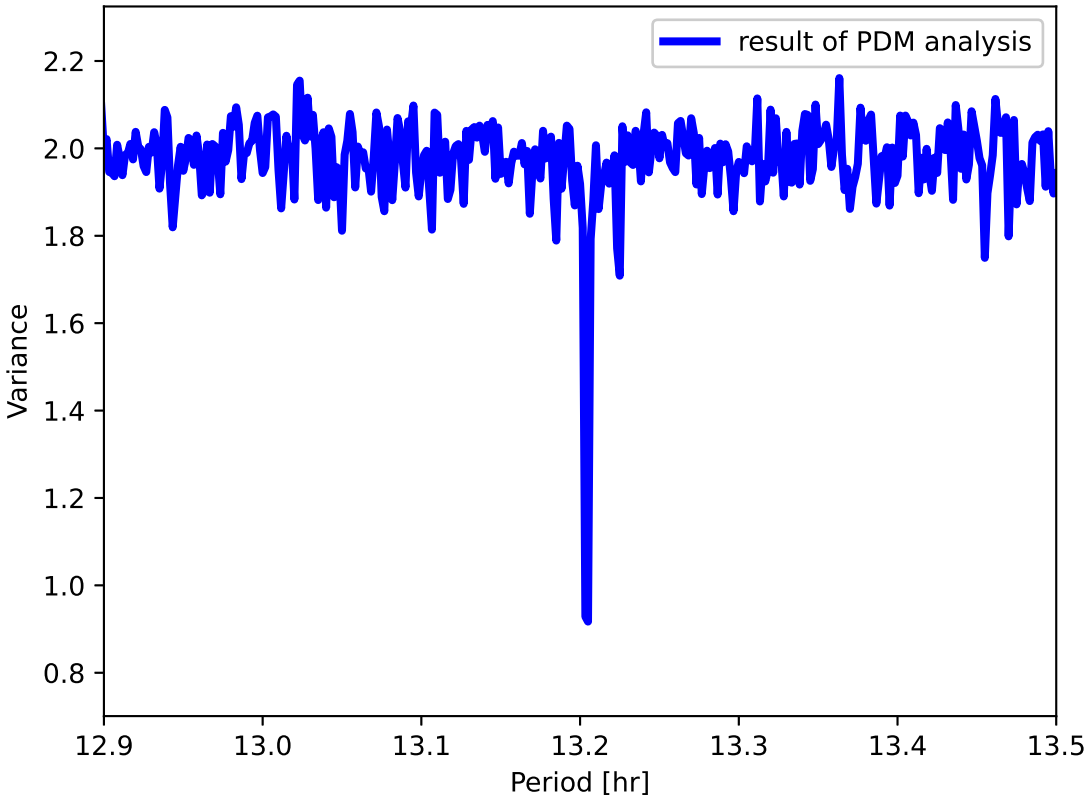

Figure 29: The result of PDM analysis for the object 4672469 around $P = 13$ hr.

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

#
# Time-stamp: <2022/11/28 01:45:32 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'linear/4672469.dat'

# output file name
file_output = 'ai202209_s11_03_05.data'

# shortest trial period
period_min_min = 750.0
period_min_day = period_min_min / (60.0 * 24.0)

# longest trial period
period_max_min = 850.0
period_max_day = period_max_min / (60.0 * 24.0)

# step size of trial period
step_min = 0.002
step_day = step_min / (60.0 * 24.0)

# number of bins
n_bins = 40

# numpy arrays for storing data
data_mjd = numpy.array ([])
data_mag = numpy.array ([])
data_err = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        # appending the data at the end of numpy arrays
        data_mjd = numpy.append (data_mjd, mjd)
        data_mag = numpy.append (data_mag, mag)
        data_err = numpy.append (data_err, err)
```

```
# opening file for writing
with open (file_output, 'w') as fh_out:
    # writing header to output file
    header  = f"#\n"
    header += f"# parameters for PDM analysis\n"
    header += f"#\n"
    header += f"# input file = {file_input}\n"
    header += f"# output file = {file_output}\n"
    header += f"# shortest trial period = {period_min_min} min\n"
    header += f"# longest trial period = {period_max_min} min\n"
    header += f"# step size of trial period = {step_min} min\n"
    header += f"# number of bins = {n_bins}\n"
    header += f"#\n"
    header += f"# results of PDM analysis\n"
    header += f"#\n"
    header += f"# trial period (day), trial period (hr), trial period (min), "
    header += f"total variance\n"
    header += f"#\n"
    fh_out.write (header)

    # initial value of trial period
    period_day = period_min_day

    # period search
    while (period_day < period_max_day):
        # calculation of phase with assumed period
        data_phase = numpy.array ([])
        for i in range ( len (data_mjd) ):
            phase = (data_mjd[i] - data_mjd[0]) / period_day
            phase -= int (phase)
            data_phase = numpy.append (data_phase, phase)

        # initialization of parameter
        total_variance = 0.0

        # calculation of variance
        for i in range (n_bins):
            # range of bin
            bin_min = i / n_bins
            bin_max = (i + 1) / n_bins

            # finding data within the bin
            data_bin = numpy.array ([])
            for j in range ( len (data_phase) ):
                if ( (data_phase[j] >= bin_min) and (data_phase[j] < bin_max) ):
                    data_bin = numpy.append (data_bin, data_mag[j])

            # if no data in the bin, then we skip.
            if (len (data_bin) == 0):
                continue

            # variance
            variance_in_bin = numpy.var (data_bin)
            # sum of variance
            total_variance += variance_in_bin

        # writing data to file
        output = f"{period_day:12.10f} {period_day * 24.0:12.8f} " \
            + f"{period_day * 24.0 * 60.0:12.6f} {total_variance:10.6f}\n"
```

```
        fh_out.write (output)

        # next trial period
        period_day += step_day
```

Execute above script to visualise the result of PDM analysis for the object 4672469 around $P \sim 13.2$ hr.

```
% chmod a+x ai202209_s11_03_05.py
% ./ai202209_s11_03_05.py
% head -20 ai202209_s11_03_05.data
#
# parameters for PDM analysis
#
# input file = linear/4672469.dat
# output file = ai202209_s11_03_05.data
# shortest trial period = 750.0 min
# longest trial period = 850.0 min
# step size of trial period = 0.002 min
# number of bins = 40
#
# results of PDM analysis
#
# trial period (day), trial period (hr), trial period (min), total variance
#
0.5208333333  12.50000000   750.000000   2.053718
0.5208347222  12.50003333   750.002000   2.050146
0.5208361111  12.50006667   750.004000   2.076764
0.5208375000  12.50010000   750.006000   2.073081
0.5208388889  12.50013333   750.008000   2.063236
0.5208402778  12.50016667   750.010000   2.049318
```

Visualise the result of PDM analysis.

Python Code 32: ai202209_s11_03_06.py

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

#
# Time-stamp: <2022/11/28 02:10:23 (CST) daisuke>
#

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s11_03_05.data'

# output file name
file_output = 'ai202209_s11_03_06.png'

# empty numpy arrays for storing data
data_per = numpy.array ([])
data_var = numpy.array ([])

# opening file
```

```
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (per_day_str, per_hr_str, per_min_str, var_str) = line.split ()
        # conversion from string into float
        per_hr = float (per_hr_str)
        var = float (var_str)
        # appending the data at the end of numpy arrays
        data_per = numpy.append (data_per, per_hr)
        data_var = numpy.append (data_var, var)

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

# labels
ax.set_xlabel ('Period [hr]')
ax.set_ylabel ('Variance')

# axes
ax.set_xlim (13.19, 13.22)

# plotting data
ax.plot (data_per, data_var, \
         linestyle='-', linewidth=3, color='blue', \
         label='result of PDM analysis')
ax.legend ()

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

Execute above script to visualise the result of PDM analysis for the object 4672469 around $P \sim 13.20$ hr.

```
% chmod a+x ai202209_s11_03_06.py
% ./ai202209_s11_03_06.py
```

Display PNG file. (30)

```
% feh -dF ai202209_s11_03_06.png
```

We examine the data.

```
% grep -v # ai202209_s11_03_05.data | sort -n +3 | head
0.5501763889  13.20423333   792.254000   0.510906
0.5501777778  13.20426667   792.256000   0.512927
0.5501819444  13.20436667   792.262000   0.523452
0.5501805556  13.20433333   792.260000   0.526314
0.5501736111  13.20416667   792.250000   0.526348
0.5501847222  13.20443333   792.266000   0.529938
0.5501722222  13.20413333   792.248000   0.530848
```

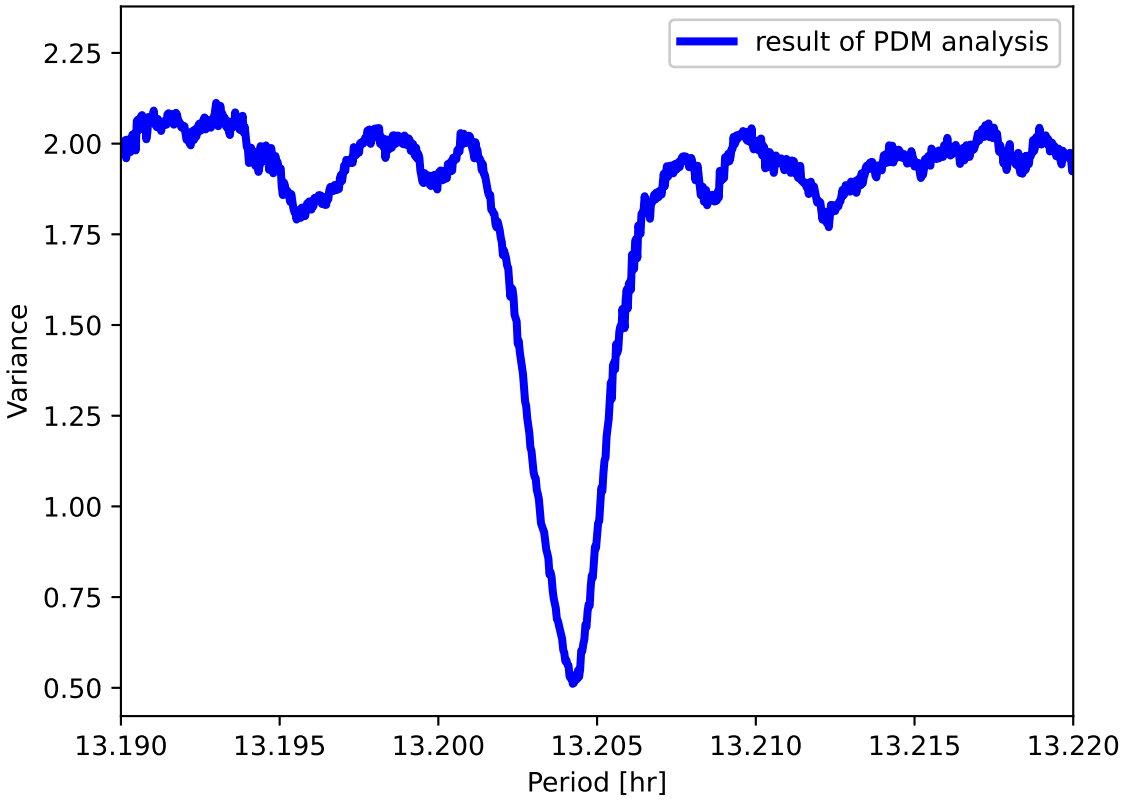

Figure 30: The result of PDM analysis for the object 4672469 around $P = 13.204$ hr.

```
0.5501791667   13.20430000    792.258000    0.531157
0.5501750000   13.20420000    792.252000    0.535127
0.5501833333   13.20440000    792.264000    0.548301
```

We adopt $P = 13.204233$ hr as the best fit period.

5.5 Folded lightcurve

Fold the time-series data using the best fit period of $P = 13.204233$ hr.

Python Code 33: ai202209_s11_03_07.py

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

#
# Time-stamp: <2022/11/28 02:36:52 (CST) daisuke>
#

# importing numpy module
import numpy

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

# data file name
file_input = 'linear/4672469.dat'

# output file name
file_output = 'ai202209_s11_03_07.png'

# best fit period (day)
p_best = 13.204233 / 24.0

# empty numpy arrays for storing data
data_mjd   = numpy.array ([])
data_phase = numpy.array ([])
data_mag   = numpy.array ([])
data_err   = numpy.array ([])

# opening file
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line if the line starts with '#'
        if (line[0] == '#'):
            continue
        # removing line feed at the end of line
        line = line.strip ()
        # splitting data
        (mjd_str, mag_str, err_str) = line.split ()
        # conversion from string into float
        mjd = float (mjd_str)
        mag = float (mag_str)
        err = float (err_str)
        phase = mjd / p_best - int (mjd / p_best)
        # appending the data at the end of numpy arrays
        data_mjd   = numpy.append (data_mjd, mjd)
        data_phase = numpy.append (data_phase, phase)
```

```
        data_mag   = numpy.append (data_mag, mag)
        data_err   = numpy.append (data_err, err)

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

# labels
ax.set_xlabel ('Phase')
ax.set_ylabel ('Apparent Magnitude [mag]')

# axes
ax.set_ylim (15.2, 17.0)
ax.invert_yaxis ()

# plotting data
ax.errorbar (data_phase, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='green', \
             ecolor='black', capsize=5, \
             label='folded lightcurve')
ax.errorbar (data_phase + 1, data_mag, yerr=data_err, \
             linestyle='None', marker='o', markersize=5, color='green', \
             ecolor='black', capsize=5)
ax.legend ()

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

Execute above script to construct folded lightcurve.

```
% chmod a+x ai202209_s11_03_07.py
% ./ai202209_s11_03_07.py
```

Display PNG file. (31)

```
% feh -dF ai202209_s11_03_07.png
```

Try following practice.

Practice 11-26

Change the marker shape, marker size, and marker colour, and make the plot again.

6 For your further reading

Read following document to learn more about PDM.

- "Period determination using phase dispersion minimization"
  - Stellingwerf, R. F., 1978, ApJ, 224, 953.
  - `https://ui.adsabs.harvard.edu/abs/1978ApJ...224..953S/abstract`

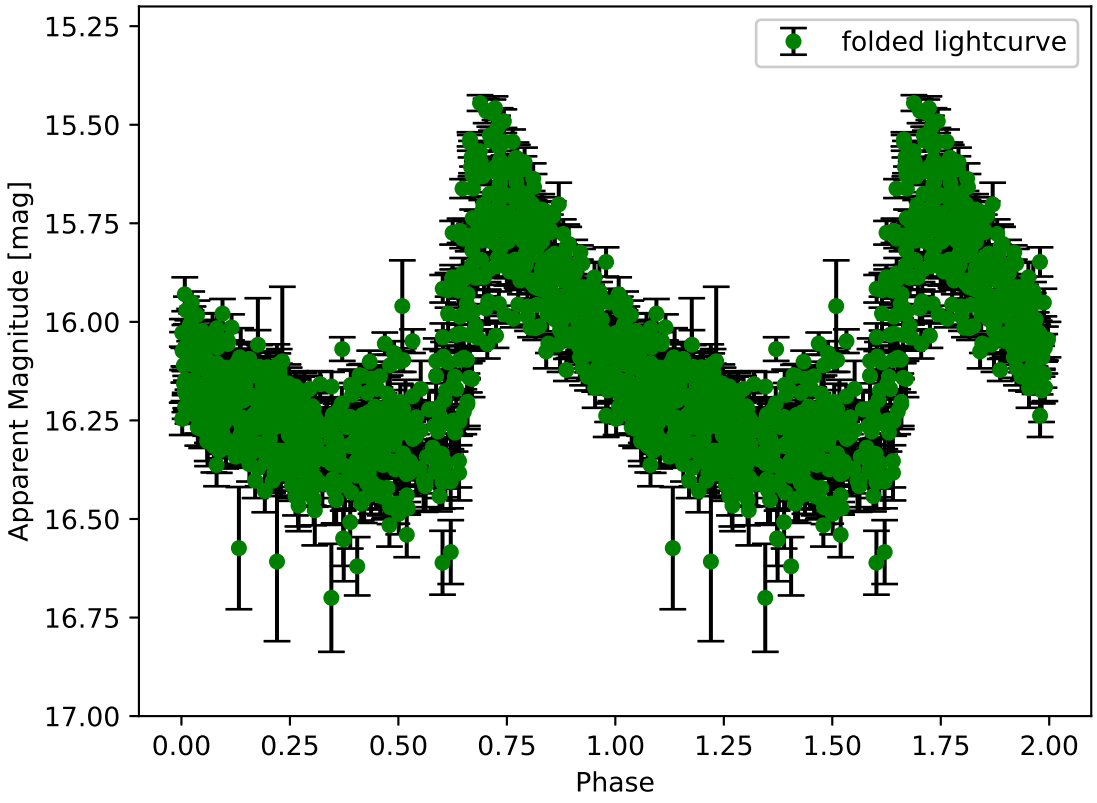

Figure 31: The folded lightcurve for the object 4672469.

7 Assignment

1. Learn about PDM (Phase Dispersion Minimisation). Give a write-up about PDM.
2. Make your own Python script to carry out PDM analysis. Show the source code of your Python script.
3. Visit "KWS" (Kamogata/Kiso/Kyoto Wide-field Survey) (`http://kws.cetus-net.org/~maehara/VSdata.py`)
   (a) Download the data of 3 objects.
   (b) Carry out a period search using PDM for each object.
   (c) What are periods you have found?
   (d) Make folded lightcurves.
   (e) What are classes of objects?
   (f) Show source codes of all the Python script you have written.
4. Visit PTF website. (`https://www.ptf.caltech.edu/page/lcgui`)
   (a) Search for data of 3 objects.
   (b) Download the data.
   (c) Carry out a period search using PDM for each object.
   (d) What are periods you have found?
   (e) Make folded lightcurves.
   (f) What are classes of objects?
   (g) Show source codes of all the Python script you have written.