Astroinformatics 2022
Session 05: Using SciPy

Kinoshita Daisuke

17 October 2022
publicly accessible version

About this file...

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

For this session, we try SciPy.

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) "`s05`". (Fig. 3) Choose the file "`ai202209_s05.ipynb`" (Fig. 4 and 5) and open it (Fig. 6).

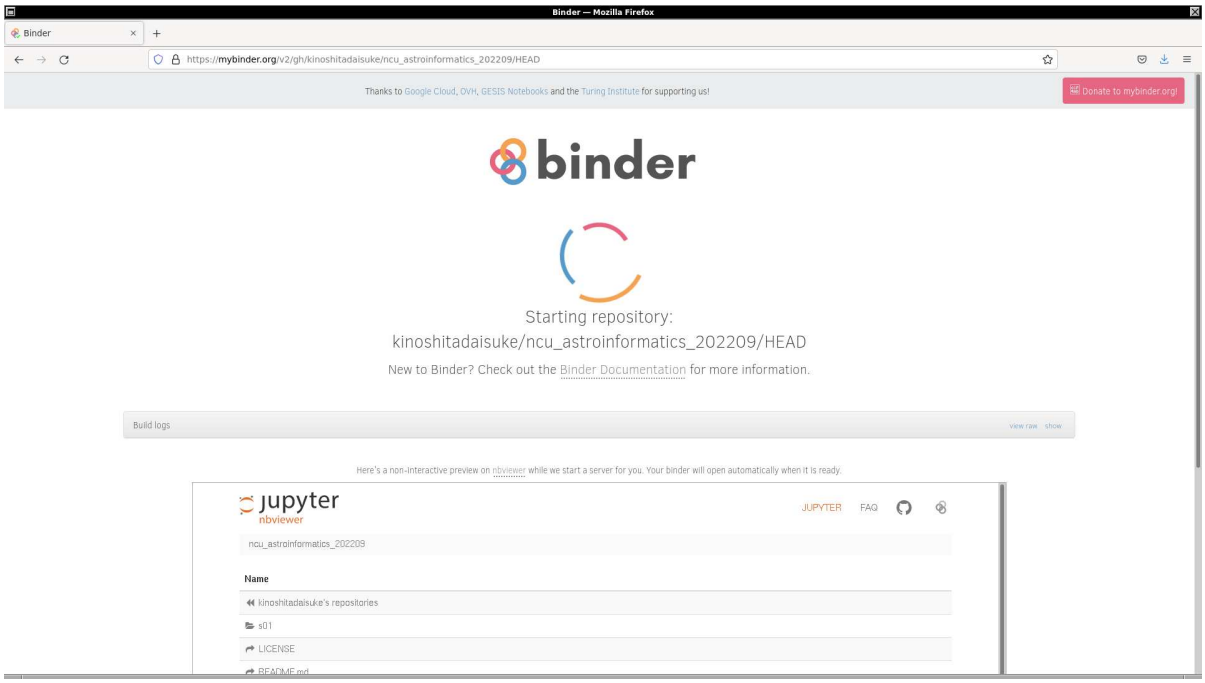

Figure 1: Using Binder to execute sample Python scripts for this session.

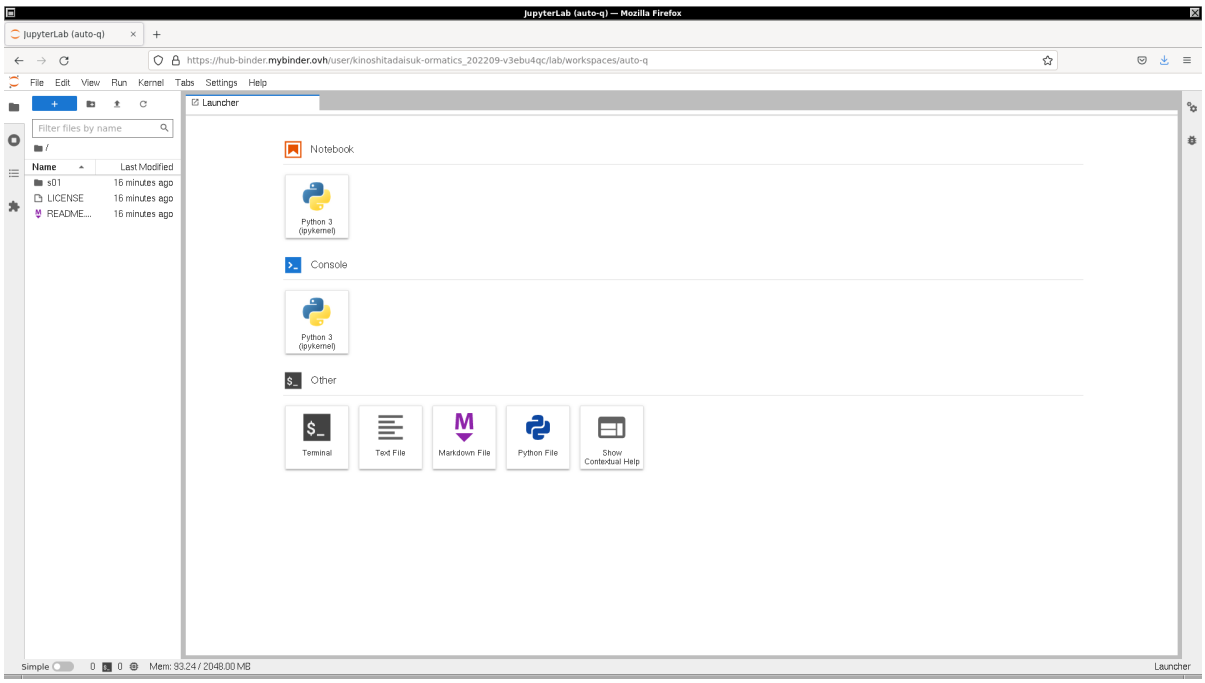

Figure 2: Using Binder to execute sample Python scripts for this session.

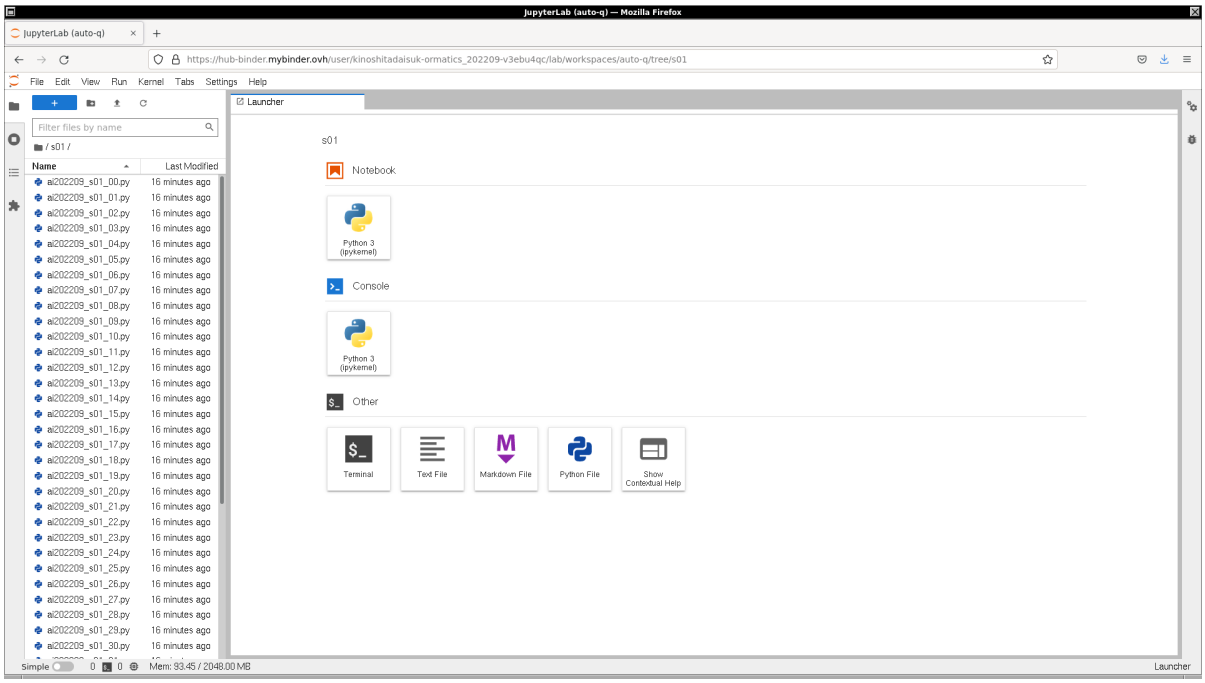

Figure 3: Using Binder to execute sample Python scripts for this session.

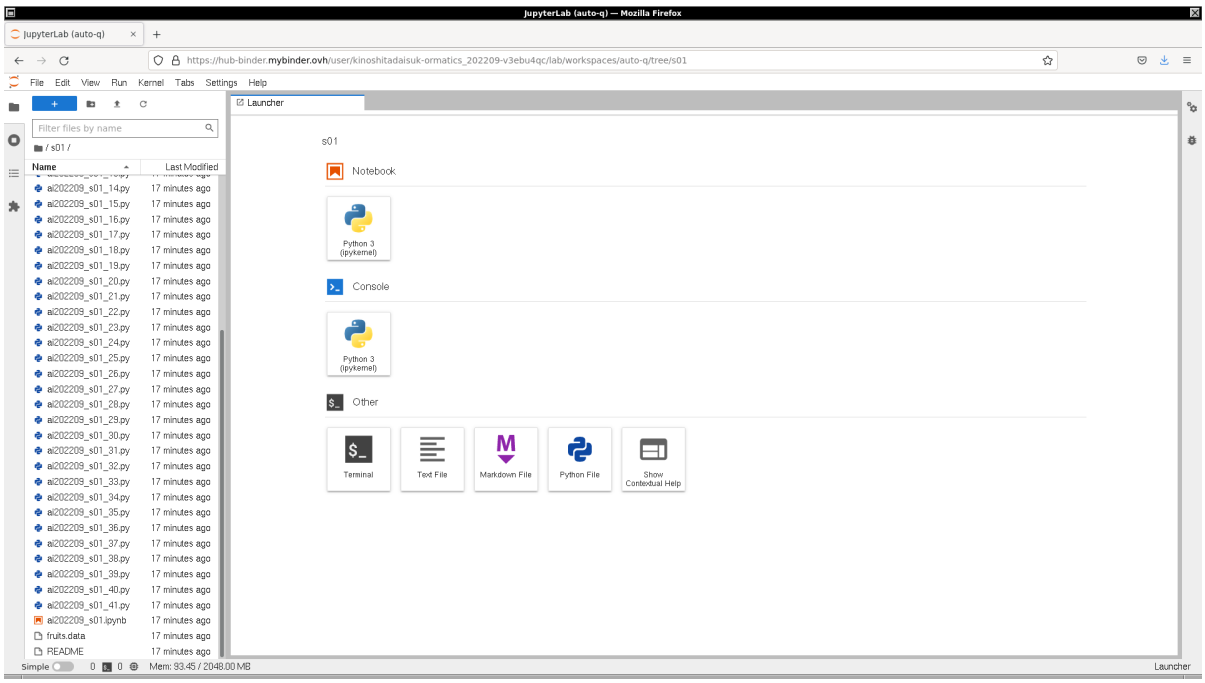

Figure 4: Using Binder to execute sample Python scripts for this session.

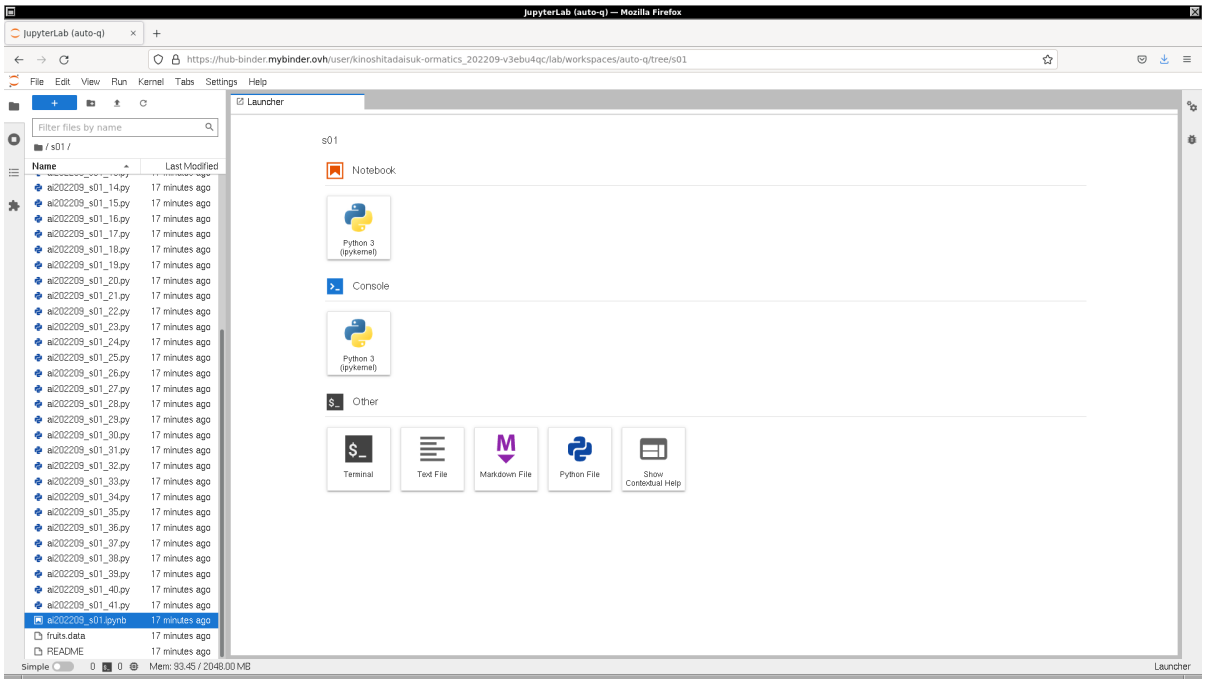

Figure 5: Using Binder to execute sample Python scripts for this session.

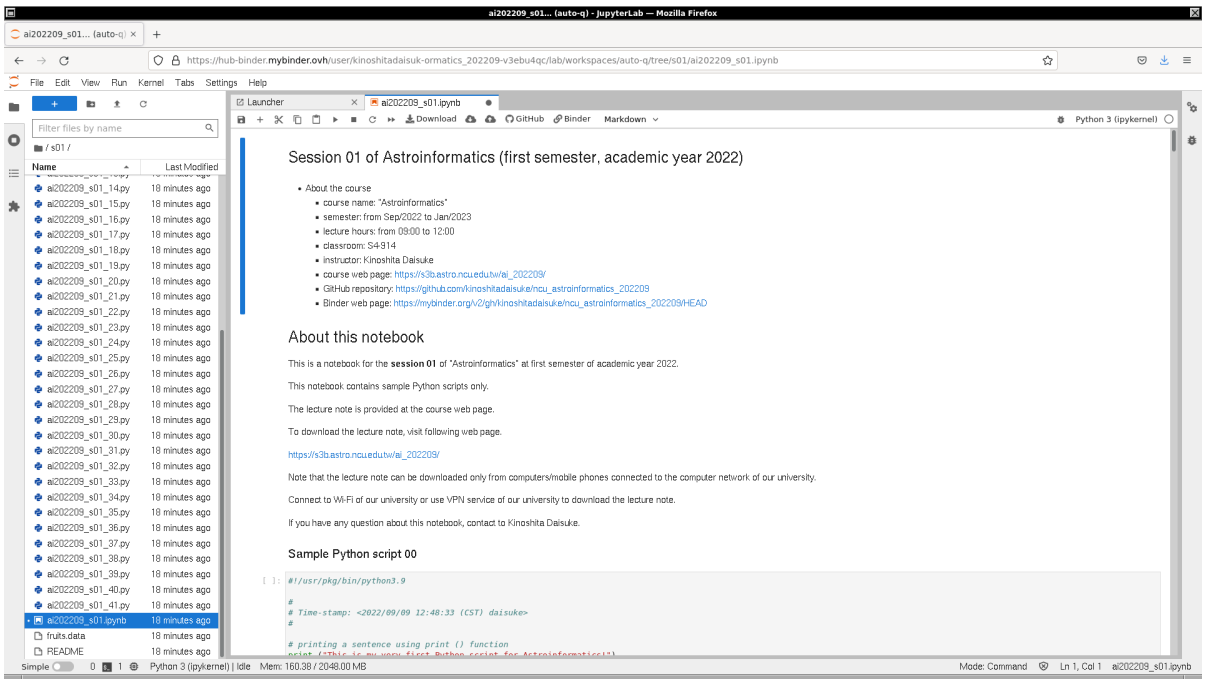

Figure 6: Using Binder to execute sample Python scripts for this session.

2 About SciPy

SciPy is a collection of algorithms for scientific calculations and data analyses. SciPy offers convenient functions for calculations of statistical values, linear algebra, interpolation, numerical integration, signal processing, etc. Visit the official website of SciPy to learn about it.

- SciPy: `https://scipy.org/`
  - SciPy Documentation: `https://docs.scipy.org/doc/scipy/`
  - SciPy User Guide: `https://docs.scipy.org/doc/scipy/tutorial/`
  - SciPy API Reference: `https://docs.scipy.org/doc/scipy/reference/`

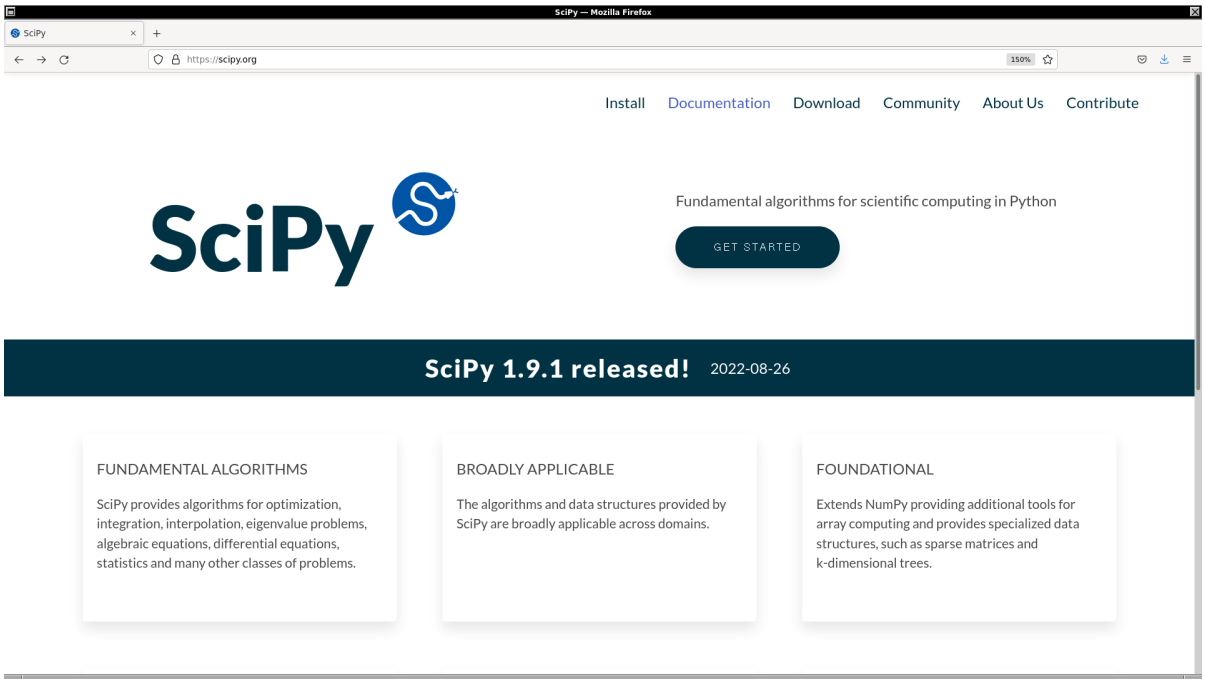

Figure 7: The official website of SciPy.

3 Constants

Many constants are available from SciPy.

3.1 Mathematical constants

Here is an example of using a mathematical constant.

Python Code 1: ai202209_s05_00.py

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

#
# Time-stamp: <2022/10/09 13:58:22 (CST) daisuke>
#

# importing scipy module
import scipy.constants

# value of pi
pi = scipy.constants.pi
```

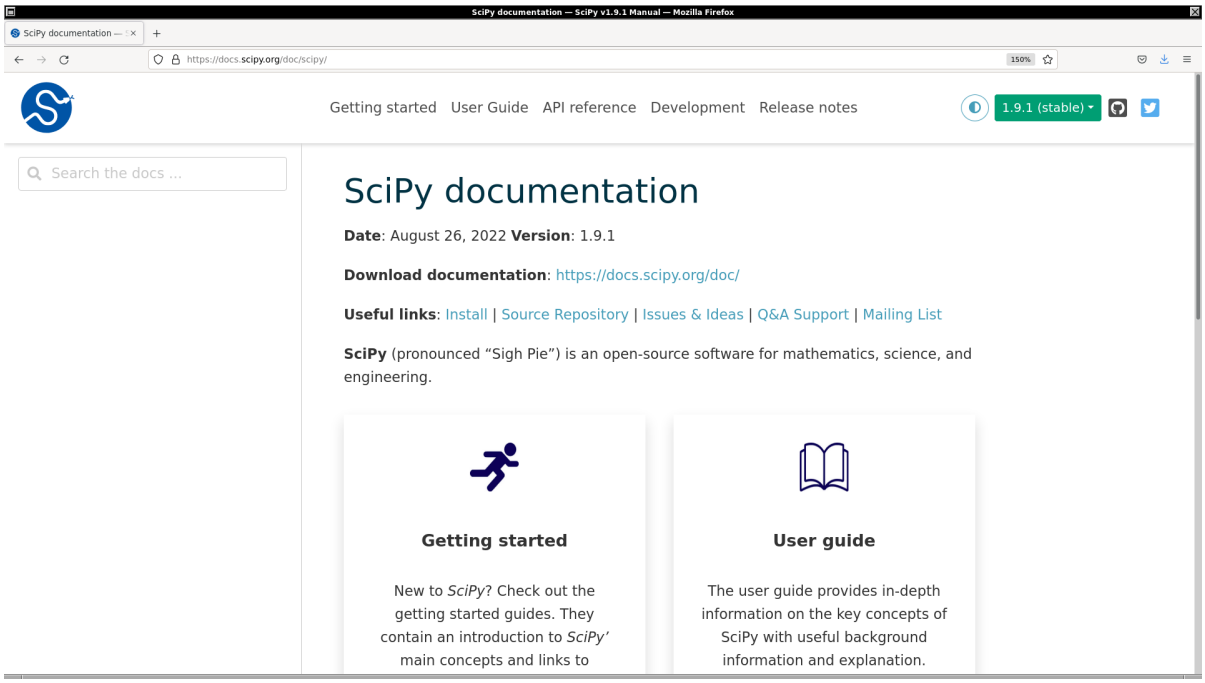

Figure 8: The official web page of SciPy documentation.

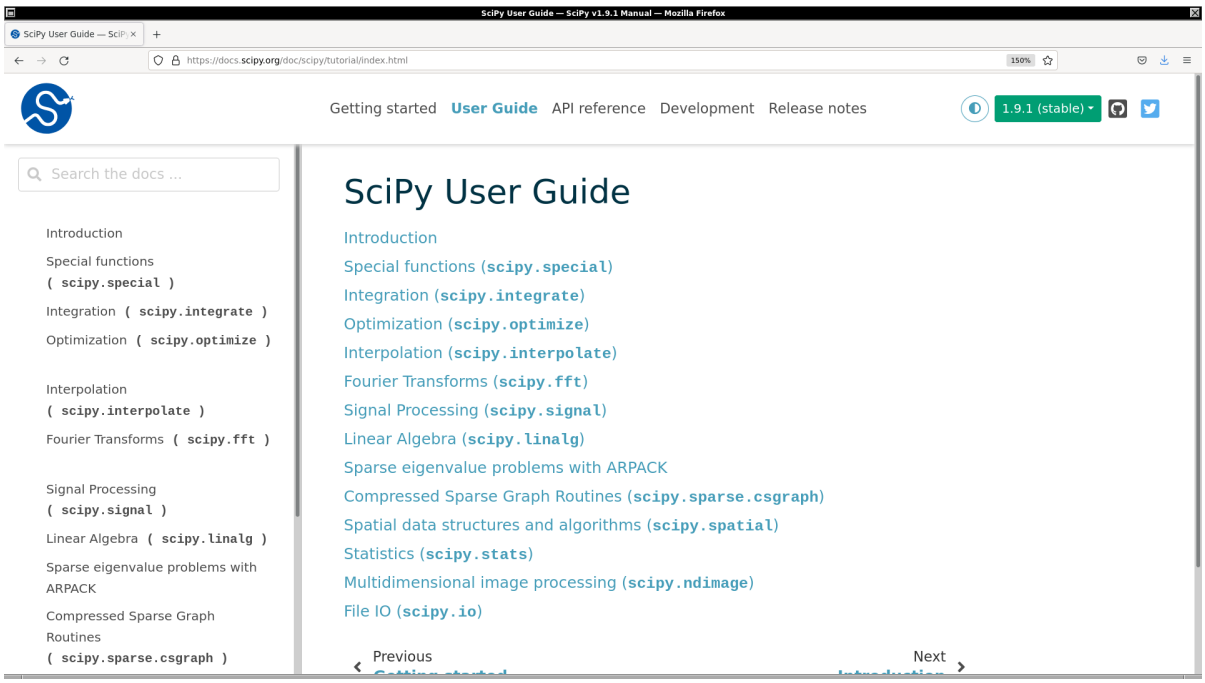

Figure 9: The official web page of SciPy user guide.

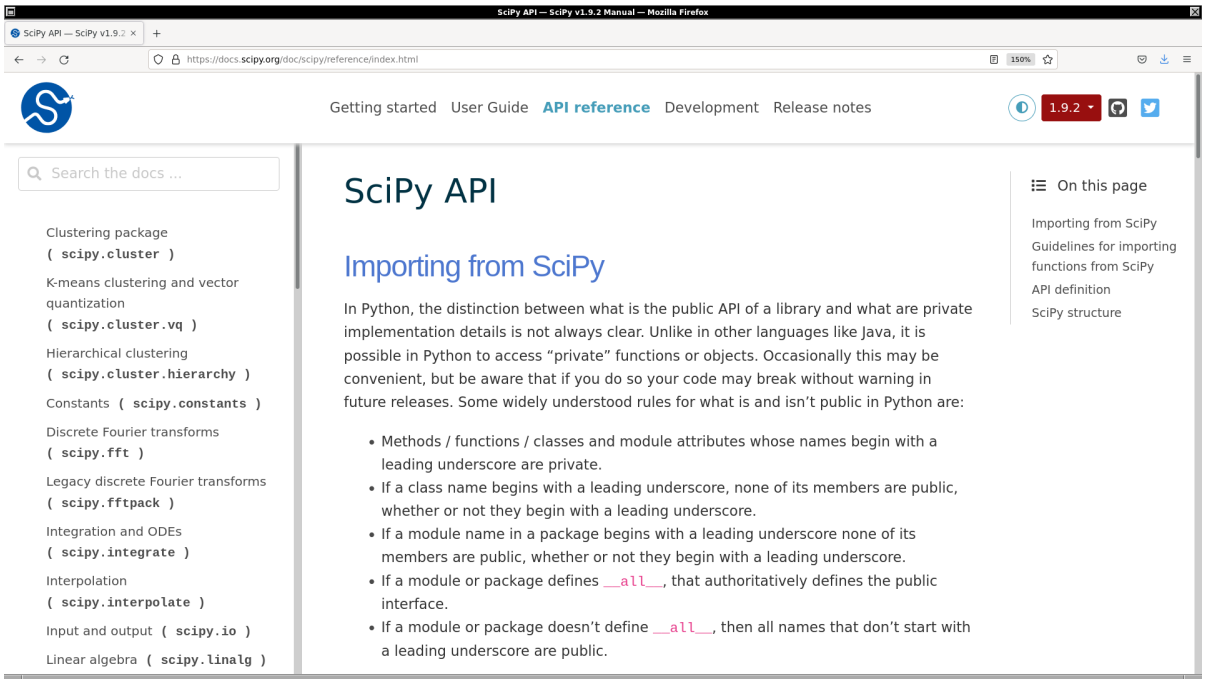

Figure 10: The official web page of SciPy API reference.

```
# printing value of pi
print (f'pi = {pi}')
```

Execute above script.

```
% ./ai202209_s05_00.py
pi = 3.141592653589793
```

Try following practice.

Practice 05-01

Print the value of golden ratio using SciPy.

3.2 Physical constants

Here is an example of using physical constants.

Python Code 2: ai202209_s05_01.py

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

#
# Time-stamp: <2022/10/09 13:58:31 (CST) daisuke>
#

# importing scipy module
import scipy.constants

#
# some physical constants
#
```

```
# speed of light in vacuum
c = scipy.constants.c

# Planck constant
h = scipy.constants.h

# gravitational constant
G = scipy.constants.G

# Avogadro constant
N_A = scipy.constants.N_A

# Boltzmann constant
k = scipy.constants.k

# Stefan-Boltzmann constant
sigma = scipy.constants.sigma

# electron mass
m_e = scipy.constants.m_e

# proton mass
m_p = scipy.constants.m_p

# printing constants
print (f'c     = {c:g}')
print (f'h     = {h:g}')
print (f'G     = {G:g}')
print (f'N_A   = {N_A:g}')
print (f'k     = {k:g}')
print (f'sigma = {sigma:g}')
print (f'm_e   = {m_e:g}')
print (f'm_p   = {m_p:g}')
```

Execute above script.

```
% ./ai202209_s05_01.py
c     = 2.99792e+08
h     = 6.62607e-34
G     = 6.6743e-11
N_A   = 6.02214e+23
k     = 1.38065e-23
sigma = 5.67037e-08
m_e   = 9.10938e-31
m_p   = 1.67262e-27
```

Try following practice.

Practice 05-02

Print the values of elementary charge and gas constant using SciPy.

Here is an example of using some more physical constants.

Python Code 3: ai202209_s05_02.py

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

#
# Time-stamp: <2022/10/09 13:58:37 (CST) daisuke>
```

```
#

# importing scipy module
import scipy.constants

#
# some more physical constants
#

# atomic mass unit
amu = scipy.constants.physical_constants['atomic mass constant']

# Bohr radius
a_0 = scipy.constants.physical_constants['Bohr radius']

# Boltzmann constant
k = scipy.constants.physical_constants['Boltzmann constant']

# electron mass
m_e = scipy.constants.physical_constants['electron mass']

# electron volt
eV = scipy.constants.physical_constants['electron volt']

# Planck constant
h = scipy.constants.physical_constants['Planck constant']

# proton mass
m_p = scipy.constants.physical_constants['proton mass']

# speed of light in vacuum
c = scipy.constants.physical_constants['speed of light in vacuum']

# standard atmospheric pressure
atm = scipy.constants.physical_constants['standard atmosphere']

# electric permittivity in vacuum
epsilon_0 = scipy.constants.physical_constants['vacuum electric permittivity']

# magnetic permeability in vacuum
mu_0 = scipy.constants.physical_constants['vacuum mag. permeability']

# printing constants
print (f'amu  = {amu[0]:g} +/- {amu[2]:g} [{amu[1]}]')
print (f'a_0  = {a_0[0]:g} +/- {a_0[2]:g} [{a_0[1]}]')
print (f'k    = {k[0]:g} +/- {k[2]:g} [{k[1]}]')
print (f'm_e  = {m_e[0]:g} +/- {m_e[2]:g} [{m_e[1]}]')
print (f'eV   = {eV[0]:g} +/- {eV[2]:g} [{eV[1]}]')
print (f'h    = {h[0]:g} +/- {h[2]:g} [{h[1]}]')
print (f'm_p  = {m_p[0]:g} +/- {m_p[2]:g} [{m_p[1]}]')
print (f'c    = {c[0]:g} +/- {c[2]:g} [{c[1]}]')
print (f'atm  = {atm[0]:g} +/- {atm[2]:g} [{atm[1]}]')
print (f'eps0 = {epsilon_0[0]:g} +/- {epsilon_0[2]:g} [{epsilon_0[1]}]')
print (f'mu0  = {mu_0[0]:g} +/- {mu_0[2]:g} [{mu_0[1]}]')
```

Execute above script.

```
% ./ai202209_s05_02.py
amu  = 1.66054e-27 +/- 5e-37 [kg]
```

```
a_0   = 5.29177e-11 +/- 8e-21 [m]
k     = 1.38065e-23 +/- 0 [J K^-1]
m_e   = 9.10938e-31 +/- 2.8e-40 [kg]
eV    = 1.60218e-19 +/- 0 [J]
h     = 6.62607e-34 +/- 0 [J Hz^-1]
m_p   = 1.67262e-27 +/- 5.1e-37 [kg]
c     = 2.99792e+08 +/- 0 [m s^-1]
atm   = 101325 +/- 0 [Pa]
eps0  = 8.85419e-12 +/- 1.3e-21 [F m^-1]
mu0   = 1.25664e-06 +/- 1.9e-16 [N A^-2]
```

Try following practice.

Practice 05-03

Print the values of neutron mass and its error using SciPy.

Here is an example of finding a constant.

Python Code 4: ai202209_s05_03.py

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

#
# Time-stamp: <2022/10/09 13:58:48 (CST) daisuke>
#

# importing scipy module
import scipy.constants

# finding a constant
search_result = scipy.constants.find ('Wien')

# printing search result
for constant in search_result:
    print (f'{constant}')
```

Execute above script.

```
% ./ai202209_s05_03.py
Wien frequency displacement law constant
Wien wavelength displacement law constant
```

Try following practice.

Practice 05-04

Try to find a physical constant of your interest, and print the value of it using SciPy.

3.3 Units

Here is an example of using non-SI units.

Python Code 5: ai202209_s05_04.py

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

#
# Time-stamp: <2022/10/09 13:58:59 (CST) daisuke>
#
```

```
# importing scipy module
import scipy.constants

# some units for length
au     = scipy.constants.au
ly     = scipy.constants.light_year
parsec = scipy.constants.parsec

# printing units for length
print (f'au     = {au:g} [m]')
print (f'ly     = {ly:g} [m]')
print (f'parsec = {parsec:g} [m]')
```

Execute above script.

```
% ./ai202209_s05_04.py
au     = 1.49598e+11 [m]
ly     = 9.46073e+15 [m]
parsec = 3.08568e+16 [m]
```

Try following practice.

Practice 05-05

Calculate the distance to Sirius in metre.

4 Random numbers

SciPy has random number generators.

4.1 Generating random numbers of uniform distribution

Generate a set of random numbers of uniform distribution. Here is an example.

Python Code 6: ai202209_s05_05.py

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

#
# Time-stamp: <2022/10/10 08:46:26 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 100 random numbers of uniform distribution between 0.0 and 100.0
ru = scipy.stats.uniform.rvs (loc=0.0, scale=100.0, size=100)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{ru}')
```

Execute above script.

```
% ./ai202209_s05_05.py
generated random numbers:
[66.96642219 37.37204796 60.94111645 58.45215098 75.97759579 95.69815318
 62.66282679 67.24796491 54.23898738 49.21719278 52.70231602 98.35634724
```

```
 19.48128134 17.06764134 10.75441094 78.74789595 56.72399593 10.32308831
 39.68780033 21.08591013 51.95866776 27.57461254 59.43587469 48.93369062
 94.18305012 22.12630801 26.56794101 76.53583531 88.58946119 73.25025876
 36.04515799  3.18177621 21.19313485 13.22508727 61.45649033 51.18465928
 75.67487097  1.98446792 79.8944423  19.43243014  5.28031602  6.10519248
 92.20093767  8.52036296 93.58632459 90.68053576 33.66937103 84.8702393
 92.58714594 47.9886461  40.65539454 21.83276492 21.94845094 39.59199913
 35.95659345 58.5146352  65.11775033 64.13660822  4.96982997 13.98433826
 87.63530263 19.9371169  37.53045117 55.59657824 72.05051964 62.33263711
 96.00190852  4.4336669   3.1789763  27.25999934 95.9647521   4.33476658
 93.07981832 88.89092755 34.34234058 98.91476899 10.67769334 84.61715877
 83.43359439 70.99898741 74.16165211 95.8281117  80.1596105  22.93061861
 39.76740522 72.12278785 46.59524061 20.83729654  9.1087158  15.24793791
 21.62657685 27.32739049 54.25845679 21.64546026 52.77570044 26.64390981
 26.42917983  3.83539394 44.60653692 50.07674838]
```

Try following practice.

Practice 05-06

Generate 1000 random numbers of uniform distribution between 1000 and 3000.

4.2 Generating random numbers of Gaussian distribution

Generate a set of random numbers of Gaussian distribution. Here is an example.

Python Code 7: ai202209_s05_06.py

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

#
# Time-stamp: <2022/10/10 08:46:46 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 100 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=100)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')
```

Execute above script.

```
% ./ai202209_s05_06.py
generated random numbers:
[ 96.7130233   99.52346426  88.26572251  99.89884437 104.08048808
  64.8850651  100.57619239  98.02975801 100.37224563  92.98054278
 118.59840555  97.48078324 103.70313309 107.6911206   95.39999309
  98.60650303  83.29285247 103.71194863  89.63025519 105.12614859
  99.94824349  98.48377849 112.21697616  92.35335624 107.22126849
 101.78827783  99.24725441  90.30195404  84.02905409 104.43835449
 108.21993707 112.03049233 105.89065115  88.33848413  98.63667264
 103.73672268 105.94077542  94.66901952  91.51238557  99.35763333
  83.54450832 104.57860363 106.5712233  100.72529171 115.8385244
  85.84552154 104.21150457 112.64093169  94.96971684  89.69498509
```

```
 107.65882761 110.58373211  93.31497666  94.02920449 110.65301508
  95.19952939 107.73384607 107.05406599 100.13506428 104.00204026
  95.46888796 105.30875692  99.59117985 104.85051167  97.72915522
  99.71863001  93.41813283 115.61884549  97.30253652 103.31556795
  94.61194556  88.22290842  97.66541555  92.79187266  93.66775648
  91.19659527 116.2186582  110.93005506  94.46789921 106.98977934
 100.34274706  79.90571854  98.32311374  95.00342838  95.18132149
 104.38390159  85.97917567  91.45981312  99.82966078 109.30880885
  95.16254306 102.33234629 101.34599639 101.53298399  91.44207414
  98.9151566   97.85894146 100.93725599 105.85151832  99.02193203]
```

Try following practice.

Practice 05-07

Generate 1000 random numbers of Gaussian distribution of mean 1000.0 and standard deviation 300.0.

5 Calculating statistical values

SciPy can be used to calculate statistical values of distributions.

5.1 Minimum and maximum values

Find the minimum and maximum values of a given distribution. Here is an example.

Python Code 8: ai202209_s05_07.py

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

#
# Time-stamp: <2022/10/10 20:43:27 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 10000 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=10000)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')

# finding minimum value
tmin = scipy.stats.tmin (rg)

# finding maximum value
tmax = scipy.stats.tmax (rg)

# printing minimum and maximum values
print (f'statistical values:')
print (f'  tmin = {tmin:8.4f}')
print (f'  tmax = {tmax:8.4f}')
```

Execute above script.

```
% ./ai202209_s05_07.py
generated random numbers:
```

```
[110.52264317  99.20638182  96.6235543  ... 103.26590017  94.52543496
  98.42368871]
statistical values:
  tmin =  63.7087
  tmax = 133.3462
```

Try following practice.

Practice 05-08

Generate 10^6 random numbers of Gaussian distribution of mean 10000 and standard deviation 2000, and find the minimum and maximum values.

5.2 Mean, variance, and standard deviation

Find the mean, variance, and standard deviation of a given distribution. Here is an example.

Python Code 9: ai202209_s05_08.py

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

#
# Time-stamp: <2022/10/10 20:43:35 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 10000 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=10000)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')

# finding minimum value
tmin = scipy.stats.tmin (rg)

# finding maximum value
tmax = scipy.stats.tmax (rg)

# calculation of arithmetic mean of distribution
mean = scipy.stats.tmean (rg)

# calculation of variance of distribution
var = scipy.stats.tvar (rg)

# calculation of standard deviation of distribution
stddev = scipy.stats.tstd (rg)

# printing arithmetic mean and standard deviation of distribution
print (f'statistical values:')
print (f'  tmin   = {tmin:8.4f}')
print (f'  tmax   = {tmax:8.4f}')
print (f'  mean   = {mean:8.4f}')
print (f'  var    = {var:8.4f}')
print (f'  stddev = {stddev:8.4f}')
```

Execute above script.

```
% ./ai202209_s05_08.py
generated random numbers:
[109.29137032  96.44114504 114.12167943 ...  91.25172191  91.02895173
  86.30897003]
statistical values:
  tmin   =  55.2194
  tmax   = 135.9143
  mean   = 100.1394
  var    =  98.9339
  stddev =   9.9466
```

Try following practice.

Practice 05-09

Generate 10^6 random numbers of Gaussian distribution of mean 10000 and standard deviation 2000, and find mean, variance, and standard deviation.

5.3 Moments of distribution

Find the moments of a given distribution. Here is an example.

Python Code 10: ai202209_s05_09.py

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

#
# Time-stamp: <2022/10/10 20:43:43 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 10000 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=10000)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')

# finding minimum value
tmin = scipy.stats.tmin (rg)

# finding maximum value
tmax = scipy.stats.tmax (rg)

# calculation of arithmetic mean of distribution
mean = scipy.stats.tmean (rg)

# calculation of variance of distribution
var = scipy.stats.tvar (rg)

# calculation of standard deviation of distribution
stddev = scipy.stats.tstd (rg)

# calculation of first moment about the mean
moment_1 = scipy.stats.moment (rg, moment=1)
```

```
# calculation of second moment about the mean
moment_2 = scipy.stats.moment (rg, moment=2)

# printing arithmetic mean and standard deviation of distribution
print (f'statistical values:')
print (f'  tmin          = {tmin:8.4f}')
print (f'  tmax          = {tmax:8.4f}')
print (f'  mean          = {mean:8.4f}')
print (f'  var           = {var:8.4f}')
print (f'  stddev        = {stddev:8.4f}')
print (f'  first moment  = {moment_1:8.4f}')
print (f'  second moment = {moment_2:8.4f}')
```

Execute above script.

```
% ./ai202209_s05_09.py
generated random numbers:
[101.3289363   98.32599469  94.34376088 ... 110.68032573  96.17707225
 104.51892656]
statistical values:
  tmin          =  65.4761
  tmax          = 140.7950
  mean          = 100.1866
  var           =  97.3763
  stddev        =   9.8679
  first moment  =   0.0000
  second moment =  97.3665
```

Try following practice.

Practice 05-10

Generate 10^6 random numbers of Gaussian distribution of mean 10000 and standard deviation 2000, and find first and second moments.

5.4 Skewness of distribution

Find the skewness of a given distribution. Here is an example.

Python Code 11: ai202209_s05_10.py

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

#
# Time-stamp: <2022/10/10 20:43:50 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 10000 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=10000)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')

# finding minimum value
```

```
tmin = scipy.stats.tmin (rg)

# finding maximum value
tmax = scipy.stats.tmax (rg)

# calculation of arithmetic mean of distribution
mean = scipy.stats.tmean (rg)

# calculation of variance of distribution
var = scipy.stats.tvar (rg)

# calculation of standard deviation of distribution
stddev = scipy.stats.tstd (rg)

# calculation of first moment about the mean
moment_1 = scipy.stats.moment (rg, moment=1)

# calculation of second moment about the mean
moment_2 = scipy.stats.moment (rg, moment=2)

# calculation of skewness
skew = scipy.stats.skew (rg)

# printing arithmetic mean and standard deviation of distribution
print (f'statistical values:')
print (f'  tmin          = {tmin:8.4f}')
print (f'  tmax          = {tmax:8.4f}')
print (f'  mean          = {mean:8.4f}')
print (f'  var           = {var:8.4f}')
print (f'  stddev        = {stddev:8.4f}')
print (f'  first moment  = {moment_1:8.4f}')
print (f'  second moment = {moment_2:8.4f}')
print (f'  skewness      = {skew:8.4f}')
```

Execute above script.

```
% ./ai202209_s05_10.py
generated random numbers:
[106.67535486 105.37678688 104.10438511 ...  79.17637222 107.13432909
 112.32169494]
statistical values:
  tmin          =  62.1047
  tmax          = 139.7473
  mean          =  99.9820
  var           =  98.3403
  stddev        =   9.9167
  first moment  =   0.0000
  second moment =  98.3305
  skewness      =   0.0530
```

Try following practice.

Practice 05-11

Generate 10^6 random numbers of Gaussian distribution of mean 10000 and standard deviation 2000, and find skewness.

5.5 Kurtosis of distribution

Find the kurtosis of a given distribution. Here is an example.

Python Code 12: ai202209_s05_11.py

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

#
# Time-stamp: <2022/10/11 06:34:11 (CST) daisuke>
#

# importing scipy module
import scipy.stats

# generating 10000 random numbers of Gaussian distribution
# of mean=100.0 and stddev=10.0
rg = scipy.stats.norm.rvs (loc=100.0, scale=10.0, size=10000)

# printing generated random numbers
print (f'generated random numbers:')
print (f'{rg}')

# calculation of statistical values
stat_values = scipy.stats.describe (rg)

# printing statistical values
print (f'statistical values:')
print (f'  number of data = {stat_values.nobs}')
print (f'  minimum value  = {stat_values.minmax[0]:8.4f}')
print (f'  maximum value  = {stat_values.minmax[1]:8.4f}')
print (f'  mean           = {stat_values.mean:8.4f}')
print (f'  variance       = {stat_values.variance:8.4f}')
print (f'  skewness       = {stat_values.skewness:8.4f}')
print (f'  kurtosis       = {stat_values.kurtosis:8.4f}')
```

Execute above script.

```
% ./ai202209_s05_11.py
generated random numbers:
[100.89610029  94.51826331  82.74272651 ... 110.74488907  80.94563404
 109.14713886]
statistical values:
  number of data = 10000
  minimum value  =  61.9259
  maximum value  = 137.7832
  mean           =  99.9966
  variance       = 100.4909
  skewness       =  -0.0105
  kurtosis       =  -0.0042
```

Try following practice.

Practice 05-12

Generate 10^6 random numbers of Gaussian distribution of mean 10000 and standard deviation 2000, and find kurtosis.

6 Linear algebra

SciPy offers some functions for linear algebra.

6.1 Inverse matrix

Find the inverse matrix of a given matrix. Here is an example.

Python Code 13: ai202209_s05_12.py

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

#
# Time-stamp: <2022/10/12 08:24:00 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.linalg

# matrix A
A = numpy.array ( [ [4.0, 7.0], [3.0, 5.0] ] )

# printing matrix A
print (f'matrix A:\n{A}')

# determinant of matrix A
A_det = scipy.linalg.det (A)

# printing the determinant of matrix A
print (f'determinant of matrix A = {A_det}')

# inverse of matrix A
A_inv = scipy.linalg.inv (A)

# printing inverse of matrix A
print (f'A^{-1}:\n{A_inv}')

# calculation of A @ A_inv
B = A @ A_inv

# printing matrix B
print (f'matrix B = A @ A_inv:\n{B}')
```

Execute above script.

```
% ./ai202209_s05_12.py
matrix A:
[[4. 7.]
 [3. 5.]]
determinant of matrix A = -1.0000000000000027
A^-1:
[[-5.  7.]
 [ 3. -4.]]
matrix B = A @ A_inv:
[[1. 0.]
 [0. 1.]]
```

Try following practice.

Practice 05-13

Make a 2×2 matrix, and find its inverse matrix using SciPy.

6.2 Eigenvalues and eigenvectors

Find the eigenvalues and eigenvectors of a given matrix. Here is an example.

Python Code 14: ai202209_s05_13.py

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

#
# Time-stamp: <2022/10/12 08:42:25 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.linalg

# matrix A
A = numpy.array ( [ [3.0, 1.0], [2.0, 2.0] ] )

# printing matrix A
print (f'matrix A:\n{A}')

# eigenvalues and eigenvectors of matrix A
eigenvalvec = scipy.linalg.eig (A)

# printing eigenvalues and eigenvectors of matrix A
print (f'eigenvalues of matrix A:\n{eigenvalvec[0]}')
print (f'eigenvectors of matrix A:\n{eigenvalvec[1]}')

# the other way to get eigenvalues of matrix A
eigenvalues = scipy.linalg.eigvals (A)

# printing eigenvalues of matrix A
print (f'eigenvalues of matrix A:\n{eigenvalues}')

# making matrix P
P = numpy.array ( [ eigenvalvec[1][0], eigenvalvec[1][1] ] )
print (f'P:\n{P}')

# making matrix P^-1
P_inv = scipy.linalg.inv (P)

# printing matrix P^-1
print (f'P_inv:\n{P_inv}')

# calculation of P^-1 A P
D = P_inv @ A @ P

# printing diagonalised matrix D
print (f'D:\n{D}')
```

Execute above script.

```
% ./ai202209_s05_13.py
matrix A:
[[3. 1.]
 [2. 2.]]
eigenvalues of matrix A:
[4.+0.j 1.+0.j]
eigenvectors of matrix A:
[[ 0.70710678 -0.4472136 ]
 [ 0.70710678  0.89442719]]
eigenvalues of matrix A:
[4.+0.j 1.+0.j]
P:
[[ 0.70710678 -0.4472136 ]
 [ 0.70710678  0.89442719]]
P_inv:
[[ 0.94280904  0.47140452]
 [-0.74535599  0.74535599]]
D:
[[ 4.00000000e+00  0.00000000e+00]
 [-1.11022302e-16  1.00000000e+00]]
```

Try following practice.

Practice 05-14

Make a 2×2 matrix, and find its eigenvalues and eigenvectors using SciPy.

7 Interpolation

SciPy can be used to carry out interpolation.

7.1 Linear interpolation

Try linear interpolation. Here is an example.

Python Code 15: ai202209_s05_14.py

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

#
# Time-stamp: <2022/10/16 23:38:20 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_14.png'

# generating data for interpolation
```

```
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 2.0 * data_x + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for linear interpolation
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='linear')

# getting Y-value for X-value at x=2.5
x1 = 2.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={2.0*x1+3.0}')

# getting Y-value for X-value at x=3.7
x2 = 3.7
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={2.0*x2+3.0}')

#
# making a plot using matplotlib
#

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

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

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

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.1)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='linear interpolation', zorder=0.0)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='cyan', label='result of interpolation #1', zorder=0.2)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.3)

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

```
# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_14.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [ 3.  5.  7.  9. 11. 13. 15. 17. 19. 21. 23.]
func_interp (2.5) = 8.0
what we expect is: x=2.5 --> y=8.0
func_interp (3.7) = 10.4
what we expect is: x=3.7 --> y=10.4
```

Display PNG file. (Fig. 11)

```
% feh -dF ai202209_s05_14.png
```

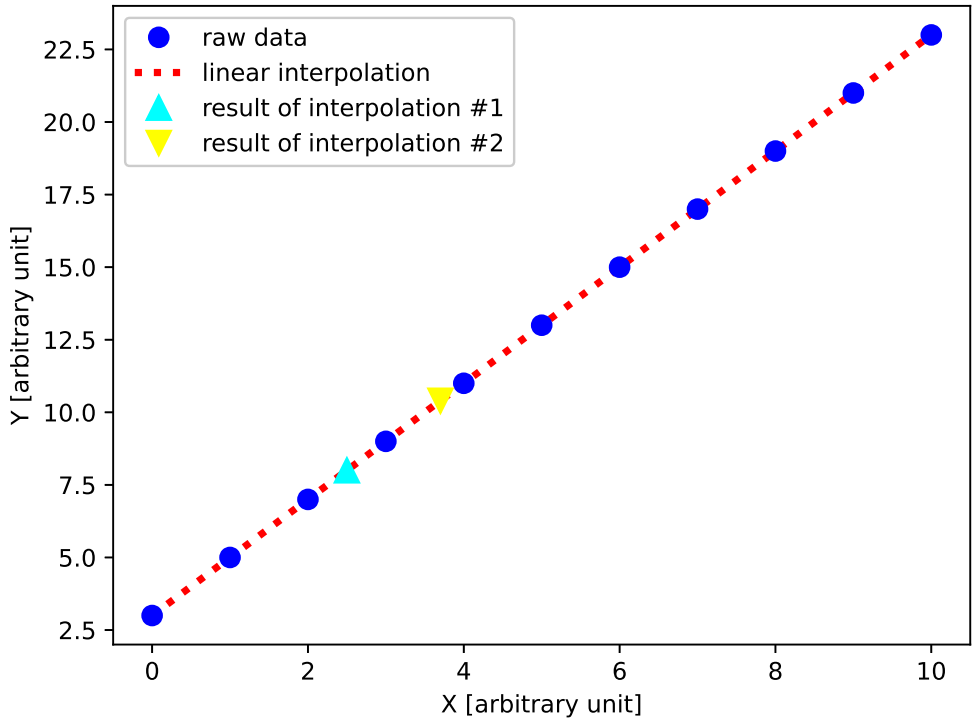

Figure 11: An example of linear interpolation using SciPy.

Try following practice.

Practice 05-15

Try linear interpolation using SciPy.

7.2 Quadratic interpolation

Try linear interpolation for a curve. Here is an example.

Python Code 16: ai202209_s05_15.py

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

#
# Time-stamp: <2022/10/16 23:44:26 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_15.png'

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 0.5 * (data_x - 4.0)**2 + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for linear interpolation
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='linear')

# getting Y-value for X-value at x=5.5
x1 = 5.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={0.5*(x1-4.0)**2+3.0}')

# getting Y-value for X-value at x=7.8
x2 = 7.8
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={0.5*(x2-4.0)**2+3.0}')

#
# making a plot using matplotlib
#

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

# making a canvas object
```

```
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)

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

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = 0.5 * (data_x0 - 4.0)**2 + 3.0
ax.plot (data_x0, data_y0, linestyle='--', linewidth=2.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='linear interpolation', zorder=0.1)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='magenta', label='result of interpolation #1', zorder=0.3)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.4)

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

# legend
ax.legend ()

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

Execute above script. The results of interpolation is not very good.

```
% ./ai202209_s05_15.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [11.   7.5  5.   3.5  3.   3.5  5.   7.5 11.  15.5 21. ]
func_interp (5.5) = 4.25
what we expect is: x=5.5 --> y=4.125
func_interp (7.8) = 10.299999999999999
what we expect is: x=7.8 --> y=10.219999999999999
```

Display PNG file. (Fig. 12)

```
% feh -dF ai202209_s05_15.png
```

Try quadratic interpolation for a curve. Here is an example.

Python Code 17: ai202209_s05_16.py

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

#
```

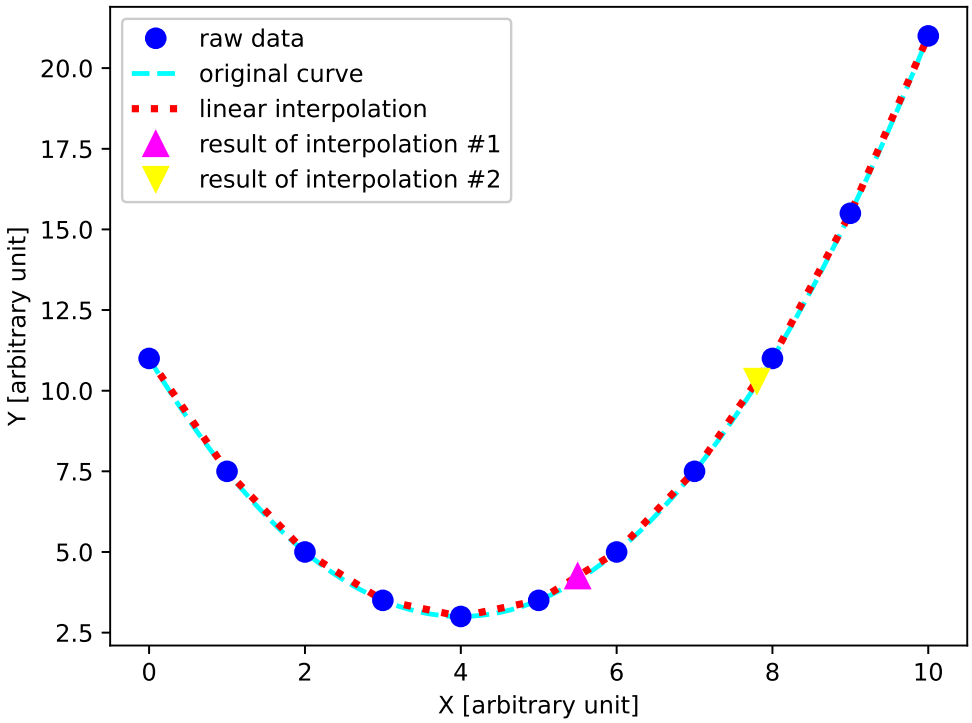

Figure 12: An example of linear interpolation for a curve using SciPy.

```
# Time-stamp: <2022/10/16 23:47:38 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_16.png'

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 0.5 * (data_x - 4.0)**2 + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for quadratic interpolation
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='quadratic')
```

```
# getting Y-value for X-value at x=5.5
x1 = 5.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={0.5*(x1-4.0)**2+3.0}')

# getting Y-value for X-value at x=7.8
x2 = 7.8
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={0.5*(x2-4.0)**2+3.0}')

#
# making a plot using matplotlib
#

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

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

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

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = 0.5 * (data_x0 - 4.0)**2 + 3.0
ax.plot (data_x0, data_y0, linestyle='--', linewidth=2.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='quadratic interpolation', zorder=0.1)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='magenta', label='result of interpolation #1', zorder=0.3)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.4)

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

# legend
ax.legend ()

# saving the figure to a file
```

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

Execute above script.

```
% ./ai202209_s05_16.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [11.   7.5  5.   3.5  3.   3.5  5.   7.5 11.  15.5 21. ]
func_interp (5.5) = 4.125
what we expect is: x=5.5 --> y=4.125
func_interp (7.8) = 10.219999999999999
what we expect is: x=7.8 --> y=10.219999999999999
```

Display PNG file. (Fig. 13)

```
% feh -dF ai202209_s05_16.png
```

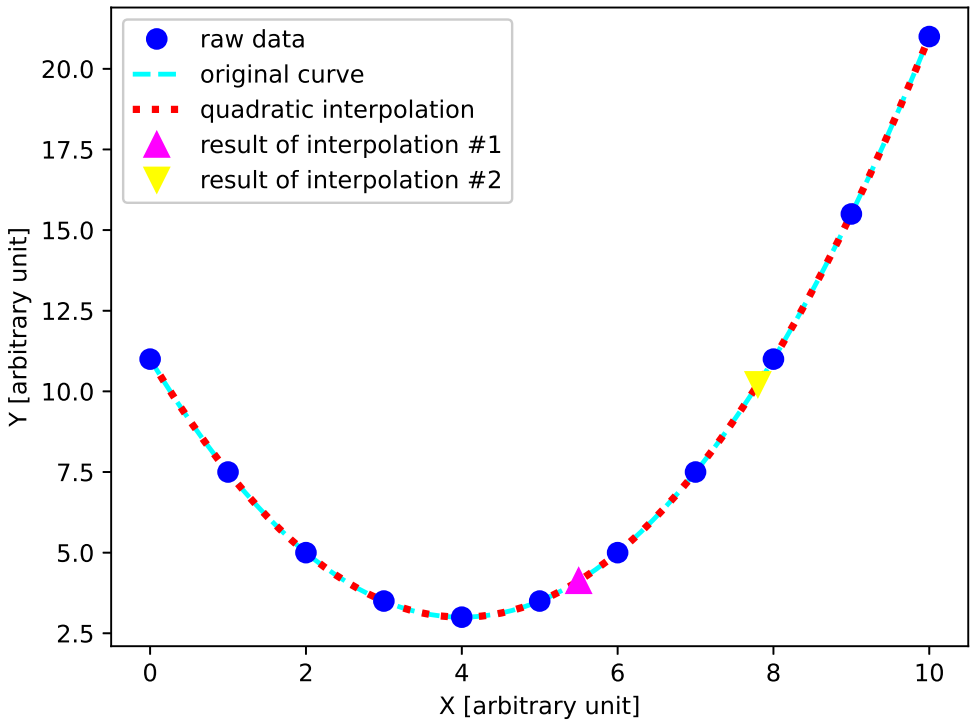

Figure 13: An example of quadratic interpolation for a curve using SciPy.

Try following practice.

Practice 05-16

Try quadratic interpolation using SciPy.

7.3 Cubic interpolation

Try linear interpolation for a sine curve. Here is an example.

Python Code 18: ai202209_s05_17.py

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

#
# Time-stamp: <2022/10/16 23:50:50 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_17.png'

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 2.0 * numpy.sin (data_x - 1.0) + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for linear interpolation
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='linear')

# getting Y-value for X-value at x=5.5
x1 = 5.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={2.0*numpy.sin (x1-1.0)+3.0}')

# getting Y-value for X-value at x=7.8
x2 = 7.8
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={2.0*numpy.sin (x2-1.0)+3.0}')

#
# making a plot using matplotlib
#

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

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

# making an axes object
```

```
ax = fig.add_subplot (111)

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = 2.0 * numpy.sin (data_x0 - 1.0) + 3.0
ax.plot (data_x0, data_y0, linestyle='--', linewidth=2.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='linear interpolation', zorder=0.1)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='magenta', label='result of interpolation #1', zorder=0.3)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.4)

# range of plot
ax.set_xlim (-0.5, 10.5)
ax.set_ylim (0.0, 8.0)

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

# legend
ax.legend ()

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

Execute above script. The results of interpolation is not very good.

```
% ./ai202209_s05_17.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [1.31705803 3.         4.68294197 4.81859485 3.28224002 1.48639501
 1.08215145 2.441169   4.3139732  4.97871649 3.82423697]
func_interp (5.5) = 1.2842732300289335
what we expect is: x=5.5 --> y=1.044939764669806
func_interp (7.8) = 3.939412358670492
what we expect is: x=7.8 --> y=3.9882267022772164
```

Display PNG file. (Fig. 14)

```
% feh -dF ai202209_s05_17.png
```

Try quadratic interpolation for a sine curve. Here is an example.

Python Code 19: ai202209_s05_18.py

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

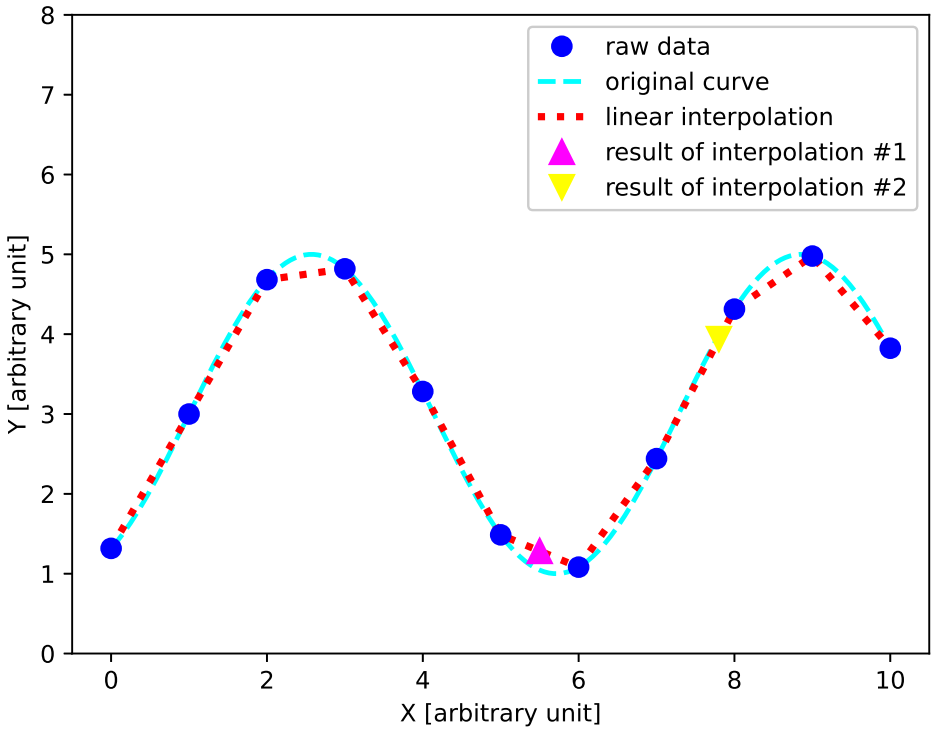

Figure 14: An example of linear interpolation for a sine curve using SciPy.

```
#
# Time-stamp: <2022/10/16 23:53:11 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_18.png'

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 2.0 * numpy.sin (data_x - 1.0) + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for quadratic interpolation
```

```
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='quadratic')

# getting Y-value for X-value at x=5.5
x1 = 5.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={2.0*numpy.sin (x1-1.0)+3.0}')

# getting Y-value for X-value at x=7.8
x2 = 7.8
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={2.0*numpy.sin (x2-1.0)+3.0}')

#
# making a plot using matplotlib
#

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

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

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

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = 2.0 * numpy.sin (data_x0 - 1.0) + 3.0
ax.plot (data_x0, data_y0, linestyle='--', linewidth=2.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='quadratic interpolation', zorder=0.1)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='magenta', label='result of interpolation #1', zorder=0.3)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.4)

# range of plot
ax.set_xlim (-0.5, 10.5)
ax.set_ylim (0.0, 8.0)

# labels
ax.set_xlabel ('X [arbitrary unit]')
```

```
ax.set_ylabel ('Y [arbitrary unit]')

# legend
ax.legend ()

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

Execute above script. The results of quadratic interpolation is much better than the results from linear interpolation.

```
% ./ai202209_s05_18.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [1.31705803 3.         4.68294197 4.81859485 3.28224002 1.48639501
 1.08215145 2.441169   4.3139732  4.97871649 3.82423697]
func_interp (5.5) = 1.0615256669000928
what we expect is: x=5.5 --> y=1.044939764669806
func_interp (7.8) = 4.001498612238118
what we expect is: x=7.8 --> y=3.9882267022772164
```

Display PNG file. (Fig. 15)

```
% feh -dF ai202209_s05_18.png
```

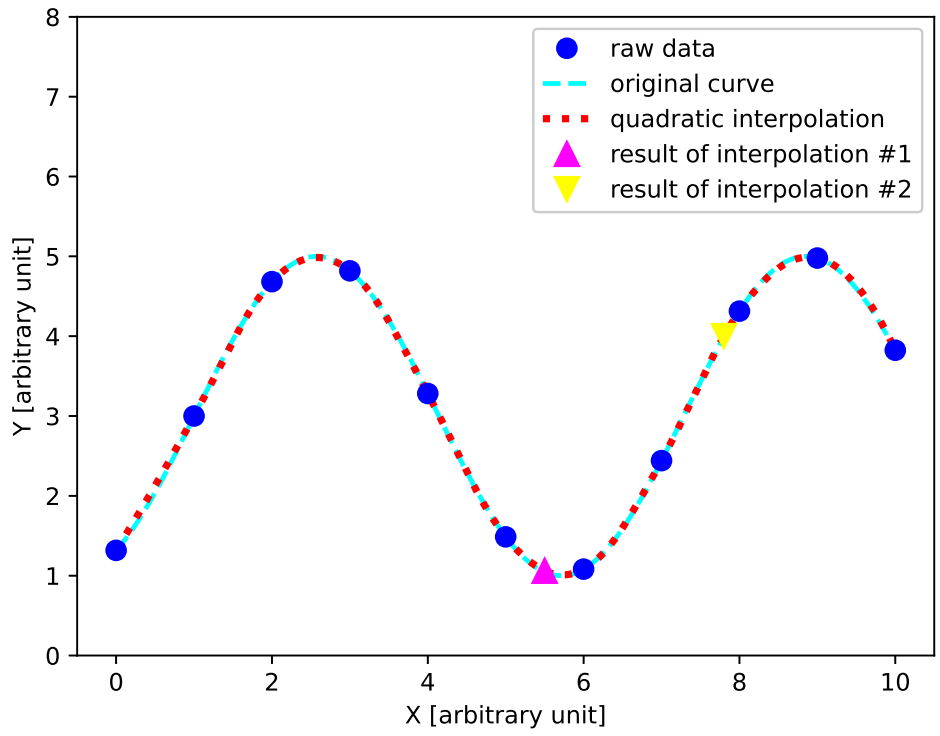

Figure 15: An example of quadratic interpolation for a sine curve using SciPy.

Try cubic interpolation for a sine curve. Here is an example.

Python Code 20: ai202209_s05_19.py

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

#
# Time-stamp: <2022/10/16 23:55:18 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_19.png'

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = 2.0 * numpy.sin (data_x - 1.0) + 3.0

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# making a function for quadratic interpolation
func_interp = scipy.interpolate.interp1d (data_x, data_y, kind='cubic')

# getting Y-value for X-value at x=5.5
x1 = 5.5
y1 = func_interp (x1)

# printing result
print (f'func_interp ({x1}) = {y1}')
print (f'what we expect is: x={x1} --> y={2.0*numpy.sin (x1-1.0)+3.0}')

# getting Y-value for X-value at x=7.8
x2 = 7.8
y2 = func_interp (x2)

# printing result
print (f'func_interp ({x2}) = {y2}')
print (f'what we expect is: x={x2} --> y={2.0*numpy.sin (x2-1.0)+3.0}')

#
# making a plot using matplotlib
#

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

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

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

```
# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = 2.0 * numpy.sin (data_x0 - 1.0) + 3.0
ax.plot (data_x0, data_y0, linestyle='--', linewidth=2.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi = numpy.linspace (0.0, 10.0, 1001)
data_yi = func_interp (data_xi)
ax.plot (data_xi, data_yi, linestyle=':', linewidth=3.0, color='red', \
         label='cubic interpolation', zorder=0.1)

# plotting interpolated values
ax.plot (x1, y1, linestyle='None', marker='^', markersize=10.0, \
         color='magenta', label='result of interpolation #1', zorder=0.3)
ax.plot (x2, y2, linestyle='None', marker='v', markersize=10.0, \
         color='yellow', label='result of interpolation #2', zorder=0.4)

# range of plot
ax.set_xlim (-0.5, 10.5)
ax.set_ylim (0.0, 8.0)

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

# legend
ax.legend ()

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

Execute above script. The results of cubic interpolation is even better than the results from quadratic interpolation.

```
% ./ai202209_s05_19.py
data_x = [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
data_y = [1.31705803 3.         4.68294197 4.81859485 3.28224002 1.48639501
 1.08215145 2.441169   4.3139732  4.97871649 3.82423697]
func_interp (5.5) = 1.0517022390374833
what we expect is: x=5.5 --> y=1.044939764669806
func_interp (7.8) = 3.991954097506399
what we expect is: x=7.8 --> y=3.9882267022772164
```

Display PNG file. (Fig. 16)

```
% feh -dF ai202209_s05_19.png
```

Try following practice.

Practice 05-17

Try cubic interpolation using SciPy.

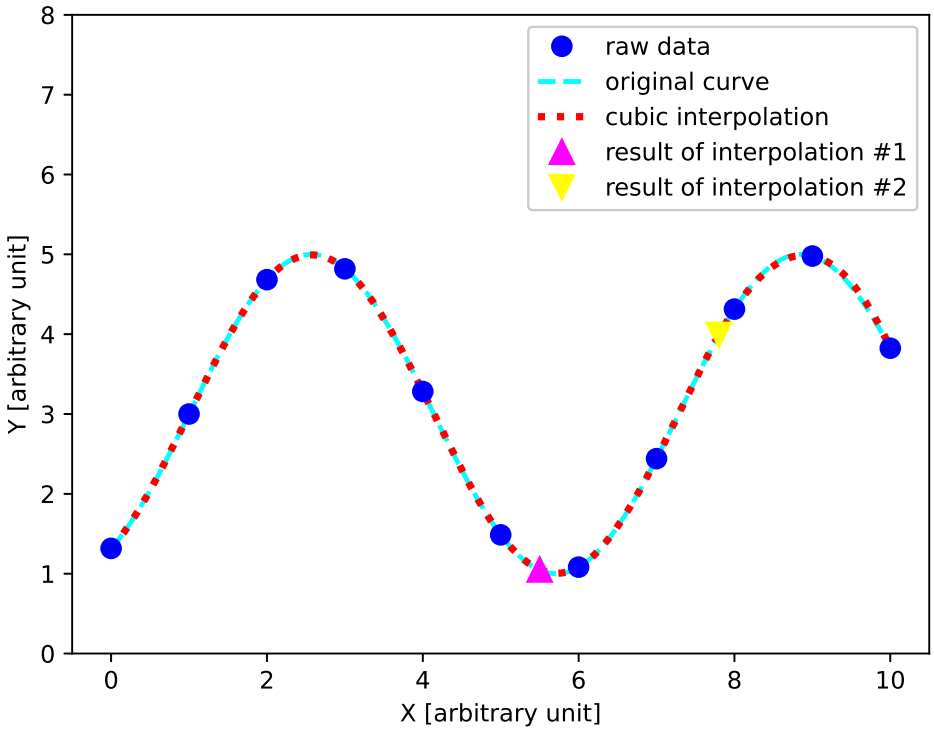

Figure 16: An example of cubic interpolation for a sine curve using SciPy.

7.4 4th order spline interpolation

Try 4th order spline interpolation. Here is an example.

Python Code 21: ai202209_s05_20.py

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

#
# Time-stamp: <2022/10/16 23:59:59 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.interpolate

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

# output image file
file_output = 'ai202209_s05_20.png'

# function for a curve
def curve (x):
    y = 10.0 * numpy.sin (x / 2.0) * numpy.cos (x / 3.0)**2 \
        * numpy.exp (-x / 10)
```

```
    return (y)

# generating data for interpolation
data_x = numpy.linspace (0.0, 10.0, 11)
data_y = curve (data_x)

# making a function for linear interpolation
spline1 = scipy.interpolate.InterpolatedUnivariateSpline (data_x, data_y, k=1)

# making a function for quadratic interpolation
spline2 = scipy.interpolate.InterpolatedUnivariateSpline (data_x, data_y, k=2)

# making a function for cubic interpolation
spline3 = scipy.interpolate.InterpolatedUnivariateSpline (data_x, data_y, k=3)

# making a function for 4th-order spline interpolation
spline4 = scipy.interpolate.InterpolatedUnivariateSpline (data_x, data_y, k=4)

#
# making a plot using matplotlib
#

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

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

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

# plotting data points
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=8.0, \
         color='blue', label='raw data', zorder=0.2)

# plotting original curve
data_x0 = numpy.linspace (0.0, 10.0, 1001)
data_y0 = curve (data_x0)
ax.plot (data_x0, data_y0, linestyle=':', linewidth=4.0, color='cyan', \
         label='original curve', zorder=0.0)

# plotting result of interpolation
data_xi1 = numpy.linspace (0.0, 10.0, 1001)
data_yi1 = spline1 (data_xi1)
ax.plot (data_xi1, data_yi1, linestyle='--', linewidth=2.0, color='magenta', \
         label='linear', zorder=0.1)

# plotting result of interpolation
data_xi2 = numpy.linspace (0.0, 10.0, 1001)
data_yi2 = spline2 (data_xi2)
ax.plot (data_xi2, data_yi2, linestyle='--', linewidth=3.0, color='yellow', \
         label='quadratic', zorder=0.1)

# plotting result of interpolation
data_xi3 = numpy.linspace (0.0, 10.0, 1001)
data_yi3 = spline3 (data_xi3)
ax.plot (data_xi3, data_yi3, linestyle='-.', linewidth=2.0, color='red', \
         label='cubic', zorder=0.1)
```

```
# plotting result of interpolation
data_xi4 = numpy.linspace (0.0, 10.0, 1001)
data_yi4 = spline4 (data_xi4)
ax.plot (data_xi4, data_yi4, linestyle='-', linewidth=1.0, color='green', \
         label='4th-order spline', zorder=0.1)

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

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_20.py
```

Display PNG file. (Fig. 17)

```
% feh -dF ai202209_s05_20.png
```

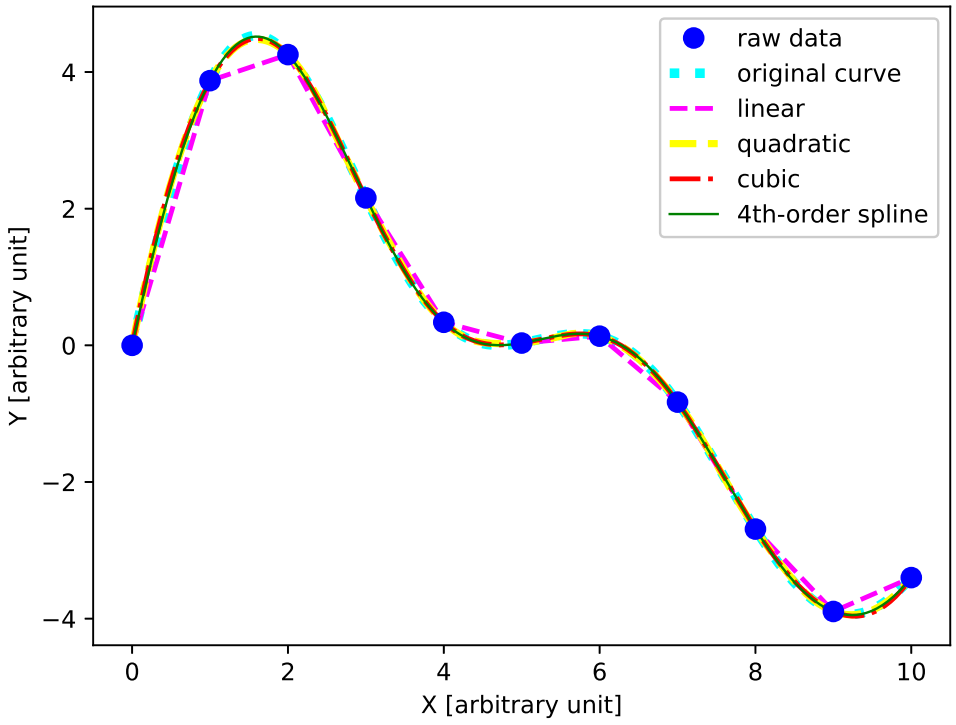

Figure 17: An example of 4th order spline interpolation using SciPy.

Try following practice.

Practice 05-18

Try 4th order spline interpolation using SciPy.

8 Numerical integration

Try numerical integration using SciPy.

8.1 Numerical integration of a given function

Calculate following integration numerically using SciPy.

$$I = \int_0^{\pi} \sin x dx \tag{1}$$

Here is an example.

Python Code 22: ai202209_s05_21.py

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

#
# Time-stamp: <2022/10/13 16:41:01 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of a curve
def curve (x):
    # curve
    y = numpy.sin (x)
    # returning a value
    return (y)

# range of integration
x0 = 0.0
x1 = numpy.pi

# numerical integration
result1 = scipy.integrate.quad (curve, x0, x1)

# printing result of numerical integration
print (f'integ. of sin (x) from 0.0 to pi = {result1[0]} +/- {result1[1]}')
```

Execute above script.

```
% ./ai202209_s05_21.py
integ. of sin (x) from 0.0 to pi = 2.0 +/- 2.220446049250313e-14
```

Try following practice.

Practice 05-19

Calculate following integration numerically using SciPy.

$$I = \int_0^{\frac{\pi}{2}} \cos x dx$$

Calculate following integration numerically using SciPy.

$$I = \int_0^2 \sqrt{4 - x^2} dx \tag{2}$$

Here is an example.

Python Code 23: ai202209_s05_22.py

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

#
# Time-stamp: <2022/10/13 21:29:43 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of a curve
def curve (x):
    # curve
    y = numpy.sqrt (4.0 - x**2)
    # returning a value
    return (y)

# range of integration
x0 = 0.0
x1 = 2.0

# numerical integration
result1 = scipy.integrate.quad (curve, x0, x1)

# printing result of numerical integration
print (f'integ. of sqrt (4-x) from 0 to 2 = {result1[0]} +/- {result1[1]}')
```

Execute above script.

```
% ./ai202209_s05_22.py
integ. of sqrt (4-x) from 0 to 2 = 3.1415926535897922 +/- 3.533564552071766e-10
```

Try following practice.

Practice 05-20

Calculate following integration numerically using SciPy.

$$I = \int_0^3 \sqrt{9 - x^2} dx$$

Calculate Gaussian distribution numerically using SciPy. Here is an example.

Python Code 24: ai202209_s05_23.py

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

#
# Time-stamp: <2022/10/14 06:15:00 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of standard normal distribution
def curve (x):
    # curve
    y = numpy.exp (-x**2 / 2.0) / numpy.sqrt (2.0 * numpy.pi)
    # returning a value
    return (y)

# numerical integration
#   numpy.NINF = negative infinity
#   numpy.PINF = positive infinity
result0 = scipy.integrate.quad (curve, numpy.NINF, numpy.PINF)
result1 = scipy.integrate.quad (curve, -1.0, +1.0)
result2 = scipy.integrate.quad (curve, -2.0, +2.0)
result3 = scipy.integrate.quad (curve, -3.0, +3.0)
result4 = scipy.integrate.quad (curve, -4.0, +4.0)
result5 = scipy.integrate.quad (curve, -5.0, +5.0)

# printing result of numerical integration
print (f'integ. of std normal func. from -inf to +inf:\n', \
       f' I0 = {result0[0]} +/- {result0[1]}')
print (f'integ. of std normal func. from -1 to +1:\n', \
       f' I1 = {result1[0]} +/- {result1[1]}')
print (f'integ. of std normal func. from -2 to +2:\n', \
       f' I2 = {result2[0]} +/- {result2[1]}')
print (f'integ. of std normal func. from -3 to +3:\n', \
       f' I3 = {result3[0]} +/- {result3[1]}')
print (f'integ. of std normal func. from -4 to +4:\n', \
       f' I4 = {result4[0]} +/- {result4[1]}')
print (f'integ. of std normal func. from -5 to +5:\n', \
       f' I5 = {result5[0]} +/- {result5[1]}')
```

Execute above script.

```
% ./ai202209_s05_23.py
integ. of std normal func. from -inf to +inf:
  I0 = 0.9999999999999998 +/- 1.0178191320905743e-08
integ. of std normal func. from -1 to +1:
  I1 = 0.682689492137086 +/- 7.579375928402476e-15
integ. of std normal func. from -2 to +2:
  I2 = 0.9544997361036417 +/- 1.8403548653972358e-11
integ. of std normal func. from -3 to +3:
  I3 = 0.9973002039367399 +/- 1.1072256503105314e-14
integ. of std normal func. from -4 to +4:
  I4 = 0.9999366575163339 +/- 4.838904125482879e-12
integ. of std normal func. from -5 to +5:
```

```
I5 = 0.9999994266968565 +/- 8.668320228277793e-10
```

Try following practice.

Practice 05-21

Integrate Gaussian distribution from -2.5σ to $+2.5\sigma$.

8.2 Numerical integration by given samples

Use trapezoidal rule to integrate given samples of data. Here is an example.

$$I = \int_0^1 (2x + 1)dx \tag{3}$$

Python Code 25: ai202209_s05_24.py

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

#
# Time-stamp: <2022/10/14 06:42:33 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of f(x) = 2x + 1
def line (x):
    # curve
    y = 2.0 * x + 1
    # returning a value
    return (y)

# data points
data_x = numpy.linspace (0.0, 1.0, 1001)
data_y = line (data_x)

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# numerical integration of given data points by trapezoidal rule
I = scipy.integrate.trapezoid (data_y, x=data_x)

# printing result of numerical integration
print (f'I = {I}')
```

Execute above script.

```
% ./ai202209_s05_24.py
data_x = [0.    0.001 0.002 ... 0.998 0.999 1.   ]
data_y = [1.    1.002 1.004 ... 2.996 2.998 3.   ]
I = 2.0
```

Try following practice.

Practice 05-22

Use trapezoidal rule to integrate your favourite function.

Here is an example for numerical integration by trapezoidal rule for following.

$$I = \int_0^2 (-x^2 + 2x)dx \tag{4}$$

Python Code 26: ai202209_s05_25.py

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

#
# Time-stamp: <2022/10/14 06:48:54 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of f(x) = -x^2 + 2x
def curve (x):
    # curve
    y = -x**2 + 2.0 * x
    # returning a value
    return (y)

# data points
data_x = numpy.linspace (0.0, 2.0, 2001)
data_y = curve (data_x)

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# numerical integration of given data points by trapezoidal rule
I = scipy.integrate.trapezoid (data_y, x=data_x)

# printing result of numerical integration
print (f'I = {I}')
```

Execute above script.

```
% ./ai202209_s05_25.py
data_x = [0.000e+00 1.000e-03 2.000e-03 ... 1.998e+00 1.999e+00 2.000e+00]
data_y = [0.       0.001999 0.003996 ... 0.003996 0.001999 0.      ]
I = 1.333333
```

Here is an example for numerical integration by trapezoidal rule for following.

$$I = \int_0^{\frac{\pi}{2}} \cos x dx \tag{5}$$

Python Code 27: ai202209_s05_26.py

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

#
# Time-stamp: <2022/10/14 06:57:42 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# function of f(x) = cos (x)
def curve (x):
    # curve
    y = numpy.cos (x)
    # returning a value
    return (y)

# data points
data_x = numpy.linspace (0.0, numpy.pi / 2.0, 1001)
data_y = curve (data_x)

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# numerical integration of given data points by trapezoidal rule
I = scipy.integrate.trapezoid (data_y, x=data_x)

# printing result of numerical integration
print (f'I = {I}')
```

Execute above script.

```
% ./ai202209_s05_26.py
data_x = [0.         0.0015708  0.00314159 ... 1.56765473 1.56922553 1.57079633]
data_y = [1.00000000e+00 9.99998766e-01 9.99995065e-01 ... 3.14158749e-03
 1.57079568e-03 6.12323400e-17]
I = 0.9999997943832332
```

Use Simpson's rule to integrate following.

$$I = \int_0^{\frac{\pi}{2}} \cos x dx \tag{6}$$

Python Code 28: ai202209_s05_27.py

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

#
# Time-stamp: <2022/10/14 07:18:01 (CST) daisuke>
#

# importing numpy module
import numpy
```

```
# importing scipy module
import scipy
import scipy.integrate

# function of f(x) = cos (x)
def curve (x):
    # curve
    y = numpy.cos (x)
    # returning a value
    return (y)

# data points
data_x = numpy.linspace (0.0, numpy.pi / 2.0, 1001)
data_y = curve (data_x)

# printing data_x and data_y
print (f'data_x = {data_x}')
print (f'data_y = {data_y}')

# numerical integration of given data points by Simpson's rule
I = scipy.integrate.simpson (data_y, x=data_x)

# printing result of numerical integration
print (f'I = {I}')
```

Execute above script.

```
% ./ai202209_s05_27.py
data_x = [0.          0.0015708   0.00314159 ... 1.56765473 1.56922553 1.57079633]
data_y = [1.00000000e+00 9.99998766e-01 9.99995065e-01 ... 3.14158749e-03
 1.57079568e-03 6.12323400e-17]
I = 1.0000000000000338
```

9 Solving differential equation

9.1 Solving a simple differential equation

Solve following differential equation numerically.

$$\frac{dy}{dx} = -0.1y \tag{7}$$

The initial condition is given by $y(x = 0) = 100$. Here is an example.

Python Code 29: ai202209_s05_28.py

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

#
# Time-stamp: <2022/10/17 00:25:08 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy
import scipy.integrate

# importing matplotlib module
```

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

# output image file
file_output = 'ai202209_s05_28.png'

# coefficient
k = 0.1

# initial condition
y_0 = 100.0

# equation to solve
def dydx (t, y):
    # dy/dx = -ky
    dy = -k * y
    # returning value
    return dy

# x values
output_x = numpy.linspace (0.0, 50.0, 5001)

# solving differential equation using Runge-Kutta method
solution = scipy.integrate.solve_ivp (dydx, [0.0, 50.0], [y_0], \
                                      method='RK45', dense_output=True, \
                                      t_eval=output_x, \
                                      rtol=10**-6, atol=10**-9)

# x and y
numerical_x = solution.t
numerical_y = solution.y[0]

# printing solution
print (f'{solution}')

# analytical solution
analytical_x = output_x
analytical_y = y_0 * numpy.exp (-k * analytical_x)

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

# axes
ax.set_title (r'$dy/dx = -ky$')
ax.set_xlabel (r'$x$')
ax.set_ylabel (r'$y$')

# plotting data
ax.plot (numerical_x, numerical_y, linestyle='--', linewidth=3.0, \
         color='blue', label='numerical solution', zorder=0.2)
ax.plot (analytical_x, analytical_y, linestyle='-', linewidth=5.0, \
         color='red', label='analytical solution', zorder=0.1)
ax.legend ()

# writing figure to file
fig.savefig (file_output, dpi=225)
```

Execute above script.

```
% ./ai202209_s05_28.py
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 146
     njev: 0
      nlu: 0
      sol: <scipy.integrate._ivp.common.OdeSolution object at 0x7ae3d85f1e60>
   status: 0
  success: True
        t: array([0.000e+00, 1.000e-02, 2.000e-02, ..., 4.998e+01, 4.999e+01,
       5.000e+01])
 t_events: None
        y: array([[100.        ,  99.90004998,  99.80019987, ...,   0.67514453,
         0.67446973,   0.67379559]])
 y_events: None
```

Display PNG file. (Fig. 18)

```
% feh -dF ai202209_s05_28.png
```

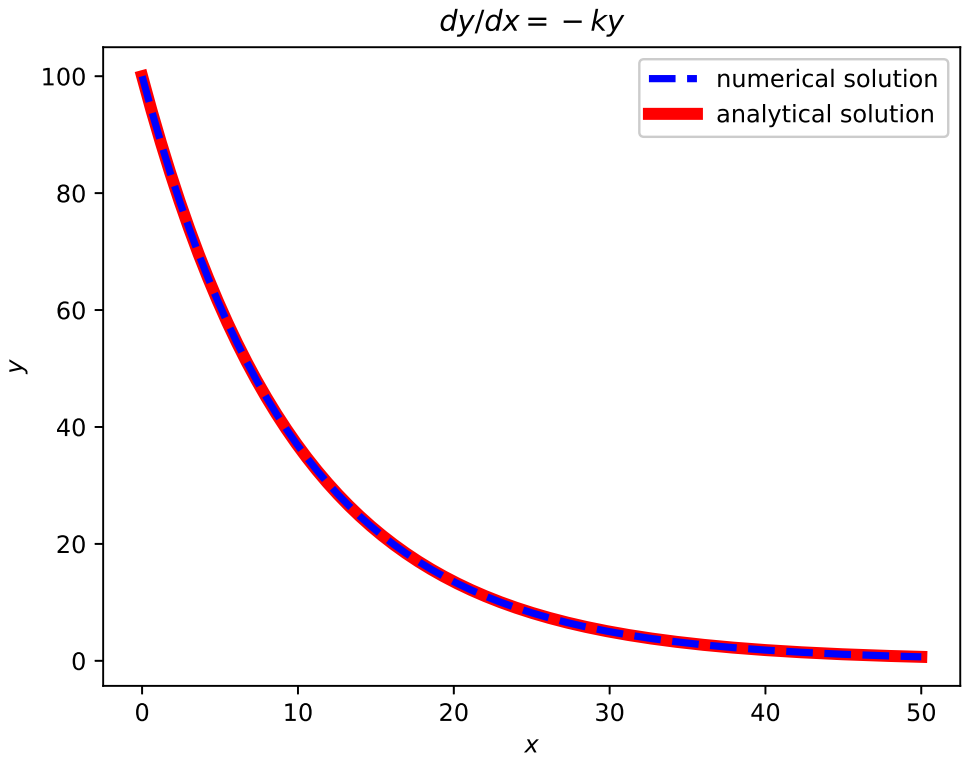

Figure 18: The numerical solution of differential equation $\frac{dy}{dx} = -0.1y$, and comparison with analytical solution.

Try following practice.

Practice 05-23

Solve your favourite first order differential equation numerically and compare the result with analytical solution.

9.2 Solving a simple differential equation

Solve the equation of motion numerically. The initial condition is given by following.

$$x(t=0) = 1.0 \quad (8)$$
$$y(t=0) = 0.0 \quad (9)$$
$$v_x(t=0) = 0.0 \quad (10)$$
$$v_y(t=0) = 1.0 \quad (11)$$
$$(12)$$

Here is an example.

Python Code 30: ai202209_s05_29.py

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

#
# Time-stamp: <2022/10/17 00:30:44 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.integrate
import scipy.stats

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

#
# initial conditions
#

# mass of the star (unit: solar mass)
M   = 1.0

# initial position of the planet (unit: astronomical unit)
qx0 = 1.0
qy0 = 0.0

# initial velocity of the planet (unit: astronomical unit per year)
vx0 = 0.0
vy0 = 1.0

# printing initial conditions
print (f'initial conditions for calculation of planetary motion:')
print (f'  qx0 = {qx0:8.3f} [au]')
print (f'  qy0 = {qy0:8.3f} [au]')
print (f'  vx0 = {vx0:8.3f} [au/year]')
print (f'  vx0 = {vy0:8.3f} [au/year]')
```

```
#
# other parameters
#

# gravitational constant
GM = 4.0 * numpy.pi**2

# time step in year
dt = 0.01

# end time of simulation in year
time_end = 10.0

# output image file
file_output = 'ai202209_s05_29.png'

#
# equation of motion
#
def eqmo (t, y):
    # initialisation of dy
    dy = numpy.zeros_like (y)
    # r^3 (cube of distance between star and planet)
    r_cubed = ( y[0]**2 + y[2]**2 )**1.5
    # Runge-Kutta method
    dy[0] = y[1]
    dy[1] = -GM * y[0] / r_cubed
    dy[2] = y[3]
    dy[3] = -GM * y[2] / r_cubed
    # returning dy
    return dy

# number of time steps of simulation
n_step = int (time_end / dt) + 1

# times to calculate position and velocity of planet
times = numpy.linspace (0.0, time_end, n_step)

# initial values
y_init = (qx0, vx0 * numpy.sqrt (GM), qy0, vy0 * numpy.sqrt (GM))

# orbital integration of planet
solution = scipy.integrate.solve_ivp (eqmo, [0.0, time_end], y_init, \
                                      t_eval=times, dense_output=True, \
                                      rtol=10**-6, atol=10**-9)

# results (positions and velocities of planet)
qx = solution.y[0]
qy = solution.y[2]
vx = solution.y[1]
xy = solution.y[3]

# finding maximum and minimum values for plotting
qx_max = scipy.stats.tmax (qx)
qy_max = scipy.stats.tmax (qy)
qx_min = scipy.stats.tmin (qx)
qy_min = scipy.stats.tmin (qy)
maxmin = sorted ([abs (qx_max), abs (qy_max), abs (qx_min), abs (qy_min)])
x_max  = maxmin[-1] * +1.3
```

```
x_min  = maxmin[-1] * -1.3
y_max  = maxmin[-1] * +1.3
y_min  = maxmin[-1] * -1.3

#
# plotting using Matplotlib
#

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

# axes
ax.set_title ('Trajectory of Planetary Motion')
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Y [au]')
ax.set_xlim (x_min, x_max)
ax.set_ylim (y_min, y_max)
ax.set_aspect ('equal')

# plotting the position of the star
ax.plot (0.0, 0.0, linestyle='None', marker='o', markersize=10.0, \
         color='yellow', label='star')

# plotting the trajectory of orbital motion of planet
ax.plot (qx, qy, linestyle='-', linewidth=3.0, color='blue', label='planet')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_29.py
initial conditions for calculation of planetary motion:
  qx0 =    1.000 [au]
  qy0 =    0.000 [au]
  vx0 =    0.000 [au/year]
  vx0 =    1.000 [au/year]
```

Display PNG file. (Fig. 19)

```
% feh -dF ai202209_s05_29.png
```

Here is one more example for following initial condition.

$$x(t=0) = 1.0 \quad (13)$$
$$y(t=0) = 0.0 \quad (14)$$
$$v_x(t=0) = 0.0 \quad (15)$$
$$v_y(t=0) = 1.2 \quad (16)$$
$$(17)$$

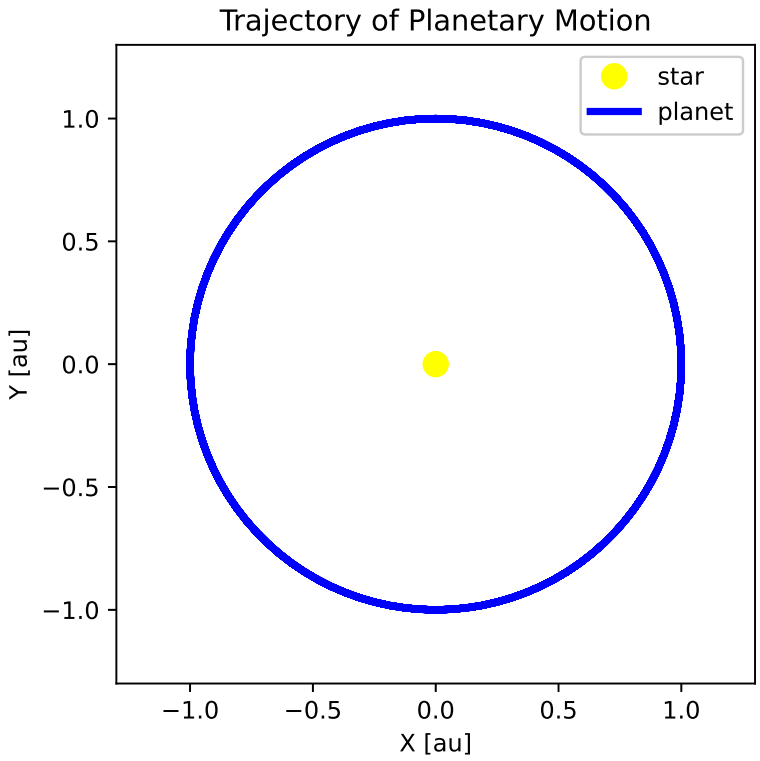

Figure 19: The trajectory of planetary motion.

Python Code 31: ai202209_s05_30.py

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

#
# Time-stamp: <2022/10/17 00:32:56 (CST) daisuke>
#

# importing argparse module
import argparse

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.integrate
import scipy.stats

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

#
# initial conditions
#
```

```
# mass of the star (unit: solar mass)
M   = 1.0

# initial position of the planet (unit: astronomical unit)
qx0 = 1.0
qy0 = 0.0

# initial velocity of the planet (unit: astronomical unit per year)
vx0 = 0.0
vy0 = 1.2

# printing initial conditions
print (f'initial conditions for calculation of planetary motion:')
print (f'  qx0 = {qx0:8.3f} [au]')
print (f'  qy0 = {qy0:8.3f} [au]')
print (f'  vx0 = {vx0:8.3f} [au/year]')
print (f'  vx0 = {vy0:8.3f} [au/year]')

#
# other parameters
#

# gravitational constant
GM = 4.0 * numpy.pi**2

# time step in year
dt = 0.01

# end time of simulation in year
time_end = 10.0

# output image file
file_output = 'ai202209_s05_30.png'

#
# equation of motion
#
def eqmo (t, y):
    # initialisation of dy
    dy = numpy.zeros_like (y)
    # r^3 (cube of distance between star and planet)
    r_cubed = ( y[0]**2 + y[2]**2 )**1.5
    # Runge-Kutta method
    dy[0] = y[1]
    dy[1] = -GM * y[0] / r_cubed
    dy[2] = y[3]
    dy[3] = -GM * y[2] / r_cubed
    # returning dy
    return dy

# number of time steps of simulation
n_step = int (time_end / dt) + 1

# times to calculate position and velocity of planet
times = numpy.linspace (0.0, time_end, n_step)

# initial values
y_init = (qx0, vx0 * numpy.sqrt (GM), qy0, vy0 * numpy.sqrt (GM))
```

```
# orbital integration of planet
solution = scipy.integrate.solve_ivp (eqmo, [0.0, time_end], y_init, \
                                      t_eval=times, dense_output=True, \
                                      rtol=10**-6, atol=10**-9)

# results (positions and velocities of planet)
qx = solution.y[0]
qy = solution.y[2]
vx = solution.y[1]
xy = solution.y[3]

# finding maximum and minimum values for plotting
qx_max = scipy.stats.tmax (qx)
qy_max = scipy.stats.tmax (qy)
qx_min = scipy.stats.tmin (qx)
qy_min = scipy.stats.tmin (qy)
maxmin = sorted ([abs (qx_max), abs (qy_max), abs (qx_min), abs (qy_min)])
x_max  = maxmin[-1] * +1.3
x_min  = maxmin[-1] * -1.3
y_max  = maxmin[-1] * +1.3
y_min  = maxmin[-1] * -1.3

#
# plotting using Matplotlib
#

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

# axes
ax.set_title ('Trajectory of Planetary Motion')
ax.set_xlabel ('X [au]')
ax.set_ylabel ('Y [au]')
ax.set_xlim (x_min, x_max)
ax.set_ylim (y_min, y_max)
ax.set_aspect ('equal')

# plotting the position of the star
ax.plot (0.0, 0.0, linestyle='None', marker='o', markersize=10.0, \
         color='yellow', label='star')

# plotting the trajectory of orbital motion of planet
ax.plot (qx, qy, linestyle='-', linewidth=3.0, color='blue', label='planet')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_30.py
initial conditions for calculation of planetary motion:
  qx0 =    1.000 [au]
  qy0 =    0.000 [au]
  vx0 =    0.000 [au/year]
```

```
  vx0 =     1.200 [au/year]
```

Display PNG file. (Fig. 20)

```
% feh -dF ai202209_s05_30.png
```

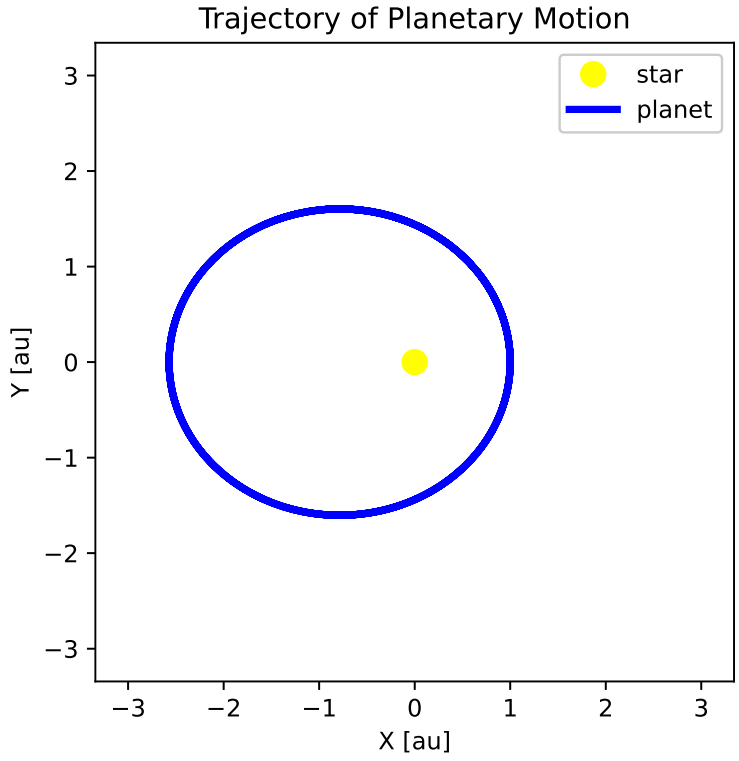

Figure 20: The trajectory of planetary motion.

Try following practice.

Practice 05-24

Solve your favourite second order differential equation numerically.

Here is an example for making an animation of planetary motion for following initial condition.

$$x(t=0) = 1.0 \quad (18)$$
$$y(t=0) = 0.0 \quad (19)$$
$$v_x(t=0) = 0.0 \quad (20)$$
$$v_y(t=0) = 1.3 \quad (21)$$
$$(22)$$

Python Code 32: ai202209_s05_31.py

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

#
```

```
# Time-stamp: <2022/10/16 13:27:14 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.integrate
import scipy.stats

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

#
# initial conditions
#

# mass of the star (unit: solar mass)
M   = 1.0

# initial position of the planet (unit: astronomical unit)
qx0 = 1.0
qy0 = 0.0

# initial velocity of the planet (unit: astronomical unit per year)
vx0 = 0.0
vy0 = 1.3

# printing initial conditions
print (f'initial conditions for calculation of planetary motion:')
print (f'  qx0 = {qx0:8.3f} [au]')
print (f'  qy0 = {qy0:8.3f} [au]')
print (f'  vx0 = {vx0:8.3f} [au/year]')
print (f'  vx0 = {vy0:8.3f} [au/year]')

#
# other parameters
#

# gravitational constant
GM = 4.0 * numpy.pi**2

# time step in year
dt = 0.01

# end time of simulation in year
time_end = 15.0

# output image file
file_output = 'ai202209_s05_31.mp4'

#
# equation of motion
#
def eqmo (t, y):
    # initialisation of dy
    dy = numpy.zeros_like (y)
```

```
    # r^3 (cube of distance between star and planet)
    r_cubed = ( y[0]**2 + y[2]**2 )**1.5
    # Runge-Kutta method
    dy[0] = y[1]
    dy[1] = -GM * y[0] / r_cubed
    dy[2] = y[3]
    dy[3] = -GM * y[2] / r_cubed
    # returning dy
    return dy

# number of time steps of simulation
n_step = int (time_end / dt) + 1

# times to calculate position and velocity of planet
times = numpy.linspace (0.0, time_end, n_step)

# initial values
y_init = (qx0, vx0 * numpy.sqrt (GM), qy0, vy0 * numpy.sqrt (GM))

# orbital integration of planet
solution = scipy.integrate.solve_ivp (eqmo, [0.0, time_end], y_init, \
                                      t_eval=times, dense_output=True, \
                                      rtol=10**-6, atol=10**-9)

# results (positions and velocities of planet)
qx = solution.y[0]
qy = solution.y[2]
vx = solution.y[1]
xy = solution.y[3]

# finding maximum and minimum values for plotting
qx_max = scipy.stats.tmax (qx)
qy_max = scipy.stats.tmax (qy)
qx_min = scipy.stats.tmin (qx)
qy_min = scipy.stats.tmin (qy)
maxmin = sorted ([abs (qx_max), abs (qy_max), abs (qx_min), abs (qy_min)])
x_max  = maxmin[-1] * +1.3
x_min  = maxmin[-1] * -1.3
y_max  = maxmin[-1] * +1.3
y_min  = maxmin[-1] * -1.3

#
# making animation using Matplotlib
#

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

# an empty list for animation
list_frame = []

for i in range (n_step):
    # empty list for objects on a plot
    list_obj = []

    # axes
    ax.set_title ('Trajectory of Planetary Motion')
```

```
    ax.set_xlabel ('X [au]')
    ax.set_ylabel ('Y [au]')
    ax.set_xlim (x_min, x_max)
    ax.set_ylim (y_min, y_max)
    ax.set_aspect ('equal')

    # plotting the position of the star
    star, = ax.plot (0.0, 0.0, linestyle='None', marker='o', \
                     markersize=10.0, color='yellow', label='star')
    list_obj.append (star)

    # plotting the trajectory of orbital motion of planet
    orb, = ax.plot (qx, qy, linestyle='-', linewidth=1.0, color='cyan', \
                    label='orbit of planet')
    list_obj.append (orb)
    planet, = ax.plot (qx[i], qy[i], linestyle='None', marker='o', \
                       markersize=5.0, color='blue', label='planet')
    list_obj.append (planet)

    # texts
    text_time = "Time: %8.2f year" % (i * dt)
    text_initial = "Initial conditions"
    text_mass    = "mass of star = %5.2f solar mass" % (M)
    text_iq      = "(qx0, qy0) = (%4.2f au, %4.2f au)" % (qx0, qy0)
    text_iv      = "(vx0, vy0) = (%4.2f au/yr, %4.2f au/yr)" % (vx0, vy0)
    text_1 = ax.text (0.03, 0.95, text_time, transform=ax.transAxes)
    list_obj.append (text_1)
    text_2 = ax.text (0.03, 0.18, text_initial, transform=ax.transAxes)
    list_obj.append (text_2)
    text_3 = ax.text (0.05, 0.13, text_mass, transform=ax.transAxes)
    list_obj.append (text_3)
    text_4 = ax.text (0.05, 0.08, text_iq, transform=ax.transAxes)
    list_obj.append (text_4)
    text_5 = ax.text (0.05, 0.03, text_iv, transform=ax.transAxes)
    list_obj.append (text_5)

    # appending plot to animation
    list_frame.append (list_obj)

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

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

Execute above script.

```
% ./ai202209_s05_31.py
initial conditions for calculation of planetary motion:
  qx0 =    1.000 [au]
  qy0 =    0.000 [au]
  vx0 =    0.000 [au/year]
  vx0 =    1.300 [au/year]
```

Play MP4 movie file.

```
% mplayer ai202209_s05_31.mp4
```

Try following practice.

Practice 05-25

Solve the equation of motion with your favourite initial condition and make an animation of the orbital motion of a planet around a star.

10 Optimisation problem

10.1 Finding minimum for a single variable function

Plot following function.

$$f(x) = 2(x-3)^2 + 4 \tag{23}$$

Here is an example.

Python Code 33: ai202209_s05_32.py

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

#
# Time-stamp: <2022/10/17 00:47:10 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output file name
file_output = 'ai202209_s05_32.png'

# function of a curve
def curve (x):
    # coefficients
    a = 2.0
    b = 3.0
    c = 4.0
    # curve
    y = a * (x - b)**2 + c
    # returning y-value
    return y

# data to plot
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = curve (data_x)

#
# making plot using Matplotlib
#

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

# axes
```

```
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='blue', \
         label='$f(x)=2(x-3)^2+4$')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_32.py
```

Display PNG file. (Fig. 21)

```
% feh -dF ai202209_s05_32.png
```

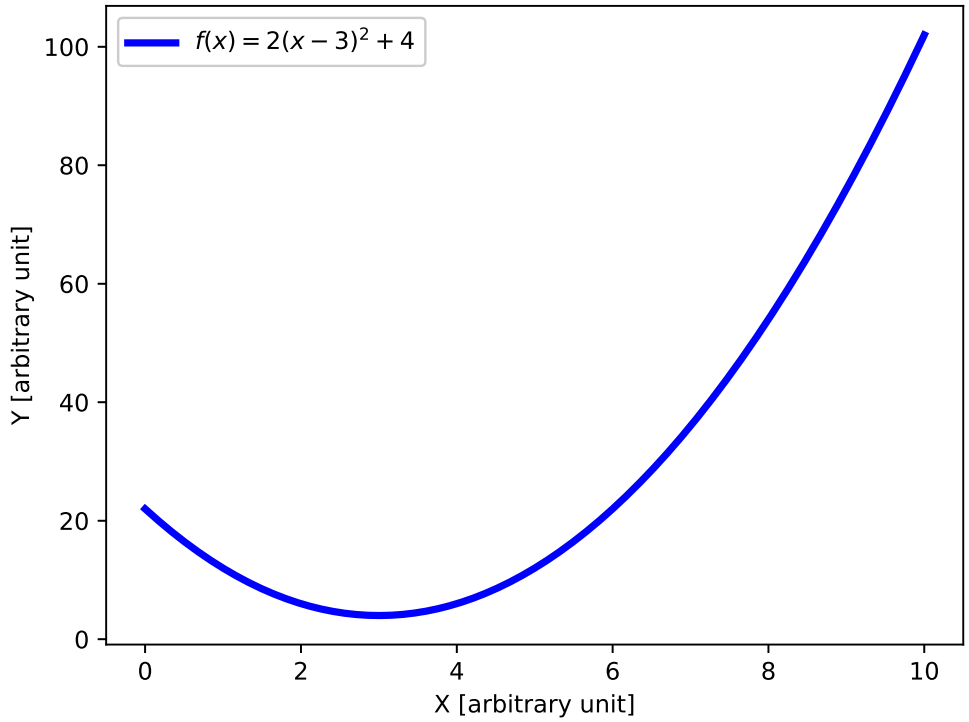

Figure 21: The function $f(x) = 2(x-3)^2 + 4$.

Find the minimum for following function.

$$f(x) = 2(x-3)^2 + 4 \tag{24}$$

Here is an example.

Python Code 34: ai202209_s05_33.py

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

#
# Time-stamp: <2022/10/17 00:51:15 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize

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

# output file name
file_output = 'ai202209_s05_33.png'

# function of a curve
def curve (x):
    # coefficients
    a = 2.0
    b = 3.0
    c = 4.0
    # curve
    y = a * (x - b)**2 + c
    # returning y-value
    return y

# finding minimum
minimum = scipy.optimize.minimize_scalar (curve, method='brent')

# printing minimum value
print (f'minimum: y={minimum.fun} at x={minimum.x}')

# data to plot
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = curve (data_x)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='blue', \
         label='$f(x)=2(x-3)^2+4$')
ax.plot (minimum.x, minimum.fun, linestyle='None', marker='o', \
         markersize=5.0, color='red', label='minimum')
```

```
# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_33.py
minimum: y=4.0 at x=3.0
```

Display PNG file. (Fig. 22)

```
% feh -dF ai202209_s05_33.png
```

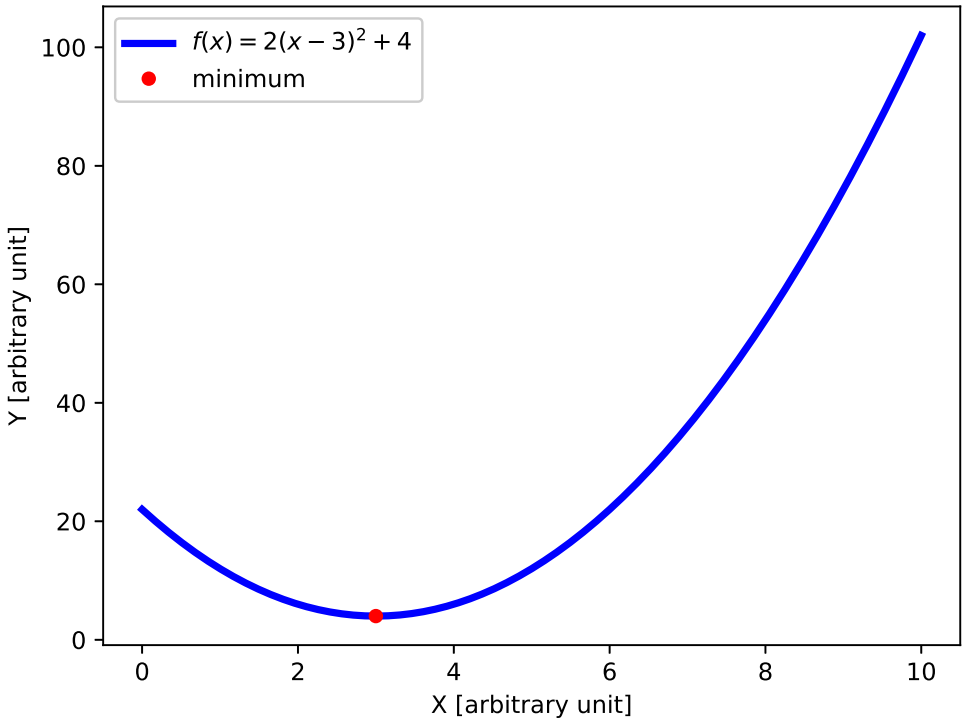

Figure 22: The minimum of the function $f(x) = 2(x-3)^2 + 4$.

Try following practice.

Practice 05-26

Find the minimum for your favourite single variable function.

10.2 Finding minimum within a given range

Plot following function.

$$f(x) = 3\sin(2\pi x + \pi) + 5 \tag{25}$$

Here is an example.

Python Code 35: ai202209_s05_34.py

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

#
# Time-stamp: <2022/10/17 01:00:02 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output file name
file_output = 'ai202209_s05_34.png'

# function of a curve
def curve (x):
    # coefficients
    a = 3.0
    b = 2.0 * numpy.pi
    c = numpy.pi
    d = 5.0
    # curve
    y = a * numpy.sin (b * x + c) + d
    # returning y-value
    return y

# data to plot
data_x = numpy.linspace (0.0, 1.0, 1001)
data_y = curve (data_x)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='blue', \
         label='$f(x)=3 \sin (2 \pi x + \pi) + 5$')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_34.py
```

Display PNG file. (Fig. 23)

```
% feh -dF ai202209_s05_34.png
```

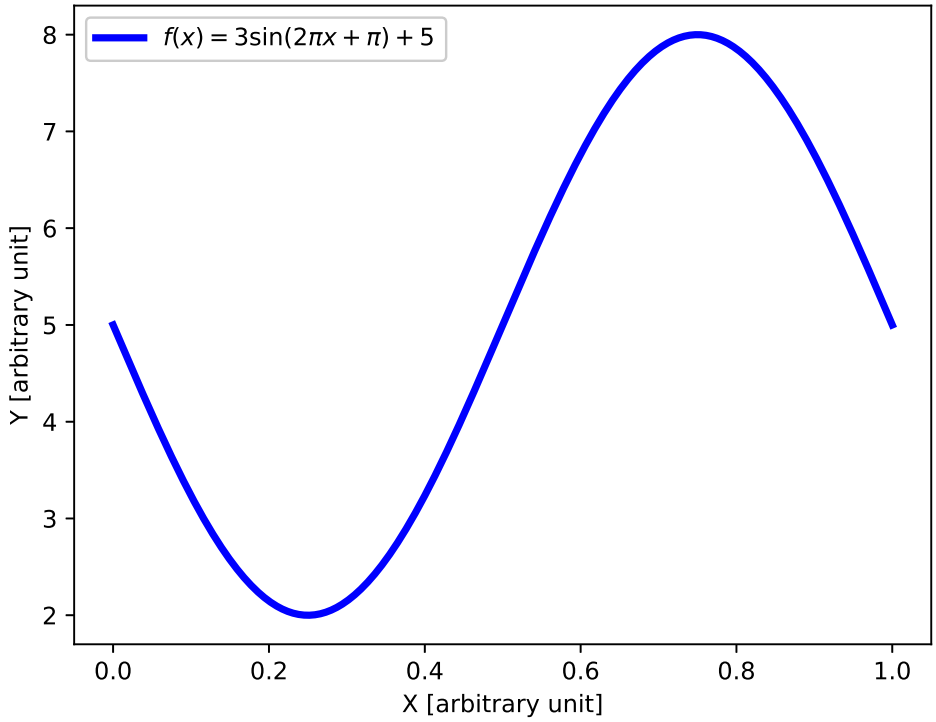

Figure 23: The function $f(x) = 3\sin(2\pi x + \pi) + 5$.

Find the minimum within the range between 0 and 1 for following function.

$$f(x) = 3\sin(2\pi x + \pi) + 5 \tag{26}$$

Here is an example.

Python Code 36: ai202209_s05_35.py

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

#
# Time-stamp: <2022/10/17 01:00:21 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize

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

# output file name
```

```
file_output = 'ai202209_s05_35.png'

# function of a curve
def curve (x):
    # coefficients
    a = 3.0
    b = 2.0 * numpy.pi
    c = numpy.pi
    d = 5.0
    # curve
    y = a * numpy.sin (b * x + c) + d
    # returning y-value
    return y

# finding minimum
minimum = scipy.optimize.minimize_scalar (curve, method='bounded', bounds=[0,1])

# printing minimum value
print (f'minimum: y={minimum.fun} at x={minimum.x}')

# data to plot
data_x = numpy.linspace (0.0, 1.0, 1001)
data_y = curve (data_x)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='-', linewidth=3.0, color='blue', \
         label='$f(x)=3 \sin (2 \pi x + \pi) + 5$')
ax.plot (minimum.x, minimum.fun, linestyle='None', marker='o', \
         markersize=5.0, color='red', label='minimum')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_35.py
minimum: y=2.0000000000662 at x=0.24999966564484488
```

Display PNG file. (Fig. 24)

```
% feh -dF ai202209_s05_35.png
```

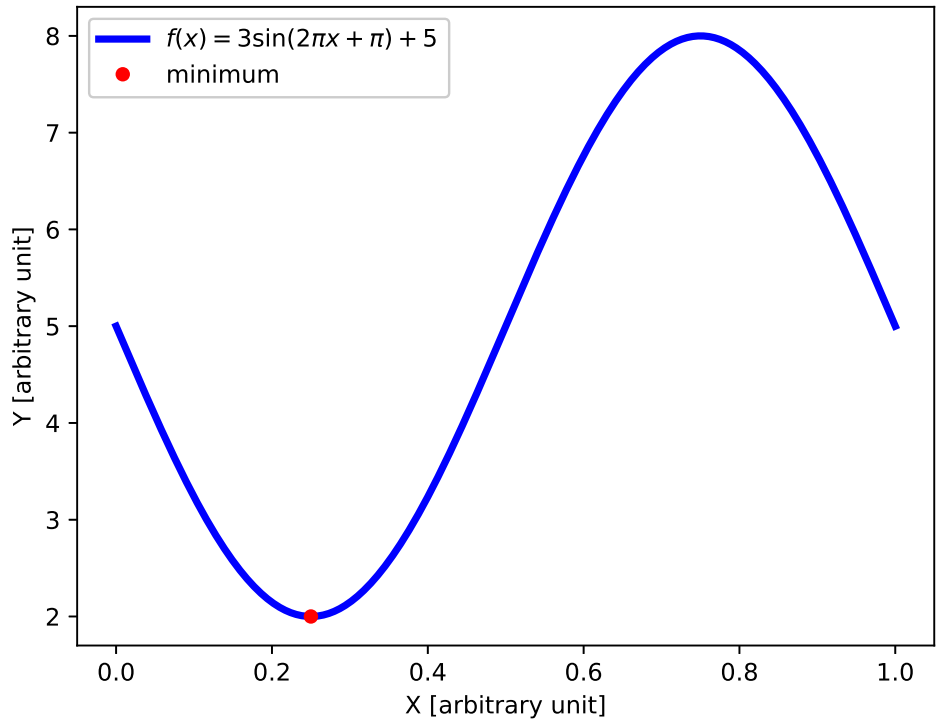

Figure 24: The minimum of the function $f(x) = 3\sin(2\pi x + \pi) + 5$ between 0 and 1.

Try following practice.

Practice 05-27

Set a range and find the minimum for your favourite single variable function within the range you have set.

10.3 Finding minimum for multiple variable function

Plot following function.

$$f(x, y) = -3\exp[-(x-5)^2]\exp[-(y-6)^2] + 4 \tag{27}$$

Here is an example.

Python Code 37: ai202209_s05_36.py

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

#
# Time-stamp: <2022/10/17 01:04:04 (CST) daisuke>
#

# importing numpy module
import numpy

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

# output file name
```

```
file_output = 'ai202209_s05_36.png'

# coefficients
a = -3.0
b = 5.0
c = 6.0
d = 4.0

# data to plot
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = numpy.linspace (0.0, 10.0, 1001)
data_xx, data_yy = numpy.meshgrid (data_x, data_y)
data_zz = a * numpy.exp (-(data_xx - b)**2) * numpy.exp (-(data_yy - c)**2) + d

#
# making plot using Matplotlib
#

# making objects "fig" and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111, projection='3d')

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')
ax.set_zlabel ('Z [arbitrary unit]')
ax.set_xlim (0.0, 10.0)
ax.set_ylim (0.0, 10.0)
ax.set_zlim (0.0, 4.5)

# plotting data
ax.plot_surface (data_xx, data_yy, data_zz, \
                 label='$f(x,y) = -3 \exp (x-5)^2 \exp (y-6)^2 + 4$')
ax.contour (data_xx, data_yy, data_zz, zdir='z', offset=0.0)

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

Execute above script.

```
% ./ai202209_s05_36.py
```

Display PNG file. (Fig. 25)

```
% feh -dF ai202209_s05_36.png
```

Find the minimum following function.

$$f(x,y) = -3\exp[-(x-5)^2]\exp[-(y-6)^2] + 4 \tag{28}$$

Here is an example.

Python Code 38: ai202209_s05_37.py

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

#
```

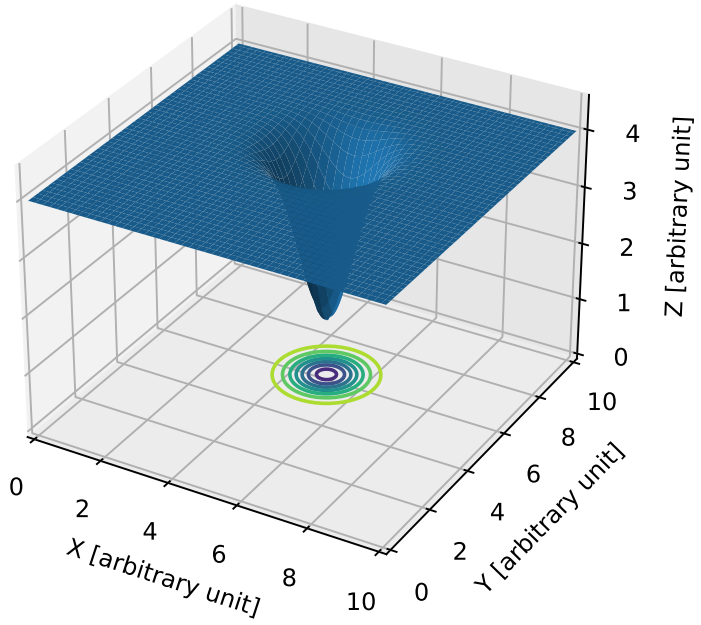

Figure 25: The function $f(x, y) = -3 \exp[-(x-5)^2] \exp[-(y-6)^2] + 4$.

```
# Time-stamp: <2022/10/17 01:06:14 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize

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

# output file name
file_output = 'ai202209_s05_37.png'

# coefficients
a = -3.0
b = 5.0
c = 6.0
d = 4.0

# function of a curve
def surface (x):
    # curve
    z = a * numpy.exp ( -(x[0] - b)**2 ) * numpy.exp ( -(x[1] - c)**2 ) + d
    # returning y-value
    return z
```

```
# finding minimum
minimum = scipy.optimize.minimize (surface, x0=(8.0, 8.0), method='Nelder-Mead')

# printing minimum value
print (f'minimum: z={minimum.fun} at (x, y) = ({minimum.x[0]}, {minimum.x[1]})')

# data to plot
data_x = numpy.linspace (0.0, 10.0, 1001)
data_y = numpy.linspace (0.0, 10.0, 1001)
data_xx, data_yy = numpy.meshgrid (data_x, data_y)
data_zz = a * numpy.exp (-(data_xx - b)**2) * numpy.exp (-(data_yy - c)**2) + d

#
# making plot using Matplotlib
#

# making objects "fig" and "ax"
fig    = matplotlib.figure.Figure ()
canvas = matplotlib.backends.backend_agg.FigureCanvasAgg (fig)
ax     = fig.add_subplot (111, projection='3d')

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')
ax.set_zlabel ('Z [arbitrary unit]')
ax.set_xlim (0.0, 10.0)
ax.set_ylim (0.0, 10.0)
ax.set_zlim (0.0, 4.5)

# plotting data
ax.plot_surface (data_xx, data_yy, data_zz, \
                 label='$f(x,y) = -3 \exp (x-5)^2 \exp (y-6)^2 + 4$')
ax.contour (data_xx, data_yy, data_zz, zdir='z', offset=0.0)
ax.plot3D (minimum.x[0], minimum.x[1], minimum.fun, linestyle='None', \
           marker='o', markersize=5.0, color='red', label='minimum')

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

Execute above script.

```
% ./ai202209_s05_37.py
minimum: z=1.0000000052721698 at (x, y) = (5.000007258438867, 6.000041288072987)
```

Display PNG file. (Fig. 26)

```
% feh -dF ai202209_s05_37.png
```

Try following practice.

Practice 05-28

Find the minimum for your favourite multiple variable function.

11 Least-squares method

Try least-squares method using SciPy.

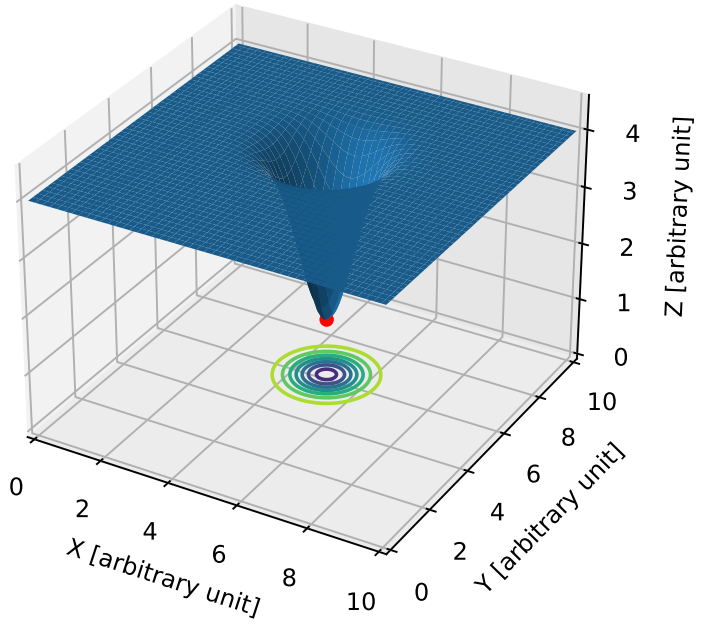

Figure 26: The minimum of the function $f(x, y) = -3\exp[-(x-5)^2]\exp[-(y-6)^2] + 4$.

11.1 Fitting to straight line using `least_squares`

Generate synthetic data for least-squares method. Here is an example.

Python Code 39: ai202209_s05_38.py

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

#
# Time-stamp: <2022/10/16 17:52:19 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.stats

# output file name
file_output = 'ai202209_s05_38.data'

# generating random numbers
err = scipy.stats.norm.rvs (loc=0.0, scale=0.5, size=16)

# function for a line
def line (x):
    # coefficients
    a = 3.0
    b = 10.0
    # line
```

```
    y = a * x + b
    # returning y
    return y

# synthetic data for least-squares method
data_x = numpy.linspace (0.0, 15.0, 16)
data_y = line (data_x) + err

# opening file for writing
with open (file_output, 'w') as fh:
    # writing generated synthetic data into file
    for i in range (len (data_x)):
        fh.write (f'{data_x[i]:8.3f}    {data_y[i]:8.3f}\n')
```

Execute above script.

```
% ./ai202209_s05_38.py
```

Show generated data.

```
% cat ai202209_s05_38.data
   0.000       10.259
   1.000       13.031
   2.000       15.754
   3.000       18.062
   4.000       22.173
   5.000       24.816
   6.000       28.002
   7.000       31.216
   8.000       33.866
   9.000       36.518
  10.000       39.714
  11.000       43.211
  12.000       46.699
  13.000       48.754
  14.000       52.318
  15.000       55.137
```

Plot the synthetic data using Matplotlib. Here is an example.

Python Code 40: ai202209_s05_39.py

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

#
# Time-stamp: <2022/10/17 01:14:31 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

```
# input file name
file_input = 'ai202209_s05_38.data'

# output file name
file_output = 'ai202209_s05_39.png'

# making empty numpy arrays
data_x = numpy.array ([])
data_y = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x = numpy.append (data_x, x)
        data_y = numpy.append (data_y, y)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}) = ({data_x[i]:8.3f}, {data_y[i]:8.3f})')

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='synthetic data for least-squares method')

# legend
ax.legend ()
```

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

Execute above script.

```
% ./ai202209_s05_39.py
(x_00, y_00) = (   0.000,   10.259)
(x_01, y_01) = (   1.000,   13.031)
(x_02, y_02) = (   2.000,   15.754)
(x_03, y_03) = (   3.000,   18.062)
(x_04, y_04) = (   4.000,   22.173)
(x_05, y_05) = (   5.000,   24.816)
(x_06, y_06) = (   6.000,   28.002)
(x_07, y_07) = (   7.000,   31.216)
(x_08, y_08) = (   8.000,   33.866)
(x_09, y_09) = (   9.000,   36.518)
(x_10, y_10) = (  10.000,   39.714)
(x_11, y_11) = (  11.000,   43.211)
(x_12, y_12) = (  12.000,   46.699)
(x_13, y_13) = (  13.000,   48.754)
(x_14, y_14) = (  14.000,   52.318)
(x_15, y_15) = (  15.000,   55.137)
```

Display PNG file. (Fig. 27)

```
% feh -dF ai202209_s05_39.png
```

Try least-squares method using SciPy. Here is an example.

Python Code 41: ai202209_s05_40.py

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

#
# Time-stamp: <2022/10/17 01:19:17 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_38.data'

# output file name
file_output = 'ai202209_s05_40.png'

# making empty numpy arrays
```

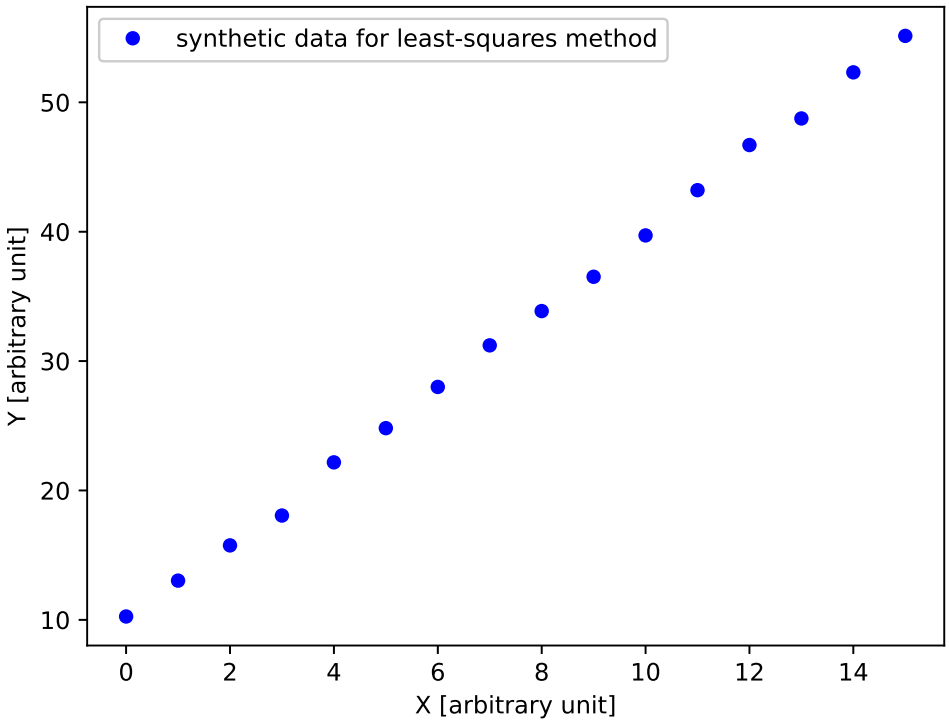

Figure 27: A plot of synthetic data for least-squares method.

```
data_x = numpy.array ([])
data_y = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x = numpy.append (data_x, x)
        data_y = numpy.append (data_y, y)
```

```
# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}) = ({data_x[i]:8.3f}, {data_y[i]:8.3f})')

# a function for straight line
def line (x, a, b):
    # line
    y = a * x + b
    # returning y
    return y

# a function to calculate residuals
def residual (param, x, y):
    # calculation of residual
    residue = param[0] * x + param[1] - y
    # returning residual
    return residue

# initial guess of coefficients
param0 = [1.0, 1.0]

# fitting
fit_lsq = scipy.optimize.least_squares (residual, param0, \
                                        args=(data_x, data_y) )

# printing result of fitting
print (f'{fit_lsq}')

# degree of freedom
dof = len (data_x) - len (fit_lsq.x)
print (f'dof = {dof}')

# reduced chi-squared
reduced_chi2 = (fit_lsq.fun**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# Jacobian
J = fit_lsq.jac
print (f'Jacobian:\n{J}')

# transpose matrix of Jacobian
Jt =J.T

# calculation of J^T J
Jt_J = numpy.matmul (Jt, J)

# covariance matrix
pcov = numpy.linalg.inv (Jt_J) * reduced_chi2
print (f'covariance matrix:\n{pcov}')

# fitted a and b
a_fitted, b_fitted = fit_lsq.x
a_err, b_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)
```

```
# fitted line
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = line (fitted_x, a_fitted, b_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='synthetic data for least-squares method', \
         zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted line by least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_40.py
(x_00, y_00) = (   0.000,   10.259)
(x_01, y_01) = (   1.000,   13.031)
(x_02, y_02) = (   2.000,   15.754)
(x_03, y_03) = (   3.000,   18.062)
(x_04, y_04) = (   4.000,   22.173)
(x_05, y_05) = (   5.000,   24.816)
(x_06, y_06) = (   6.000,   28.002)
(x_07, y_07) = (   7.000,   31.216)
(x_08, y_08) = (   8.000,   33.866)
(x_09, y_09) = (   9.000,   36.518)
(x_10, y_10) = (  10.000,   39.714)
(x_11, y_11) = (  11.000,   43.211)
(x_12, y_12) = (  12.000,   46.699)
(x_13, y_13) = (  13.000,   48.754)
(x_14, y_14) = (  14.000,   52.318)
(x_15, y_15) = (  15.000,   55.137)
 active_mask: array([0., 0.])
        cost: 0.9970849985294089
         fun: array([-0.44927205, -0.19981911,  0.09863383,  0.81208677, -0.27746029
        0.10099265, -0.06355441, -0.25610147,  0.11535147,  0.48480441,
        0.31025735, -0.16528971, -0.63183677,  0.33461618, -0.20793088,
       -0.00547794])
        grad: array([5.52978523e-08, 3.62188368e-08])
         jac: array([[ 0.        ,  1.        ],
       [ 0.99999999,  1.        ],
```

```
       [ 2.00000002,  1.        ],
       [ 3.00000005,  1.00000002],
       [ 4.00000004,  1.00000002],
       [ 5.00000003,  1.00000002],
       [ 6.0000001 ,  1.00000002],
       [ 7.00000009,  1.00000002],
       [ 8.00000008,  1.00000002],
       [ 9.00000007,  1.00000002],
       [10.00000006,  1.00000002],
       [11.00000005,  1.00000002],
       [12.00000004,  1.00000002],
       [13.00000003,  1.00000002],
       [14.00000002,  1.00000002],
       [15.00000001,  1.00000002]])
     message: 'Both `ftol` and `xtol` termination conditions are satisfied.'
        nfev: 6
        njev: 6
  optimality: 5.5297852311064766e-08
      status: 4
     success: True
           x: array([3.02145294, 9.80972795])
dof = 14
reduced chi-squared = 0.14244071407562986
Jacobian:
[[ 0.          1.        ]
 [ 0.99999999  1.        ]
 [ 2.00000002  1.        ]
 [ 3.00000005  1.00000002]
 [ 4.00000004  1.00000002]
 [ 5.00000003  1.00000002]
 [ 6.0000001   1.00000002]
 [ 7.00000009  1.00000002]
 [ 8.00000008  1.00000002]
 [ 9.00000007  1.00000002]
 [10.00000006  1.00000002]
 [11.00000005  1.00000002]
 [12.00000004  1.00000002]
 [13.00000003  1.00000002]
 [14.00000002  1.00000002]
 [15.00000001  1.00000002]]
covariance matrix:
[[ 0.00041894 -0.00314207]
 [-0.00314207  0.0324681 ]]
a =    3.021 +/-    0.020 (   0.677%)
b =    9.810 +/-    0.180 (   1.837%)
```

Display PNG file. (Fig. 28)

```
% feh -dF ai202209_s05_40.png
```

Try following practice.

Practice 05-29

Generate a set of synthetic data and fit the data using a straight line.

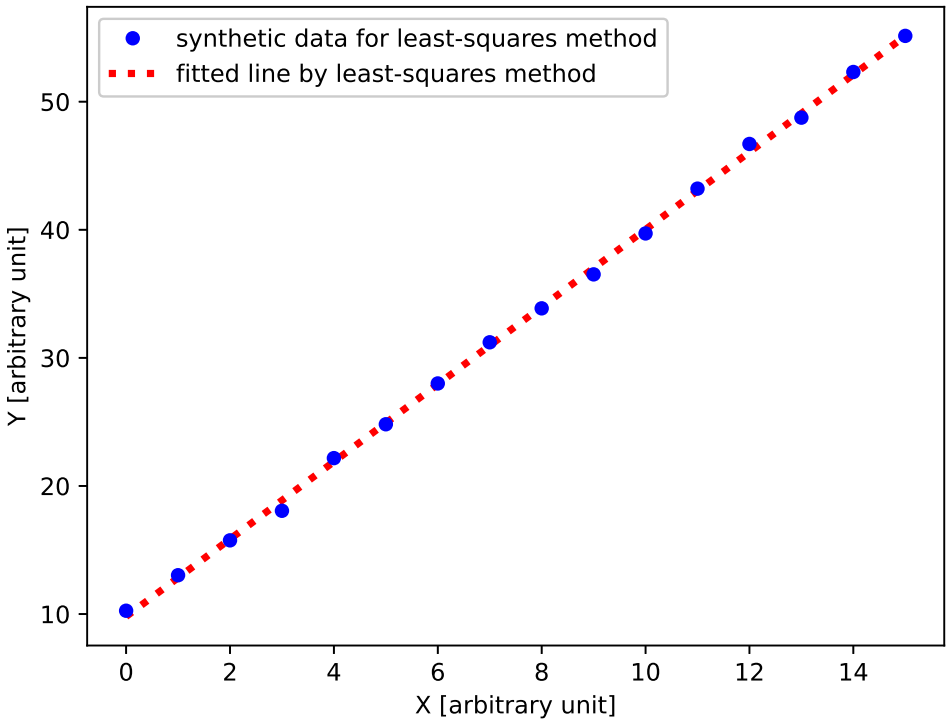

Figure 28: The result of least-squares method to fit the data to a straight line.

11.2 Weighted least-squares method using `least_squares`

Generate a set of synthetic data with errors. Here is an example.

Python Code 42: ai202209_s05_41.py

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

#
# Time-stamp: <2022/10/16 18:51:19 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.stats

# output file name
file_output = 'ai202209_s05_41.data'

# generating random numbers
err = scipy.stats.norm.rvs (loc=0.0, scale=1.0, size=32)

# function for a line
def line (x):
    # coefficients
    a = 3.0
    b = 10.0
    # line
```

```
    y = a * x + b
    # returning y
    return y

# synthetic data for least-squares method
data_x = numpy.linspace (0.0, 31.0, 32)
data_y = line (data_x) + err
for i in range (10, 20):
    data_y[i] += 10.0
    err[i]    += 12.0

# opening file for writing
with open (file_output, 'w') as fh:
    # writing generated synthetic data into file
    for i in range (len (data_x)):
        fh.write (f'{data_x[i]:8.3f}  {data_y[i]:8.3f}  {abs (err[i]):8.3f}\n')
```

Execute above script.

```
% ./ai202209_s05_41.py
```

Show generated data.

```
% cat ai202209_s05_41.data
   0.000     9.151     0.849
   1.000    12.068     0.932
   2.000    16.570     0.570
   3.000    19.502     0.502
   4.000    21.898     0.102
   5.000    25.383     0.383
   6.000    28.172     0.172
   7.000    29.869     1.131
   8.000    34.176     0.176
   9.000    37.451     0.451
  10.000    50.994    12.994
  11.000    52.674    11.674
  12.000    57.704    13.704
  13.000    58.934    11.934
  14.000    62.594    12.594
  15.000    65.536    12.536
  16.000    68.946    12.946
  17.000    71.401    12.401
  18.000    74.699    12.699
  19.000    77.839    12.839
  20.000    67.853     2.147
  21.000    72.323     0.677
  22.000    77.306     1.306
  23.000    78.390     0.610
  24.000    83.143     1.143
  25.000    86.313     1.313
  26.000    89.857     1.857
  27.000    90.437     0.563
  28.000    93.226     0.774
  29.000    95.470     1.530
  30.000    99.168     0.832
  31.000   101.345     1.655
```

Plot the synthetic data using Matplotlib. Here is an example.

Python Code 43: ai202209_s05_42.py

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

#
# Time-stamp: <2022/10/17 01:25:39 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s05_41.data'

# output file name
file_output = 'ai202209_s05_42.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
```

```
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_42.py
(x_00, y_00, err_00) = (   0.000,    9.151,    0.849)
(x_01, y_01, err_01) = (   1.000,   12.068,    0.932)
(x_02, y_02, err_02) = (   2.000,   16.570,    0.570)
(x_03, y_03, err_03) = (   3.000,   19.502,    0.502)
(x_04, y_04, err_04) = (   4.000,   21.898,    0.102)
(x_05, y_05, err_05) = (   5.000,   25.383,    0.383)
(x_06, y_06, err_06) = (   6.000,   28.172,    0.172)
(x_07, y_07, err_07) = (   7.000,   29.869,    1.131)
(x_08, y_08, err_08) = (   8.000,   34.176,    0.176)
(x_09, y_09, err_09) = (   9.000,   37.451,    0.451)
(x_10, y_10, err_10) = (  10.000,   50.994,   12.994)
(x_11, y_11, err_11) = (  11.000,   52.674,   11.674)
(x_12, y_12, err_12) = (  12.000,   57.704,   13.704)
(x_13, y_13, err_13) = (  13.000,   58.934,   11.934)
(x_14, y_14, err_14) = (  14.000,   62.594,   12.594)
(x_15, y_15, err_15) = (  15.000,   65.536,   12.536)
(x_16, y_16, err_16) = (  16.000,   68.946,   12.946)
(x_17, y_17, err_17) = (  17.000,   71.401,   12.401)
(x_18, y_18, err_18) = (  18.000,   74.699,   12.699)
(x_19, y_19, err_19) = (  19.000,   77.839,   12.839)
(x_20, y_20, err_20) = (  20.000,   67.853,    2.147)
(x_21, y_21, err_21) = (  21.000,   72.323,    0.677)
```

```
(x_22, y_22, err_22) = (  22.000,    77.306,     1.306)
(x_23, y_23, err_23) = (  23.000,    78.390,     0.610)
(x_24, y_24, err_24) = (  24.000,    83.143,     1.143)
(x_25, y_25, err_25) = (  25.000,    86.313,     1.313)
(x_26, y_26, err_26) = (  26.000,    89.857,     1.857)
(x_27, y_27, err_27) = (  27.000,    90.437,     0.563)
(x_28, y_28, err_28) = (  28.000,    93.226,     0.774)
(x_29, y_29, err_29) = (  29.000,    95.470,     1.530)
(x_30, y_30, err_30) = (  30.000,    99.168,     0.832)
(x_31, y_31, err_31) = (  31.000,   101.345,     1.655)
```

Display PNG file. (Fig. 29)

```
% feh -dF ai202209_s05_42.png
```

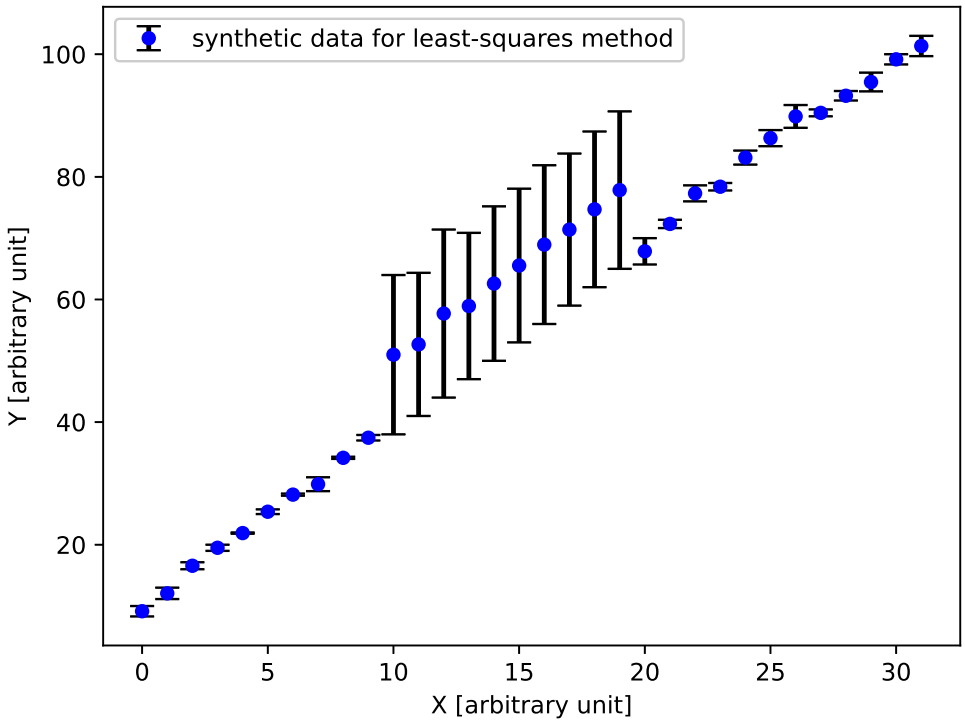

Figure 29: A plot of synthetic data for least-squares method.

Try least-squares method using SciPy. Here is an example.

Python Code 44: ai202209_s05_43.py

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

#
# Time-stamp: <2022/10/17 01:27:41 (CST) daisuke>
#

# importing sys module
import sys
```

```
# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_41.data'

# output file name
file_output = 'ai202209_s05_43.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
```

```
            f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for straight line
def line (x, a, b):
    # line
    y = a * x + b
    # returning y
    return y

# a function to calculate residuals
def residual (param, x, y):
    # calculation of residual
    residue = param[0] * x + param[1] - y
    # returning residual
    return residue

# initial guess of coefficients
param0 = [1.0, 1.0]

# fitting
fit_lsq = scipy.optimize.least_squares (residual, param0, \
                                        args=(data_x, data_y) )

# printing result of fitting
print (f'{fit_lsq}')

# degree of freedom
dof = len (data_x) - len (fit_lsq.x)
print (f'dof = {dof}')

# reduced chi-squared
reduced_chi2 = (fit_lsq.fun**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# Jacobian
J = fit_lsq.jac
print (f'Jacobian:\n{J}')

# transpose matrix of Jacobian
Jt =J.T

# calculation of J^T J
Jt_J = numpy.matmul (Jt, J)

# covariance matrix
pcov = numpy.linalg.inv (Jt_J) * reduced_chi2
print (f'covariance matrix:\n{pcov}')

# fitted a and b
a_fitted, b_fitted = fit_lsq.x
a_err, b_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted line
```

```
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = line (fitted_x, a_fitted, b_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted line by least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_43.py
(x_00, y_00, err_00) = (   0.000,    9.151,    0.849)
(x_01, y_01, err_01) = (   1.000,   12.068,    0.932)
(x_02, y_02, err_02) = (   2.000,   16.570,    0.570)
(x_03, y_03, err_03) = (   3.000,   19.502,    0.502)
(x_04, y_04, err_04) = (   4.000,   21.898,    0.102)
(x_05, y_05, err_05) = (   5.000,   25.383,    0.383)
(x_06, y_06, err_06) = (   6.000,   28.172,    0.172)
(x_07, y_07, err_07) = (   7.000,   29.869,    1.131)
(x_08, y_08, err_08) = (   8.000,   34.176,    0.176)
(x_09, y_09, err_09) = (   9.000,   37.451,    0.451)
(x_10, y_10, err_10) = (  10.000,   50.994,   12.994)
(x_11, y_11, err_11) = (  11.000,   52.674,   11.674)
(x_12, y_12, err_12) = (  12.000,   57.704,   13.704)
(x_13, y_13, err_13) = (  13.000,   58.934,   11.934)
(x_14, y_14, err_14) = (  14.000,   62.594,   12.594)
(x_15, y_15, err_15) = (  15.000,   65.536,   12.536)
(x_16, y_16, err_16) = (  16.000,   68.946,   12.946)
(x_17, y_17, err_17) = (  17.000,   71.401,   12.401)
(x_18, y_18, err_18) = (  18.000,   74.699,   12.699)
(x_19, y_19, err_19) = (  19.000,   77.839,   12.839)
(x_20, y_20, err_20) = (  20.000,   67.853,    2.147)
(x_21, y_21, err_21) = (  21.000,   72.323,    0.677)
(x_22, y_22, err_22) = (  22.000,   77.306,    1.306)
(x_23, y_23, err_23) = (  23.000,   78.390,    0.610)
(x_24, y_24, err_24) = (  24.000,   83.143,    1.143)
```

```
(x_25, y_25, err_25) = (  25.000,   86.313,    1.313)
(x_26, y_26, err_26) = (  26.000,   89.857,    1.857)
(x_27, y_27, err_27) = (  27.000,   90.437,    0.563)
(x_28, y_28, err_28) = (  28.000,   93.226,    0.774)
(x_29, y_29, err_29) = (  29.000,   95.470,    1.530)
(x_30, y_30, err_30) = (  30.000,   99.168,    0.832)
(x_31, y_31, err_31) = (  31.000,  101.345,    1.655)
 active_mask: array([0., 0.])
        cost: 411.6433896361805
         fun: array([ 4.80966479,  4.84357352,  3.29248224,  3.31139096,  3.86629969
       3.33220841,  3.49411713,  4.74802585,  3.39193458,  3.0678433 ,
      -7.52424798, -6.25333925, -8.33243053, -6.61152181, -7.32061308,
      -7.31170436, -7.77079564, -7.27488691, -7.62197819, -7.81106947,
       5.12583926,  3.60674798,  1.5746567 ,  3.44156543,  1.63947415,
       1.42038287,  0.8272916 ,  3.19820032,  3.36010904,  4.06701777,
       3.31992649,  4.09383521])
        grad: array([-1.73960605e-06,  9.05574922e-08])
         jac: array([[ 0.        ,  1.        ],
       [ 0.99999998,  1.        ],
       [ 1.99999996,  1.        ],
       [ 3.00000002,  1.        ],
       [ 4.        ,  1.        ],
       [ 4.99999998,  1.        ],
       [ 6.00000004,  1.        ],
       [ 7.00000018,  1.        ],
       [ 8.        ,  1.        ],
       [ 8.99999982,  1.        ],
       [ 9.99999996,  1.        ],
       [10.99999994,  1.        ],
       [12.00000008,  1.        ],
       [13.00000006,  1.        ],
       [14.00000004,  1.        ],
       [14.99999986,  1.        ],
       [16.        ,  1.        ],
       [17.0000003 ,  1.        ],
       [17.99999996,  1.        ],
       [18.99999962,  1.        ],
       [19.9999996 ,  1.        ],
       [20.9999999 ,  1.        ],
       [22.0000002 ,  1.        ],
       [23.00000018,  1.        ],
       [24.00000016,  1.        ],
       [25.00000014,  1.        ],
       [26.00000012,  1.        ],
       [26.99999945,  1.        ],
       [28.00000008,  1.        ],
       [29.00000006,  1.        ],
       [30.00000004,  1.        ],
       [31.00000002,  1.        ]])
    message: ''xtol' termination condition is satisfied.'
       nfev: 6
       njev: 5
  optimality: 1.7396060485452836e-06
      status: 3
     success: True
           x: array([ 2.95090872, 13.96066479])
dof = 30
reduced chi-squared = 27.442892642412023
Jacobian:
```

```
[[ 0.          1.        ]
 [ 0.99999998  1.        ]
 [ 1.99999996  1.        ]
 [ 3.00000002  1.        ]
 [ 4.          1.        ]
 [ 4.99999998  1.        ]
 [ 6.00000004  1.        ]
 [ 7.00000018  1.        ]
 [ 8.          1.        ]
 [ 8.99999982  1.        ]
 [ 9.99999996  1.        ]
 [10.99999994  1.        ]
 [12.00000008  1.        ]
 [13.00000006  1.        ]
 [14.00000004  1.        ]
 [14.99999986  1.        ]
 [16.          1.        ]
 [17.0000003   1.        ]
 [17.99999996  1.        ]
 [18.99999962  1.        ]
 [19.9999996   1.        ]
 [20.9999999   1.        ]
 [22.0000002   1.        ]
 [23.00000018  1.        ]
 [24.00000016  1.        ]
 [25.00000014  1.        ]
 [26.00000012  1.        ]
 [26.99999945  1.        ]
 [28.00000008  1.        ]
 [29.00000006  1.        ]
 [30.00000004  1.        ]
 [31.00000002  1.        ]]
covariance matrix:
[[ 0.01005971 -0.15592553]
 [-0.15592553  3.27443605]]
a =     2.951 +/-     0.100 (    3.399%)
b =    13.961 +/-     1.810 (   12.962%)
```

Display PNG file. (Fig. 30) This is not what we want.

```
% feh -dF ai202209_s05_43.png
```

Try weighted least-squares method using SciPy. Here is an example.

Python Code 45: ai202209_s05_44.py

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

#
# Time-stamp: <2022/10/17 01:42:27 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy
```

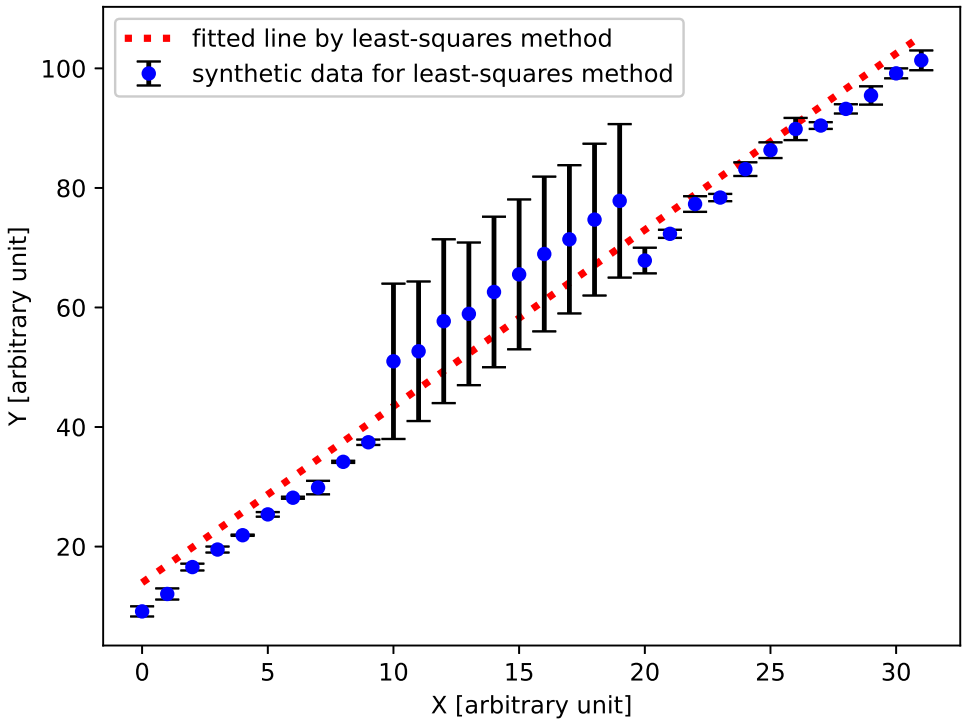

Figure 30: The result of least-squares method to fit the data to a straight line.

```
# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_41.data'

# output file name
file_output = 'ai202209_s05_44.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
```

```
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for straight line
def line (x, a, b):
    # line
    y = a * x + b
    # returning y
    return y

# a function to calculate residuals
def residual (param, x, y, y_err):
    # calculation of residual
    residue = (param[0] * x + param[1] - y) / y_err
    # returning residual
    return residue

# initial guess of coefficients
param0 = [1.0, 1.0]

# fitting
fit_lsq = scipy.optimize.least_squares (residual, param0, \
                                        args=(data_x, data_y, data_err) )

# printing result of fitting
print (f'{fit_lsq}')

# degree of freedom
dof = len (data_x) - len (fit_lsq.x)
print (f'dof = {dof}')

# reduced chi-squared
reduced_chi2 = (fit_lsq.fun**2).sum () / dof
```

```
print (f'reduced chi-squared = {reduced_chi2}')

# Jacobian
J = fit_lsq.jac
print (f'Jacobian:\n{J}')

# transpose matrix of Jacobian
Jt =J.T

# calculation of J^T J
Jt_J = numpy.matmul (Jt, J)

# covariance matrix
pcov = numpy.linalg.inv (Jt_J) * reduced_chi2
print (f'covariance matrix:\n{pcov}')

# fitted a and b
a_fitted, b_fitted = fit_lsq.x
a_err, b_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted line
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = line (fitted_x, a_fitted, b_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted line by weighted least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_44.py
(x_00, y_00, err_00) = (   0.000,    9.151,    0.849)
(x_01, y_01, err_01) = (   1.000,   12.068,    0.932)
(x_02, y_02, err_02) = (   2.000,   16.570,    0.570)
(x_03, y_03, err_03) = (   3.000,   19.502,    0.502)
(x_04, y_04, err_04) = (   4.000,   21.898,    0.102)
(x_05, y_05, err_05) = (   5.000,   25.383,    0.383)
(x_06, y_06, err_06) = (   6.000,   28.172,    0.172)
(x_07, y_07, err_07) = (   7.000,   29.869,    1.131)
(x_08, y_08, err_08) = (   8.000,   34.176,    0.176)
(x_09, y_09, err_09) = (   9.000,   37.451,    0.451)
(x_10, y_10, err_10) = (  10.000,   50.994,   12.994)
(x_11, y_11, err_11) = (  11.000,   52.674,   11.674)
(x_12, y_12, err_12) = (  12.000,   57.704,   13.704)
(x_13, y_13, err_13) = (  13.000,   58.934,   11.934)
(x_14, y_14, err_14) = (  14.000,   62.594,   12.594)
(x_15, y_15, err_15) = (  15.000,   65.536,   12.536)
(x_16, y_16, err_16) = (  16.000,   68.946,   12.946)
(x_17, y_17, err_17) = (  17.000,   71.401,   12.401)
(x_18, y_18, err_18) = (  18.000,   74.699,   12.699)
(x_19, y_19, err_19) = (  19.000,   77.839,   12.839)
(x_20, y_20, err_20) = (  20.000,   67.853,    2.147)
(x_21, y_21, err_21) = (  21.000,   72.323,    0.677)
(x_22, y_22, err_22) = (  22.000,   77.306,    1.306)
(x_23, y_23, err_23) = (  23.000,   78.390,    0.610)
(x_24, y_24, err_24) = (  24.000,   83.143,    1.143)
(x_25, y_25, err_25) = (  25.000,   86.313,    1.313)
(x_26, y_26, err_26) = (  26.000,   89.857,    1.857)
(x_27, y_27, err_27) = (  27.000,   90.437,    0.563)
(x_28, y_28, err_28) = (  28.000,   93.226,    0.774)
(x_29, y_29, err_29) = (  29.000,   95.470,    1.530)
(x_30, y_30, err_30) = (  30.000,   99.168,    0.832)
(x_31, y_31, err_31) = (  31.000,  101.345,    1.655)
 active_mask: array([0., 0.])
        cost: 13.880979569320415
         fun: array([ 1.12278293,  1.096259  , -0.86809814, -0.87917384,  1.45220987
       -0.9175037 , -0.90077468,  1.00224353, -1.06813577, -1.05880537,
       -0.849242  , -0.83344011, -0.85917318, -0.83950381, -0.84906843,
       -0.84952913, -0.85541685, -0.85023432, -0.85489292, -0.85760686,
        0.91320736,  0.70328932, -1.16493297,  0.62306265, -1.21387673,
       -1.1972509 , -1.1472912 ,  0.48836752,  0.60907182,  0.79274004,
        0.60139776,  0.79083602])
        grad: array([-1.22581564e-06,  4.33289076e-07])
         jac: array([[ 0.        ,  1.1778563 ],
       [ 1.07296136,  1.07296137],
       [ 3.50877189,  1.75438594],
       [ 5.97609548,  1.99203185],
       [39.21568588,  9.80392145],
       [13.05483026,  2.61096603],
       [34.88372105,  5.81395342],
       [ 6.18921299,  0.88417329],
       [45.454545  ,  5.68181798],
       [19.95565399,  2.21729493],
       [ 0.76958596,  0.07695859],
       [ 0.94226486,  0.08566044],
       [ 0.87565675,  0.07297139],
       [ 1.08932462,  0.0837942 ],
       [ 1.11164046,  0.07940289],
```

```
       [ 1.19655392,  0.07977026],
       [ 1.23590298,  0.07724393],
       [ 1.37085719,  0.08063866],
       [ 1.41743445,  0.07874636],
       [ 1.47986603,  0.07788769],
       [ 9.31532369,  0.46576617],
       [31.01920235,  1.47710496],
       [16.84532926,  0.76569676],
       [37.70491759,  1.6393442 ],
       [20.9973754 ,  0.87489061],
       [19.04036566,  0.7616146 ],
       [14.00107708,  0.53850294],
       [47.95737096,  1.77619887],
       [36.17571044,  1.29198962],
       [18.95424852,  0.65359475],
       [36.05769224,  1.20192303],
       [18.73111781,  0.60422959]])
     message: 'Both `ftol` and `xtol` termination conditions are satisfied.'
        nfev: 6
        njev: 6
  optimality: 1.225815639216421e-06
      status: 4
     success: True
           x: array([ 2.98547067, 10.10424271])
dof = 30
reduced chi-squared = 0.9253986379546946
Jacobian:
[[ 0.          1.1778563 ]
 [ 1.07296136  1.07296137]
 [ 3.50877189  1.75438594]
 [ 5.97609548  1.99203185]
 [39.21568588  9.80392145]
 [13.05483026  2.61096603]
 [34.88372105  5.81395342]
 [ 6.18921299  0.88417329]
 [45.454545    5.68181798]
 [19.95565399  2.21729493]
 [ 0.76958596  0.07695859]
 [ 0.94226486  0.08566044]
 [ 0.87565675  0.07297139]
 [ 1.08932462  0.0837942 ]
 [ 1.11164046  0.07940289]
 [ 1.19655392  0.07977026]
 [ 1.23590298  0.07724393]
 [ 1.37085719  0.08063866]
 [ 1.41743445  0.07874636]
 [ 1.47986603  0.07788769]
 [ 9.31532369  0.46576617]
 [31.01920235  1.47710496]
 [16.84532926  0.76569676]
 [37.70491759  1.6393442 ]
 [20.9973754   0.87489061]
 [19.04036566  0.7616146 ]
 [14.00107708  0.53850294]
 [47.95737096  1.77619887]
 [36.17571044  1.29198962]
 [18.95424852  0.65359475]
 [36.05769224  1.20192303]
 [18.73111781  0.60422959]]
```

```
covariance matrix:
[[ 0.00014976 -0.00098998]
 [-0.00098998  0.01120068]]
a =     2.985 +/-     0.012 (    0.410%)
b =    10.104 +/-     0.106 (    1.047%)
```

Display PNG file. (Fig. 31) The fit looks OK.

```
% feh -dF ai202209_s05_44.png
```

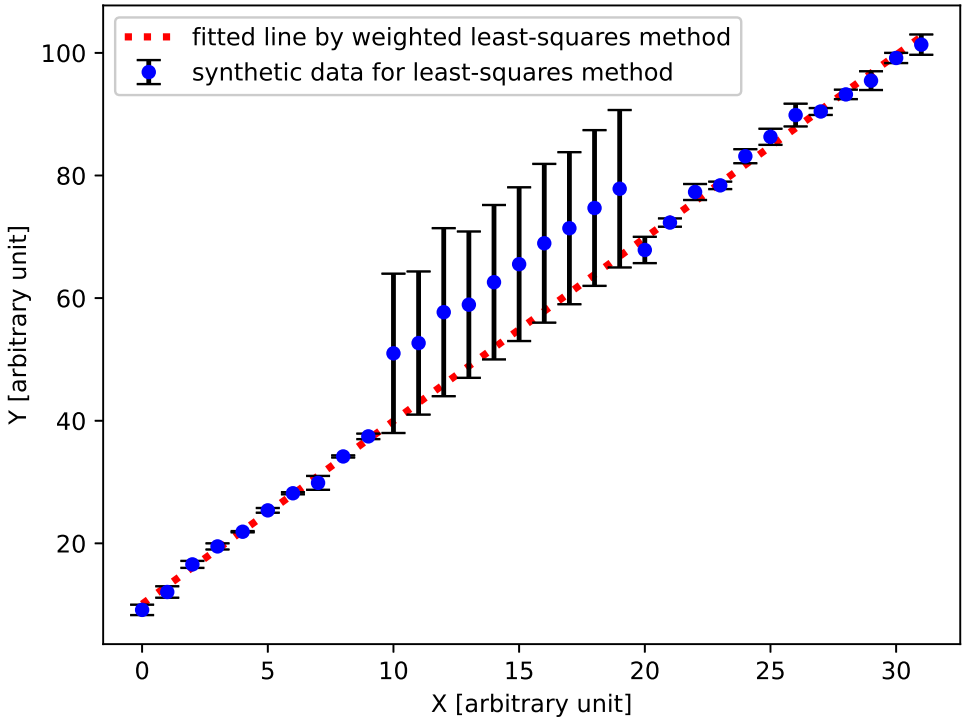

Figure 31: The result of weighted least-squares method to fit the data to a straight line.

Try following practice.

Practice 05-30

Generate a set of synthetic data with error and carry out weighted least-squares method.

11.3 Least-squares method using `curve_fit`

Generate a set of synthetic data. Here is an example.

Python Code 46: ai202209_s05_45.py

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

#
# Time-stamp: <2022/10/16 20:10:07 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

# importing scipy module
import scipy.stats

# output file name
file_output = 'ai202209_s05_45.data'

# generating random numbers
err = scipy.stats.norm.rvs (loc=0.0, scale=2.0, size=16)

# function for a line
def curve (x):
    # coefficients
    a = 2.0
    b = 7.0
    c = 5.0
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# synthetic data for least-squares method
data_x = numpy.linspace (0.0, 15.0, 16)
data_y = curve (data_x) + err

# opening file for writing
with open (file_output, 'w') as fh:
    # writing generated synthetic data into file
    for i in range (len (data_x)):
        fh.write (f'{data_x[i]:8.3f}    {data_y[i]:8.3f}\n')
```

Execute above script.

```
% ./ai202209_s05_45.py
```

Show generated data.

```
% cat ai202209_s05_45.data
   0.000     102.770
   1.000      78.232
   2.000      53.174
   3.000      33.937
   4.000      20.523
   5.000      17.035
   6.000       9.238
   7.000       0.473
   8.000      10.726
   9.000      17.325
  10.000      21.837
  11.000      35.852
  12.000      57.142
  13.000      77.242
  14.000     104.698
  15.000     132.374
```

Plot the synthetic data using Matplotlib. Here is an example.

Python Code 47: ai202209_s05_46.py

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

#
# Time-stamp: <2022/10/17 01:48:02 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s05_45.data'

# output file name
file_output = 'ai202209_s05_46.png'

# making empty numpy arrays
data_x = numpy.array ([])
data_y = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x = numpy.append (data_x, x)
        data_y = numpy.append (data_y, y)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}) = ({data_x[i]:8.3f}, {data_y[i]:8.3f})')

#
```

```
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='synthetic data for least-squares method')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_46.py
(x_00, y_00) = (   0.000,  102.770)
(x_01, y_01) = (   1.000,   78.232)
(x_02, y_02) = (   2.000,   53.174)
(x_03, y_03) = (   3.000,   33.937)
(x_04, y_04) = (   4.000,   20.523)
(x_05, y_05) = (   5.000,   17.035)
(x_06, y_06) = (   6.000,    9.238)
(x_07, y_07) = (   7.000,    0.473)
(x_08, y_08) = (   8.000,   10.726)
(x_09, y_09) = (   9.000,   17.325)
(x_10, y_10) = (  10.000,   21.837)
(x_11, y_11) = (  11.000,   35.852)
(x_12, y_12) = (  12.000,   57.142)
(x_13, y_13) = (  13.000,   77.242)
(x_14, y_14) = (  14.000,  104.698)
(x_15, y_15) = (  15.000,  132.374)
```

Display PNG file. (Fig. 32)

```
% feh -dF ai202209_s05_46.png
```

Try least-squares method using SciPy. Here is an example.

Python Code 48: ai202209_s05_47.py

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

#
# Time-stamp: <2022/10/17 01:49:33 (CST) daisuke>
#

# importing sys module
import sys
```

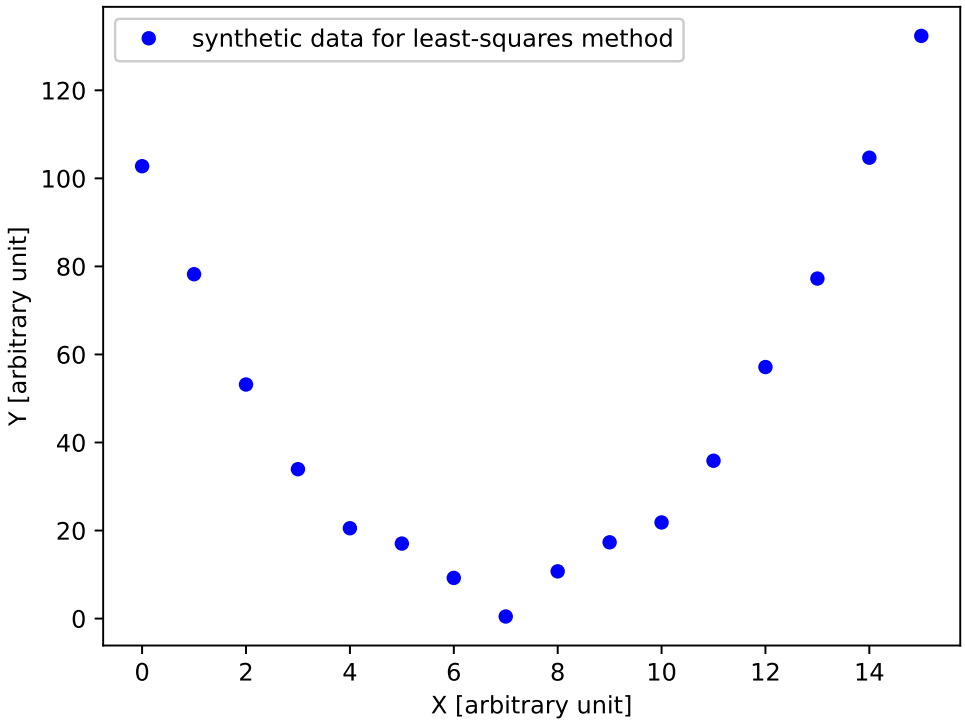

Figure 32: A plot of synthetic data for least-squares method.

```
# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_45.data'

# output file name
file_output = 'ai202209_s05_47.png'

# making empty numpy arrays
data_x = numpy.array ([])
data_y = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str) = line.split ()
```

```
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x = numpy.append (data_x, x)
        data_y = numpy.append (data_y, y)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}) = ({data_x[i]:8.3f}, {data_y[i]:8.3f})')

# a function for curve
def curve (x, a, b, c):
    # curve
    y = a * (x - b)**2 + c
    # returning y
    return y

# initial guess of coefficients
param0 = [1.0, 1.0, 1.0]

# fitting
popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0)

# result of fitting
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted, c_fitted = popt

# degree of freedom
dof = len (data_x) - len (popt)
print (f'dof = {dof}')

# reduced chi-squared
residual     = data_y - curve (data_x, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err, c_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')
print (f'c = {c_fitted:8.3f} +/- {c_err:8.3f} ({c_err/c_fitted*100.0:8.3f}%)')

# range of data
```

```
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = curve (fitted_x, a_fitted, b_fitted, c_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.plot (data_x, data_y, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='synthetic data for least-squares method', \
         zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted curve by least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_47.py
(x_00, y_00) = (   0.000,  102.770)
(x_01, y_01) = (   1.000,   78.232)
(x_02, y_02) = (   2.000,   53.174)
(x_03, y_03) = (   3.000,   33.937)
(x_04, y_04) = (   4.000,   20.523)
(x_05, y_05) = (   5.000,   17.035)
(x_06, y_06) = (   6.000,    9.238)
(x_07, y_07) = (   7.000,    0.473)
(x_08, y_08) = (   8.000,   10.726)
(x_09, y_09) = (   9.000,   17.325)
(x_10, y_10) = (  10.000,   21.837)
(x_11, y_11) = (  11.000,   35.852)
(x_12, y_12) = (  12.000,   57.142)
(x_13, y_13) = (  13.000,   77.242)
(x_14, y_14) = (  14.000,  104.698)
(x_15, y_15) = (  15.000,  132.374)
popt:
[1.98247619 6.97053069 5.60274302]
pcov:
[[ 1.33281053e-03  3.55960022e-04 -2.79485867e-02]
 [ 3.55960022e-04  1.51936881e-03 -4.47429761e-03]
 [-2.79485867e-02 -4.47429761e-03  1.06816287e+00]]
dof = 13
```

```
reduced chi-squared = 7.61301365906055
a =     1.982 +/-     0.037 (    1.842%)
b =     6.971 +/-     0.039 (    0.559%)
c =     5.603 +/-     1.034 (   18.447%)
```

Display PNG file. (Fig. 33)

```
% feh -dF ai202209_s05_47.png
```

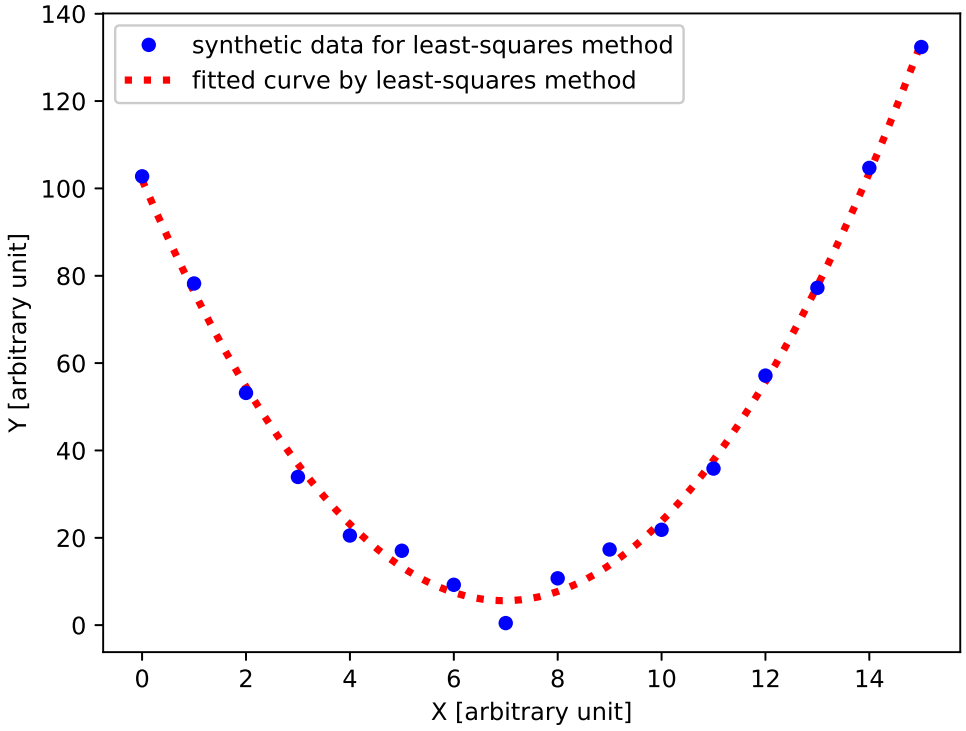

Figure 33: The result of least-squares method to fit the data to a curve.

Try following practice.

Practice 05-31

Generate a set of synthetic data and carry out least-squares method using `curve_fit`.

11.4 Weighted least-squares method using `curve_fit`

Generate a set of synthetic data with errors. Here is an example.

Python Code 49: ai202209_s05_48.py

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

#
# Time-stamp: <2022/10/16 20:27:18 (CST) daisuke>
#
```

```
# importing numpy module
import numpy

# importing scipy module
import scipy.stats

# output file name
file_output = 'ai202209_s05_48.data'

# generating random numbers
err = scipy.stats.norm.rvs (loc=0.0, scale=5.0, size=32)

# function for a line
def curve (x):
    # coefficients
    a = 2.0
    b = 13.0
    c = 5.0
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# synthetic data for least-squares method
data_x   = numpy.linspace (0.0, 31.0, 32)
data_y   = curve (data_x) + err
data_err = numpy.absolute (err)
for i in range (0, 7):
    data_y[i]   -= 50.0
    data_err[i] += 50.0
for i in range (25, 32):
    data_y[i]   += 50.0
    data_err[i] += 50.0

# opening file for writing
with open (file_output, 'w') as fh:
    # writing generated synthetic data into file
    for i in range (len (data_x)):
        fh.write (f'{data_x[i]:8.3f}  {data_y[i]:8.3f}  {data_err[i]:8.3f}\n')
```

Execute above script.

```
% ./ai202209_s05_48.py
```

Show generated data.

```
% cat ai202209_s05_48.data
   0.000   298.037    55.037
   1.000   243.727    50.727
   2.000   195.670    51.330
   3.000   146.815    58.185
   4.000   118.874    51.874
   5.000    95.231    62.231
   6.000    52.668    50.332
   7.000    74.068     2.932
   8.000    56.342     1.342
   9.000    35.480     1.520
  10.000    14.929     8.071
```

```
  11.000    16.755     3.755
  12.000     7.075     0.075
  13.000     6.653     1.653
  14.000     7.753     0.753
  15.000    11.943     1.057
  16.000    28.294     5.294
  17.000    35.222     1.778
  18.000    66.076    11.076
  19.000    84.353     7.353
  20.000    97.038     5.962
  21.000   137.383     4.383
  22.000   155.149    11.851
  23.000   200.199     4.801
  24.000   250.649     3.649
  25.000   348.704    55.704
  26.000   400.097    57.097
  27.000   449.563    52.563
  28.000   499.167    55.833
  29.000   570.620    53.620
  30.000   626.495    56.505
  31.000   711.954    58.954
```

Plot the synthetic data using Matplotlib. Here is an example.

Python Code 50: ai202209_s05_49.py

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

#
# Time-stamp: <2022/10/17 01:55:06 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s05_48.data'

# output file name
file_output = 'ai202209_s05_49.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str, err_str) = line.split ()
```

```
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_49.py
(x_00, y_00, err_00) = (   0.000,  298.037,   55.037)
(x_01, y_01, err_01) = (   1.000,  243.727,   50.727)
(x_02, y_02, err_02) = (   2.000,  195.670,   51.330)
(x_03, y_03, err_03) = (   3.000,  146.815,   58.185)
(x_04, y_04, err_04) = (   4.000,  118.874,   51.874)
(x_05, y_05, err_05) = (   5.000,   95.231,   62.231)
(x_06, y_06, err_06) = (   6.000,   52.668,   50.332)
(x_07, y_07, err_07) = (   7.000,   74.068,    2.932)
(x_08, y_08, err_08) = (   8.000,   56.342,    1.342)
(x_09, y_09, err_09) = (   9.000,   35.480,    1.520)
(x_10, y_10, err_10) = (  10.000,   14.929,    8.071)
(x_11, y_11, err_11) = (  11.000,   16.755,    3.755)
(x_12, y_12, err_12) = (  12.000,    7.075,    0.075)
(x_13, y_13, err_13) = (  13.000,    6.653,    1.653)
(x_14, y_14, err_14) = (  14.000,    7.753,    0.753)
(x_15, y_15, err_15) = (  15.000,   11.943,    1.057)
(x_16, y_16, err_16) = (  16.000,   28.294,    5.294)
(x_17, y_17, err_17) = (  17.000,   35.222,    1.778)
(x_18, y_18, err_18) = (  18.000,   66.076,   11.076)
(x_19, y_19, err_19) = (  19.000,   84.353,    7.353)
(x_20, y_20, err_20) = (  20.000,   97.038,    5.962)
(x_21, y_21, err_21) = (  21.000,  137.383,    4.383)
(x_22, y_22, err_22) = (  22.000,  155.149,   11.851)
(x_23, y_23, err_23) = (  23.000,  200.199,    4.801)
(x_24, y_24, err_24) = (  24.000,  250.649,    3.649)
(x_25, y_25, err_25) = (  25.000,  348.704,   55.704)
(x_26, y_26, err_26) = (  26.000,  400.097,   57.097)
(x_27, y_27, err_27) = (  27.000,  449.563,   52.563)
(x_28, y_28, err_28) = (  28.000,  499.167,   55.833)
(x_29, y_29, err_29) = (  29.000,  570.620,   53.620)
(x_30, y_30, err_30) = (  30.000,  626.495,   56.505)
(x_31, y_31, err_31) = (  31.000,  711.954,   58.954)
```

Display PNG file. (Fig. 34)

```
% feh -dF ai202209_s05_49.png
```

Try least-squares method using SciPy. Here is an example.

Python Code 51: ai202209_s05_50.py

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

#
# Time-stamp: <2022/10/17 01:56:43 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats
```

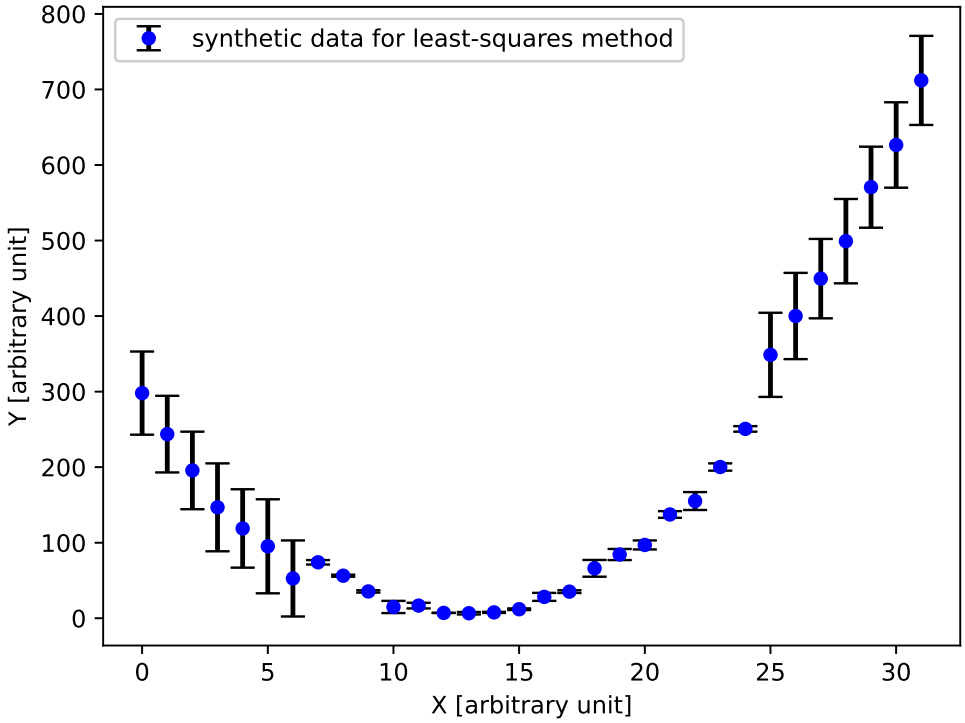

Figure 34: A plot of synthetic data for least-squares method.

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

# input file name
file_input = 'ai202209_s05_48.data'

# output file name
file_output = 'ai202209_s05_50.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
```

```
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for curve
def curve (x, a, b, c):
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# initial guess of coefficients
param0 = [1.0, 1.0, 1.0]

# fitting
popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0)

# result of fitting
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted, c_fitted = popt

# degree of freedom
dof = len (data_x) - len (popt)
print (f'dof = {dof}')

# reduced chi-squared
residual     = data_y - curve (data_x, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err, c_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')
print (f'c = {c_fitted:8.3f} +/- {c_err:8.3f} ({c_err/c_fitted*100.0:8.3f}%)')
```

```
# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = curve (fitted_x, a_fitted, b_fitted, c_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted curve by least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_50.py
(x_00, y_00, err_00) = (   0.000,  298.037,   55.037)
(x_01, y_01, err_01) = (   1.000,  243.727,   50.727)
(x_02, y_02, err_02) = (   2.000,  195.670,   51.330)
(x_03, y_03, err_03) = (   3.000,  146.815,   58.185)
(x_04, y_04, err_04) = (   4.000,  118.874,   51.874)
(x_05, y_05, err_05) = (   5.000,   95.231,   62.231)
(x_06, y_06, err_06) = (   6.000,   52.668,   50.332)
(x_07, y_07, err_07) = (   7.000,   74.068,    2.932)
(x_08, y_08, err_08) = (   8.000,   56.342,    1.342)
(x_09, y_09, err_09) = (   9.000,   35.480,    1.520)
(x_10, y_10, err_10) = (  10.000,   14.929,    8.071)
(x_11, y_11, err_11) = (  11.000,   16.755,    3.755)
(x_12, y_12, err_12) = (  12.000,    7.075,    0.075)
(x_13, y_13, err_13) = (  13.000,    6.653,    1.653)
(x_14, y_14, err_14) = (  14.000,    7.753,    0.753)
(x_15, y_15, err_15) = (  15.000,   11.943,    1.057)
(x_16, y_16, err_16) = (  16.000,   28.294,    5.294)
(x_17, y_17, err_17) = (  17.000,   35.222,    1.778)
(x_18, y_18, err_18) = (  18.000,   66.076,   11.076)
```

```
(x_19, y_19, err_19) = (  19.000,   84.353,    7.353)
(x_20, y_20, err_20) = (  20.000,   97.038,    5.962)
(x_21, y_21, err_21) = (  21.000,  137.383,    4.383)
(x_22, y_22, err_22) = (  22.000,  155.149,   11.851)
(x_23, y_23, err_23) = (  23.000,  200.199,    4.801)
(x_24, y_24, err_24) = (  24.000,  250.649,    3.649)
(x_25, y_25, err_25) = (  25.000,  348.704,   55.704)
(x_26, y_26, err_26) = (  26.000,  400.097,   57.097)
(x_27, y_27, err_27) = (  27.000,  449.563,   52.563)
(x_28, y_28, err_28) = (  28.000,  499.167,   55.833)
(x_29, y_29, err_29) = (  29.000,  570.620,   53.620)
(x_30, y_30, err_30) = (  30.000,  626.495,   56.505)
(x_31, y_31, err_31) = (  31.000,  711.954,   58.954)
popt:
[ 2.00313276 12.19603601 -3.79040287]
pcov:
[[ 1.48699231e-03  2.45264281e-03 -1.10533822e-01]
 [ 2.45264281e-03  1.03453510e-02 -9.89244611e-02]
 [-1.10533822e-01 -9.89244611e-02  1.79402903e+01]]
dof = 29
reduced chi-squared = 275.8430212997363
a =    2.003 +/-    0.039 (   1.925%)
b =   12.196 +/-    0.102 (   0.834%)
c =   -3.790 +/-    4.236 (-111.745%)
```

Display PNG file. (Fig. 35) The result of the fit is not what we want.

```
% feh -dF ai202209_s05_50.png
```

Try weighted least-squares method using SciPy. Here is an example.

Python Code 52: ai202209_s05_51.py

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

#
# Time-stamp: <2022/10/17 01:59:26 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_48.data'

# output file name
file_output = 'ai202209_s05_51.png'
```

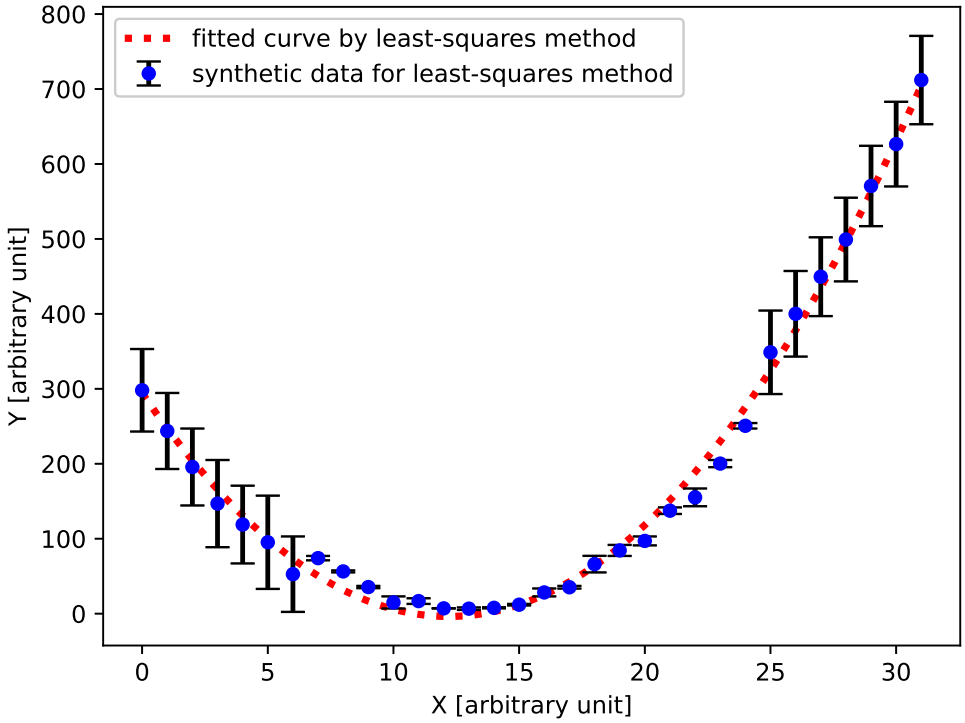

Figure 35: The result of least-squares method to fit the data to a curve.

```
# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
```

```
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for curve
def curve (x, a, b, c):
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# initial guess of coefficients
param0 = [1.0, 1.0, 1.0]

# fitting
popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0, \
                                       sigma=data_err)

# result of fitting
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted, c_fitted = popt

# degree of freedom
dof = len (data_x) - len (popt)
print (f'dof = {dof}')

# reduced chi-squared
residual     = data_y - curve (data_x, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err, c_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')
print (f'c = {c_fitted:8.3f} +/- {c_err:8.3f} ({c_err/c_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = curve (fitted_x, a_fitted, b_fitted, c_fitted)
```

```
#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted curve by weighted least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_51.py
(x_00, y_00, err_00) = (   0.000,  298.037,   55.037)
(x_01, y_01, err_01) = (   1.000,  243.727,   50.727)
(x_02, y_02, err_02) = (   2.000,  195.670,   51.330)
(x_03, y_03, err_03) = (   3.000,  146.815,   58.185)
(x_04, y_04, err_04) = (   4.000,  118.874,   51.874)
(x_05, y_05, err_05) = (   5.000,   95.231,   62.231)
(x_06, y_06, err_06) = (   6.000,   52.668,   50.332)
(x_07, y_07, err_07) = (   7.000,   74.068,    2.932)
(x_08, y_08, err_08) = (   8.000,   56.342,    1.342)
(x_09, y_09, err_09) = (   9.000,   35.480,    1.520)
(x_10, y_10, err_10) = (  10.000,   14.929,    8.071)
(x_11, y_11, err_11) = (  11.000,   16.755,    3.755)
(x_12, y_12, err_12) = (  12.000,    7.075,    0.075)
(x_13, y_13, err_13) = (  13.000,    6.653,    1.653)
(x_14, y_14, err_14) = (  14.000,    7.753,    0.753)
(x_15, y_15, err_15) = (  15.000,   11.943,    1.057)
(x_16, y_16, err_16) = (  16.000,   28.294,    5.294)
(x_17, y_17, err_17) = (  17.000,   35.222,    1.778)
(x_18, y_18, err_18) = (  18.000,   66.076,   11.076)
(x_19, y_19, err_19) = (  19.000,   84.353,    7.353)
(x_20, y_20, err_20) = (  20.000,   97.038,    5.962)
(x_21, y_21, err_21) = (  21.000,  137.383,    4.383)
(x_22, y_22, err_22) = (  22.000,  155.149,   11.851)
(x_23, y_23, err_23) = (  23.000,  200.199,    4.801)
(x_24, y_24, err_24) = (  24.000,  250.649,    3.649)
(x_25, y_25, err_25) = (  25.000,  348.704,   55.704)
(x_26, y_26, err_26) = (  26.000,  400.097,   57.097)
```

```
(x_27, y_27, err_27) = (  27.000,  449.563,   52.563)
(x_28, y_28, err_28) = (  28.000,  499.167,   55.833)
(x_29, y_29, err_29) = (  29.000,  570.620,   53.620)
(x_30, y_30, err_30) = (  30.000,  626.495,   56.505)
(x_31, y_31, err_31) = (  31.000,  711.954,   58.954)
popt:
[ 2.00244289 12.98647428  5.12600138]
pcov:
[[ 0.00043395  0.00025594 -0.00152057]
 [ 0.00025594  0.0011109  -0.00455378]
 [-0.00152057 -0.00455378  0.02468447]]
dof = 29
reduced chi-squared = 1221.3805461484149
a =    2.002 +/-    0.021 (   1.040%)
b =   12.986 +/-    0.033 (   0.257%)
c =    5.126 +/-    0.157 (   3.065%)
```

Display PNG file. (Fig. 36) The result of the fit looks OK.

```
% feh -dF ai202209_s05_51.png
```

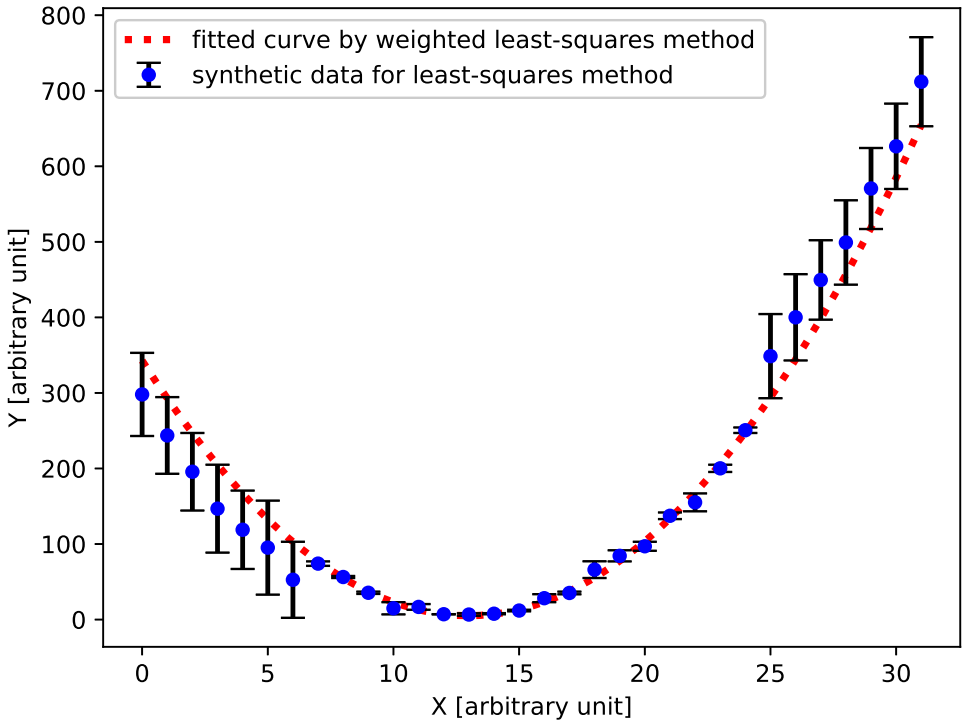

Figure 36: The result of weighted least-squares method to fit the data to a curve.

Try following practice.

Practice 05-32

Generate a set of synthetic data with error and carry out weighted least-squares method using `curve_fit`.

11.5 Initial guess of fitting coefficients

Generate a set of synthetic data with errors. Here is an example.

Python Code 53: ai202209_s05_52.py

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

#
# Time-stamp: <2022/10/16 20:50:14 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.stats

# output file name
file_output = 'ai202209_s05_52.data'

# generating random numbers
err = scipy.stats.norm.rvs (loc=0.0, scale=5.0, size=21)

# function for a line
def curve (x):
    # coefficients
    a = 2.0
    b = 3000.0
    c = 5.0
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# synthetic data for least-squares method
data_x   = numpy.linspace (2990.0, 3010.0, 21)
data_y   = curve (data_x) + err
data_err = numpy.absolute (err)

# opening file for writing
with open (file_output, 'w') as fh:
    # writing generated synthetic data into file
    for i in range (len (data_x)):
        fh.write (f'{data_x[i]:8.3f}  {data_y[i]:8.3f}  {data_err[i]:8.3f}\n')
```

Execute above script.

```
% ./ai202209_s05_52.py
```

Show generated data.

```
% cat ai202209_s05_52.data
2990.000   204.341     0.659
2991.000   158.566     8.434
2992.000   133.277     0.277
2993.000    99.501     3.499
2994.000    72.365     4.635
2995.000    51.738     3.262
```

```
2996.000    38.450     1.450
2997.000    18.559     4.441
2998.000    13.774     0.774
2999.000     1.298     5.702
3000.000    17.020    12.020
3001.000    11.905     4.905
3002.000     6.399     6.601
3003.000    24.515     1.515
3004.000    31.147     5.853
3005.000    47.470     7.530
3006.000    68.607     8.393
3007.000    99.938     3.062
3008.000   128.760     4.240
3009.000   156.914    10.086
3010.000   208.565     3.565
```

Plot the synthetic data using Matplotlib. Here is an example.

Python Code 54: ai202209_s05_53.py

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

#
# Time-stamp: <2022/10/17 02:06:43 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

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

# input file name
file_input = 'ai202209_s05_52.data'

# output file name
file_output = 'ai202209_s05_53.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x" and "y"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
```

```
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_53.py
(x_00, y_00, err_00) = (2990.000,  204.341,    0.659)
(x_01, y_01, err_01) = (2991.000,  158.566,    8.434)
(x_02, y_02, err_02) = (2992.000,  133.277,    0.277)
(x_03, y_03, err_03) = (2993.000,   99.501,    3.499)
(x_04, y_04, err_04) = (2994.000,   72.365,    4.635)
```

```
(x_05, y_05, err_05) = (2995.000,    51.738,     3.262)
(x_06, y_06, err_06) = (2996.000,    38.450,     1.450)
(x_07, y_07, err_07) = (2997.000,    18.559,     4.441)
(x_08, y_08, err_08) = (2998.000,    13.774,     0.774)
(x_09, y_09, err_09) = (2999.000,     1.298,     5.702)
(x_10, y_10, err_10) = (3000.000,    17.020,    12.020)
(x_11, y_11, err_11) = (3001.000,    11.905,     4.905)
(x_12, y_12, err_12) = (3002.000,     6.399,     6.601)
(x_13, y_13, err_13) = (3003.000,    24.515,     1.515)
(x_14, y_14, err_14) = (3004.000,    31.147,     5.853)
(x_15, y_15, err_15) = (3005.000,    47.470,     7.530)
(x_16, y_16, err_16) = (3006.000,    68.607,     8.393)
(x_17, y_17, err_17) = (3007.000,    99.938,     3.062)
(x_18, y_18, err_18) = (3008.000,   128.760,     4.240)
(x_19, y_19, err_19) = (3009.000,   156.914,    10.086)
(x_20, y_20, err_20) = (3010.000,   208.565,     3.565)
```

Display PNG file. (Fig. 37)

```
% feh -dF ai202209_s05_53.png
```

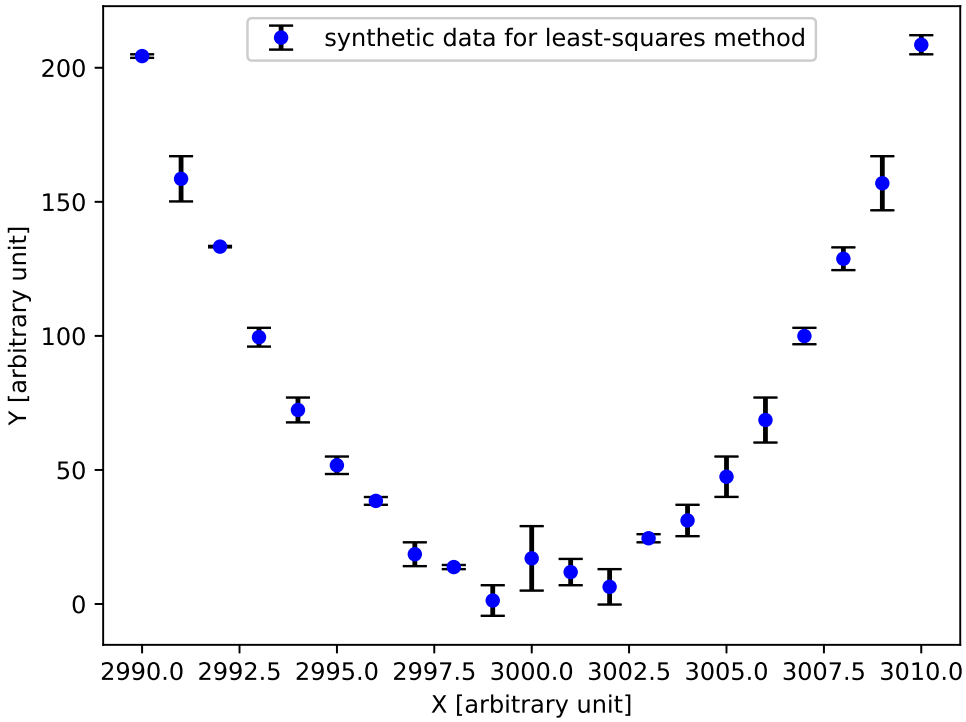

Figure 37: A plot of synthetic data for least-squares method.

Try weighted least-squares method using SciPy. Here is an example.

Python Code 55: ai202209_s05_54.py

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

```
#
# Time-stamp: <2022/10/16 20:52:21 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_52.data'

# output file name
file_output = 'ai202209_s05_54.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
```

```
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for curve
def curve (x, a, b, c):
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# initial guess of coefficients
param0 = [1.0, 1.0, 1.0]

# fitting
popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0, \
                                       sigma=data_err)

# result of fitting
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted, c_fitted = popt

# degree of freedom
dof = len (data_x) - len (popt)
print (f'dof = {dof}')

# reduced chi-squared
residual     = data_y - curve (data_x, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err, c_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')
print (f'c = {c_fitted:8.3f} +/- {c_err:8.3f} ({c_err/c_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = curve (fitted_x, a_fitted, b_fitted, c_fitted)

#
# making plot using Matplotlib
#

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

```
ax      = fig.add_subplot (111)

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted curve by weighted least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_54.py
(x_00, y_00, err_00) = (2990.000,  204.341,    0.659)
(x_01, y_01, err_01) = (2991.000,  158.566,    8.434)
(x_02, y_02, err_02) = (2992.000,  133.277,    0.277)
(x_03, y_03, err_03) = (2993.000,   99.501,    3.499)
(x_04, y_04, err_04) = (2994.000,   72.365,    4.635)
(x_05, y_05, err_05) = (2995.000,   51.738,    3.262)
(x_06, y_06, err_06) = (2996.000,   38.450,    1.450)
(x_07, y_07, err_07) = (2997.000,   18.559,    4.441)
(x_08, y_08, err_08) = (2998.000,   13.774,    0.774)
(x_09, y_09, err_09) = (2999.000,    1.298,    5.702)
(x_10, y_10, err_10) = (3000.000,   17.020,   12.020)
(x_11, y_11, err_11) = (3001.000,   11.905,    4.905)
(x_12, y_12, err_12) = (3002.000,    6.399,    6.601)
(x_13, y_13, err_13) = (3003.000,   24.515,    1.515)
(x_14, y_14, err_14) = (3004.000,   31.147,    5.853)
(x_15, y_15, err_15) = (3005.000,   47.470,    7.530)
(x_16, y_16, err_16) = (3006.000,   68.607,    8.393)
(x_17, y_17, err_17) = (3007.000,   99.938,    3.062)
(x_18, y_18, err_18) = (3008.000,  128.760,    4.240)
(x_19, y_19, err_19) = (3009.000,  156.914,   10.086)
(x_20, y_20, err_20) = (3010.000,  208.565,    3.565)
Traceback (most recent call last):
  File "/amd/ogikubo/root/volume1/nas0/daisuke/backup/daisuke/tex/astro/NCU/Lect
ure/Astroinformatics_202209/session_05/script_05/./ai202209_s05_54.py", line 81,
 in <module>
    popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0, \
  File "/usr/pkg/lib/python3.9/site-packages/scipy/optimize/_minpack_py.py", lin
e 794, in curve_fit
    raise RuntimeError("Optimal parameters not found: " + errmsg)
RuntimeError: Optimal parameters not found: Number of calls to function has reac
hed maxfev = 800.
```

The least-squares method failed, and coefficients were not found.
Change the initial guess of coefficients and try weighted least-squares method using SciPy again. Here is an example.

Python Code 56: ai202209_s05_55.py

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

#
# Time-stamp: <2022/10/17 02:10:55 (CST) daisuke>
#

# importing sys module
import sys

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'ai202209_s05_52.data'

# output file name
file_output = 'ai202209_s05_55.png'

# making empty numpy arrays
data_x   = numpy.array ([])
data_y   = numpy.array ([])
data_err = numpy.array ([])

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # splitting line into "x", "y", and "err"
        (x_str, y_str, err_str) = line.split ()
        # converting string into float
        try:
            x = float (x_str)
        except:
            print (f'cannot convert "{x_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            y = float (y_str)
        except:
            print (f'cannot convert "{y_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
            sys.exit (1)
        try:
            err = float (err_str)
        except:
            print (f'cannot convert "{err_str}" into float.')
            print (f'something is wrong.')
            print (f'exiting...')
```

```
            sys.exit (1)
        # appending data into numpy arrays
        data_x   = numpy.append (data_x, x)
        data_y   = numpy.append (data_y, y)
        data_err = numpy.append (data_err, err)

# printing data
for i in range (len (data_x)):
    print (f'(x_{i:02d}, y_{i:02d}, err_{i:02d})', \
           f'= ({data_x[i]:8.3f}, {data_y[i]:8.3f}, {data_err[i]:8.3f})')

# a function for curve
def curve (x, a, b, c):
    # line
    y = a * (x - b)**2 + c
    # returning y
    return y

# initial guess of coefficients
param0 = [1.0, 5000.0, 1.0]

# fitting
popt, pcov = scipy.optimize.curve_fit (curve, data_x, data_y, p0=param0, \
                                       sigma=data_err)

# result of fitting
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted, c_fitted = popt

# degree of freedom
dof = len (data_x) - len (popt)
print (f'dof = {dof}')

# reduced chi-squared
residual     = data_y - curve (data_x, a_fitted, b_fitted, c_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err, c_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.3f} +/- {a_err:8.3f} ({a_err/a_fitted*100.0:8.3f}%)')
print (f'b = {b_fitted:8.3f} +/- {b_err:8.3f} ({b_err/b_fitted*100.0:8.3f}%)')
print (f'c = {c_fitted:8.3f} +/- {c_err:8.3f} ({c_err/c_fitted*100.0:8.3f}%)')

# range of data
x_min = scipy.stats.tmin (data_x)
x_max = scipy.stats.tmax (data_x)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = curve (fitted_x, a_fitted, b_fitted, c_fitted)

#
# making plot using Matplotlib
#
```

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

# axes
ax.set_xlabel ('X [arbitrary unit]')
ax.set_ylabel ('Y [arbitrary unit]')

# plotting data
ax.errorbar (data_x, data_y, yerr=data_err, \
             linestyle='None', marker='o', markersize=5.0, color='blue', \
             elinewidth=2.0, ecolor='black', capsize=5.0, \
             label='synthetic data for least-squares method', \
             zorder=0.2)
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, color='red', \
         label='fitted curve by weighted least-squares method', zorder=0.1)

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_55.py
(x_00, y_00, err_00) = (2990.000,  204.341,    0.659)
(x_01, y_01, err_01) = (2991.000,  158.566,    8.434)
(x_02, y_02, err_02) = (2992.000,  133.277,    0.277)
(x_03, y_03, err_03) = (2993.000,   99.501,    3.499)
(x_04, y_04, err_04) = (2994.000,   72.365,    4.635)
(x_05, y_05, err_05) = (2995.000,   51.738,    3.262)
(x_06, y_06, err_06) = (2996.000,   38.450,    1.450)
(x_07, y_07, err_07) = (2997.000,   18.559,    4.441)
(x_08, y_08, err_08) = (2998.000,   13.774,    0.774)
(x_09, y_09, err_09) = (2999.000,    1.298,    5.702)
(x_10, y_10, err_10) = (3000.000,   17.020,   12.020)
(x_11, y_11, err_11) = (3001.000,   11.905,    4.905)
(x_12, y_12, err_12) = (3002.000,    6.399,    6.601)
(x_13, y_13, err_13) = (3003.000,   24.515,    1.515)
(x_14, y_14, err_14) = (3004.000,   31.147,    5.853)
(x_15, y_15, err_15) = (3005.000,   47.470,    7.530)
(x_16, y_16, err_16) = (3006.000,   68.607,    8.393)
(x_17, y_17, err_17) = (3007.000,   99.938,    3.062)
(x_18, y_18, err_18) = (3008.000,  128.760,    4.240)
(x_19, y_19, err_19) = (3009.000,  156.914,   10.086)
(x_20, y_20, err_20) = (3010.000,  208.565,    3.565)
popt:
[1.98227047e+00 3.00001982e+03 5.65426534e+00]
pcov:
[[ 1.84845742e-04 -2.67512170e-04 -3.63455056e-03]
 [-2.67512170e-04  7.21914441e-04 -4.06310410e-03]
 [-3.63455056e-03 -4.06310410e-03  3.86995799e-01]]
dof = 18
reduced chi-squared = 36.907818739547196
a =    1.982 +/-    0.014 (   0.686%)
b = 3000.020 +/-    0.027 (   0.001%)
c =    5.654 +/-    0.622 (  11.002%)
```

This time, coefficients were successfully found.
Display PNG file. (Fig. 38)

```
% feh -dF ai202209_s05_55.png
```

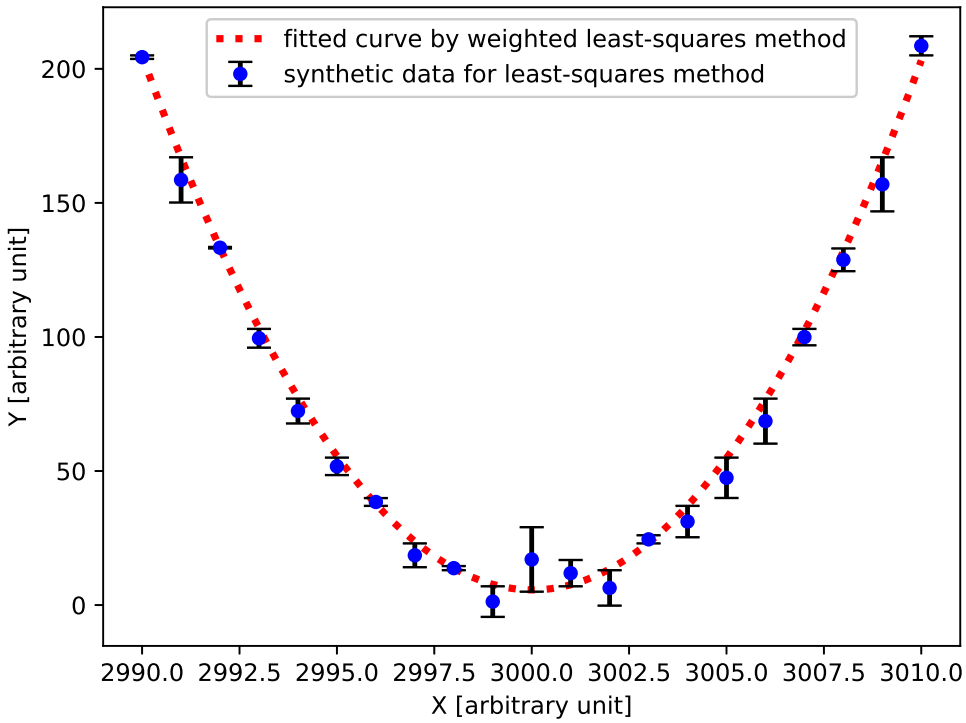

Figure 38: The result of weighted least-squares method to fit the data to a curve.

Try following practice.

Practice 05-33

Set the initial guess for coefficients, and carry out weighted least-squares method using `curve_fit`.

11.6 Least-squares method for real data

Download following data file. The data file contains semimajor axis and orbital period of planets and dwarf planets in solar system.

- data file: `https://s3b.astro.ncu.edu.tw/ai_202209/data/solsys.data`

Plot the data. Here is an example.

Python Code 57: ai202209_s05_56.py

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

#
# Time-stamp: <2022/10/17 02:14:49 (CST) daisuke>
#

# importing numpy module
```

```
import numpy

# importing scipy module
import scipy.optimize
import scipy.stats

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

# input file name
file_input = 'solsys.data'

# output file name
file_output = 'ai202209_s05_56.png'

# numpy arrays for storing data
data_a = numpy.array ([])
data_p = numpy.array ([])

# 1 au in km
au = 1.49597871 * 10**8

# 1 year in sec
year = 365.25 * 24 * 3600

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
            continue
        # splitting line into "x", "y", and "err"
        (name, a_km_str, period_sec_str) = line.split ()
        # converting string into float
        try:
            a_km = float (a_km_str)
        except:
            print (f'cannot convert "{a_km_str}" into float.')
            sys.exit (1)
        try:
            period_sec = float (period_sec_str)
        except:
            print (f'cannot convert "{period_sec_str}" into float.')
            sys.exit (1)
        # converting unit
        a_au = a_km / au
        period_yr = period_sec / year
        # appending data to numpy arrays
        data_a = numpy.append (data_a, a_au)
        data_p = numpy.append (data_p, period_yr)

# printing data
print (f'data_a:\n{data_a}')
print (f'data_p:\n{data_p}')

#
# making plot using Matplotlib
```

```
#

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

# axes
ax.set_xlabel ('Semimajor Axis [au]')
ax.set_ylabel ('Orbital Period [yr]')
ax.set_xscale ('log')
ax.set_yscale ('log')

# plotting data
ax.plot (data_a, data_p, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='planets and dwarf planets in solar system')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_56.py
data_a:
[3.87098356e-01 7.23328910e-01 9.99669590e-01 1.52358759e+00
 5.20332309e+00 9.57862678e+00 1.92826745e+01 3.03035949e+01
 3.95801805e+01 2.76703921e+00 2.76944953e+00 2.67057993e+00
 2.36294129e+00 4.29152012e+01 4.52600307e+01 6.81405142e+01
 5.29297708e+02]
data_p:
[2.40846399e-01 6.15193031e-01 9.99521805e-01 1.88065562e+00
 1.18637560e+01 2.96415580e+01 8.46739127e+01 1.66816209e+02
 2.49014627e+02 4.60289612e+00 4.60891169e+00 4.36431988e+00
 3.63234926e+00 2.81141484e+02 3.04495213e+02 5.62491953e+02
 1.21775024e+04]
```

Display PNG file. (Fig. 39)

```
% feh -dF ai202209_s05_56.png
```

Carry out least-squares method. Here is an example.

Python Code 58: ai202209_s05_57.py

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

#
# Time-stamp: <2022/10/17 02:17:26 (CST) daisuke>
#

# importing numpy module
import numpy

# importing scipy module
import scipy.optimize
```

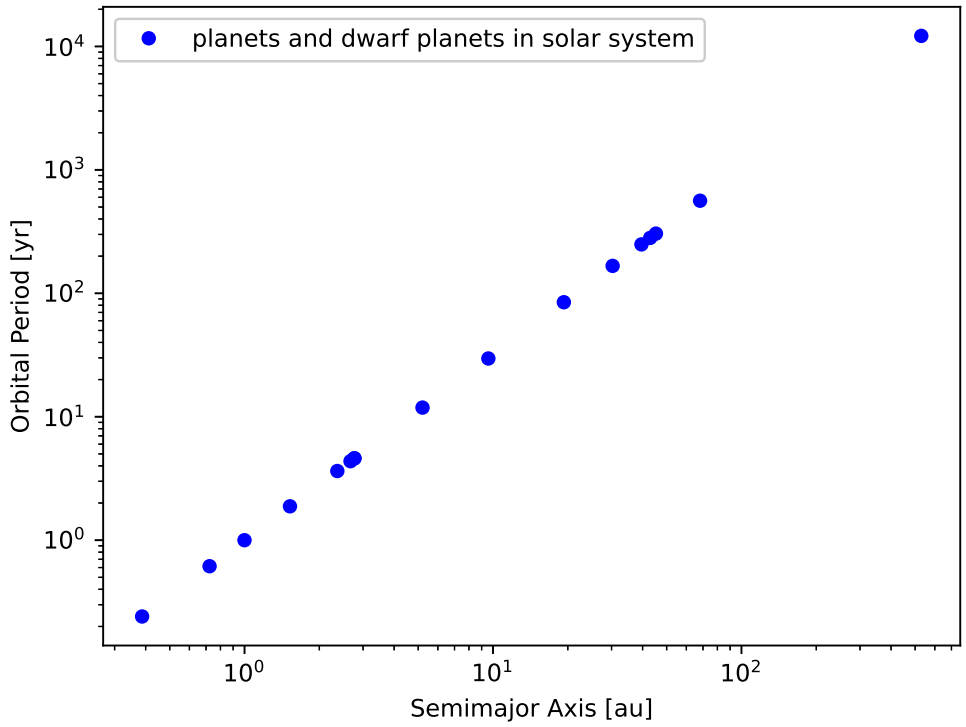

Figure 39: The semimajor axis and orbital period of planets and dwarf planets in solar system.

```
import scipy.stats

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

# input file name
file_input = 'solsys.data'

# output file name
file_output = 'ai202209_s05_57.png'

# numpy arrays for storing data
data_a = numpy.array ([])
data_p = numpy.array ([])

# 1 au in km
au = 1.49597871 * 10**8

# 1 year in sec
year = 365.25 * 24 * 3600

# opening file for reading
with open (file_input, 'r') as fh:
    # reading file line-by-line
    for line in fh:
        # skipping line, if line starts with '#'
        if (line[0] == '#'):
```

```
            continue
        # splitting line into "x", "y", and "err"
        (name, a_km_str, period_sec_str) = line.split ()
        # converting string into float
        try:
            a_km = float (a_km_str)
        except:
            print (f'cannot convert "{a_km_str}" into float.')
            sys.exit (1)
        try:
            period_sec = float (period_sec_str)
        except:
            print (f'cannot convert "{period_sec_str}" into float.')
            sys.exit (1)
        # converting unit
        a_au = a_km / au
        period_yr = period_sec / year
        # appending data to numpy arrays
        data_a = numpy.append (data_a, a_au)
        data_p = numpy.append (data_p, period_yr)

# printing data
print (f'data_a:\n{data_a}')
print (f'data_p:\n{data_p}')

# a function for straight line
def line (x, a, b):
    # line
    y = a * x + b
    # returning y
    return (y)

# initial guess of coefficients
param0 = [1.0, 1.0]

# least-squares fitting
popt, pcov = scipy.optimize.curve_fit (line, numpy.log10 (data_a), \
                                       numpy.log10 (data_p), p0=param0)

# fitted parameters and covariance matrix
print (f'popt:\n{popt}')
print (f'pcov:\n{pcov}')

# fitted a and b
a_fitted, b_fitted = popt

# degrees of freedom
dof = len (data_a) - len (popt)
print ("dof = %d" % (dof) )

# reduced chi-squared
residual     = data_p - line (data_a, a_fitted, b_fitted)
reduced_chi2 = (residual**2).sum () / dof
print (f'reduced chi-squared = {reduced_chi2}')

# fitted a and b
a_err, b_err = numpy.sqrt ( numpy.diagonal (pcov) )
print (f'a = {a_fitted:8.5f} +/- {a_err:8.5f} ({a_err/a_fitted*100.0:8.5f}%)')
print (f'b = {b_fitted:8.5f} +/- {b_err:8.5f} ({b_err/b_fitted*100.0:8.5f}%)')
```

```
# range of data
x_min = scipy.stats.tmin (data_a)
x_max = scipy.stats.tmax (data_a)

# fitted curve
fitted_x = numpy.linspace (x_min, x_max, 1000)
fitted_y = 10**line (numpy.log10 (fitted_x), a_fitted, b_fitted)

#
# making plot using Matplotlib
#

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

# axes
ax.set_xlabel ('Semimajor Axis [au]')
ax.set_ylabel ('Orbital Period [yr]')
ax.set_xscale ('log')
ax.set_yscale ('log')

# plotting data
ax.plot (data_a, data_p, linestyle='None', marker='o', markersize=5.0, \
         color='blue', label='planets and dwarf planets in solar system')
ax.plot (fitted_x, fitted_y, linestyle=':', linewidth=3.0, \
         color='red', label='fitted line by least-squares method')

# legend
ax.legend ()

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

Execute above script.

```
% ./ai202209_s05_57.py
data_a:
[3.87098356e-01 7.23328910e-01 9.99669590e-01 1.52358759e+00
 5.20332309e+00 9.57862678e+00 1.92826745e+01 3.03035949e+01
 3.95801805e+01 2.76703921e+00 2.76944953e+00 2.67057993e+00
 2.36294129e+00 4.29152012e+01 4.52600307e+01 6.81405142e+01
 5.29297708e+02]
data_p:
[2.40846399e-01 6.15193031e-01 9.99521805e-01 1.88065562e+00
 1.18637560e+01 2.96415580e+01 8.46739127e+01 1.66816209e+02
 2.49014627e+02 4.60289612e+00 4.60891169e+00 4.36431988e+00
 3.63234926e+00 2.81141484e+02 3.04495213e+02 5.62491953e+02
 1.21775024e+04]
popt:
[ 1.50000200e+00 -1.07162561e-05]
pcov:
[[ 2.45839388e-10 -2.20153496e-10]
 [-2.20153496e-10  3.61882500e-10]]
dof = 15
reduced chi-squared = 8663613.406599611
a =  1.50000 +/-  0.00002 ( 0.00105%)
```

```
b = -0.00001 +/-  0.00002 (-177.51731%)
```

The gradient is estimated to be 1.5 as expected from Kepler's third law.
Display PNG file. (Fig. 40)

```
% feh -dF ai202209_s05_57.png
```

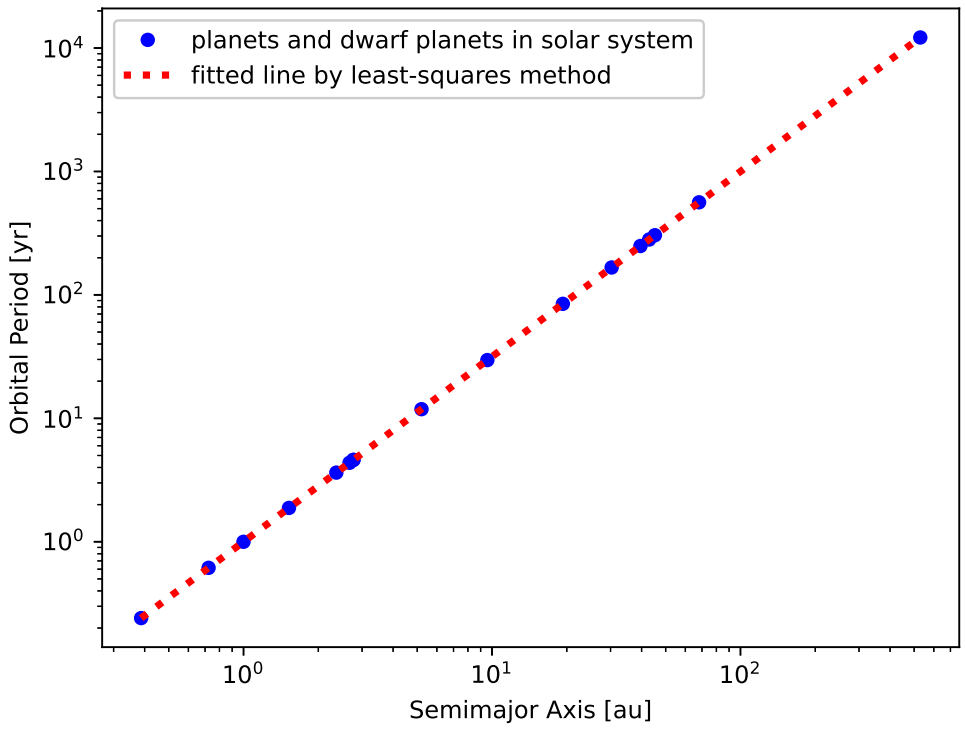

Figure 40: The semimajor axis and orbital period of planets and dwarf planets in solar system. From the least-squares method, the gradient is estimated to be 1.5 which is consistent with Kepler's third law.

Try following practice.

Practice 05-34

Download real data, and carry out weighted least-squares method using `curve_fit`.

12 For your further reading

Read the official document of SciPy to learn more about it.

- SciPy: `https://docs.scipy.org/doc/scipy/`
  - SciPy User Guide: `https://docs.scipy.org/doc/scipy/tutorial/`
  - SciPy API Reference: `https://docs.scipy.org/doc/scipy/reference/`
    - Constants: `https://docs.scipy.org/doc/scipy/reference/constants.html`
    - Integration and ODEs: `https://docs.scipy.org/doc/scipy/reference/integrate.html`
    - Interpolation: `https://docs.scipy.org/doc/scipy/reference/interpolate.html`
    - Linear Algebra: `https://docs.scipy.org/doc/scipy/reference/linalg.html`
    - Optimization and Root Finding: `https://docs.scipy.org/doc/scipy/reference/optimize.html`
    - Statistical Functions: `https://docs.scipy.org/doc/scipy/reference/stats.html`

13 Assignment

1. Statistical values
   (a) What is moment?
   (b) What is skewness?
   (c) What is kurtosis?
2. Study about Runge-Kutta method.
   - Describe Runge-Kutta method.
   - Use Runge-Kutta method to simulate a ball thrown to the air.
3. Study about least-squares method.
   (a) Describe the least-squares method.
   (b) What is degree of freedom?
   (c) What is χ^2?
   (d) What is reduced χ^2?
   (e) Describe the way to calculate the errors of fitted coefficients?
4. Atmospheric extinction coefficient.
   (a) What is atmospheric extinction coefficient?
   (b) Make a Python script to download following file. Show the source code of your Python script.
      - `http://s3b.astro.ncu.edu.tw/ai_202102/data/f16_v.data`
   (c) Downloaded file contains V-band photometric measurements of the star F16. Make a airmass vs. instrumental magnitude plot. Show the source code of your Python script.
   (d) Make a Python script to carry out least-squares fitting of the atmospheric extinction profile of the star. Show the plot of atmospheric extinction profile and fitted line. What is the obtained value of atmospheric extinction coefficient? Show the source code of your Python script. Assume linear profile for the extinction.
   (e)
5. Radial profile of a star.
   (a) Make a Python script to download following file. Show the source code of your Python script.
      - `http://s3b.astro.ncu.edu.tw/ai_202209/data/radprof.data`
   (b) Make a Python script to plot the data. Show the source code of your Python script.
   (c) Make a Python script to carry out least-squares fitting of the radial profile of the star. Show the plot of radial profile and fitted curve. Show the source code of your Python script. Assume Gaussian profile for the PSF.
   (d) Make a Python script to carry out least-squares fitting of the radial profile of the star. Show the plot of radial profile and fitted curve. Show the source code of your Python script. Assume Moffat profile for the PSF.
6. Age of the solar system.
   (a) What is a meteorite?
   (b) What is a chondrite?
   (c) The age of the meteorite can be estimated using the radioactive decay of nuclides. The probability of the decay does not depend on the age of the nuclide. Derive following relation,

$$N(t) = N_0 e^{-\lambda(t-t_0)}, \tag{29}$$

   where $N(t)$ is the abundance of a parent species at time t, N_0 is the abundance of a parent species at time $t = t_0$, and λ is the decay constant.
   (d) Show that the mean lifetime τ_m can be written as

$$\tau_m = \frac{1}{\lambda}. \tag{30}$$

(e) ^{87}Rb is a radioactive isotope, and ^{87}Rb decays into ^{87}Sr. ^{86}Sr is a stable isotope. Show that the current amount of ^{87}Sr $N_{87Sr/86Sr}(t)$ is expressed as

$$N_{87Sr/86Sr}(t) = N_{87Sr/86Sr}(t=0) + \left(e^{\lambda t} - 1\right) N_{87Rb/86Sr}(t), \tag{31}$$

where $N_{87Sr/86Sr}(t=0)$ is the initial amount of ^{87}Sr and $N_{87Rb/86Sr}(t)$ is the current amount of ^{87}Rb.

(f) The half-life of ^{87}Rb is $t_{1/2,^{87}Rb} = 4.88 \times 10^{10}$ yr. What is the decay constant of ^{87}Rb?

(g) Describe the method to estimate the age of meteorites using isotopic abundances data.

(h) Table 1 is the isotopic abundances of selected H chondrites. Estimate the age of the solar system using those data. Show the source code of your Python script.

(i) Table 2 is the isotopic abundances of selected LL chondrites. Estimate the age of the solar system using those data. Show the source code of your Python script.

Table 1: Rb-Sr isotopic abundances of a group of H chondrites. Data are from Kaushal and Wetherill (1969).

Chondrite Name	^{87}Rb/^{86}Sr	^{87}Sr/^{86}Sr
Buth Furnace	0.09	0.706
Kyushu	0.60	0.739
Bruderheim	0.72	0.747
Homestead	0.80	0.751
Modoc	0.86	0.757

Table 2: Rb-Sr isotopic abundances of a group of LL chondrites. Data are from Minster and Allegre (1981).

Chondrite Name	^{87}Rb/^{86}Sr	^{87}Sr/^{86}Sr
Chainpur (LL3) No.1	0.7580	0.74864
Chainpur (LL3) No.2	0.7255	0.74650
Ngawi (LL3)	0.5422	0.74107
Soko Banja (LL4) A, No. 1	1.520	0.79891
Soko Banja (LL4) A, No. 2	1.490	0.79692
Soko Banja (LL4) B, No. 1	1.555	0.80152
Soko Banja (LL4) B, No. 2	1.685	0.80952
Soko Banja (LL4) A, No. 1	1.520	0.79891
Soko Banja I No.1	0.1542	0.70910
Soko Banja I No.2	0.1533	0.70895
Guidder (LL5) A	0.4060	0.72576
Guidder (LL5) B	0.6020	0.73838
Guidder (LL5) C	0.6014	0.73837
Olivenza (LL5)	0.7790	0.75035
Manbhoom (LL6)	0.5600	0.73570
Douar Mghila (LL6)	0.6291	0.74044
Siena (LL6) “clear”	0.6701	0.74259
Siena (LL6) “black”	0.8670	0.75655
Saint Severin (LL6) A, No. 1	0.1057	0.70583
Saint Severin (LL6) A, No. 2	0.1069	0.70606
Saint Severin (LL6) B, No. 1	0.1610	0.70941
Saint Severin (LL6) B, No. 2	0.1621	0.70952
Ensisheim (LL6) A, No. 1	0.0313	0.70149
Ensisheim (LL6) A, No. 2	0.0278	0.70184
Ensisheim (LL6) B	0.0409	0.70237
Ensisheim (LL6) C, No. 1	0.0525	0.70214
Ensisheim (LL6) C, No. 2	0.0525	0.70218