

Computational Materials Science (計算材料学特論)

Lecture materials updated

http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms

2025年度Q2 計算材料学特論 (資料: 英語 + 日本語版)

COMPUTATIONAL MATERIALS SCIENCE 2025 Q2

数値解析に関する講義資料・pythonプログラム (神谷担当分)

Lecture materials for numerical analyses (by Kamiya)

講義で使うプレゼン資料は共通して使うpythonプログラム」の下にあります

Lecture presentation slides are found after the "Common python programs" section below.

Update News:

- June 12, 10:32 Lecture materials on June 13 have been uploaded ([20250612DifferentialIntegration.zip](#))
- June 10, 11:46 Final version of lecture materials on June 10 have been updated ([20250610ComputerAndErrorSources.zip](#))

We will wait for five minutes.

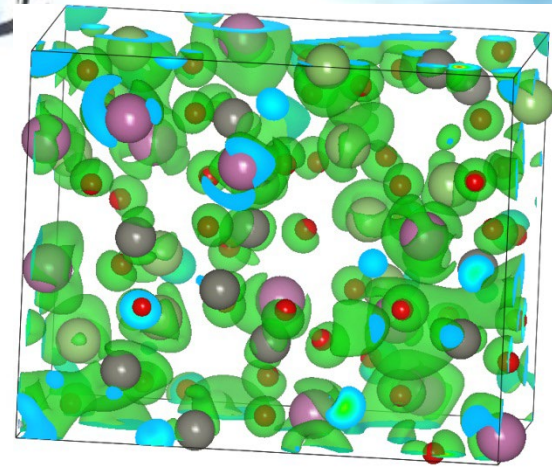
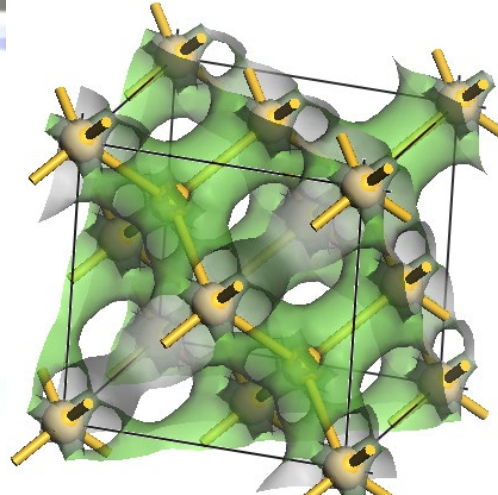
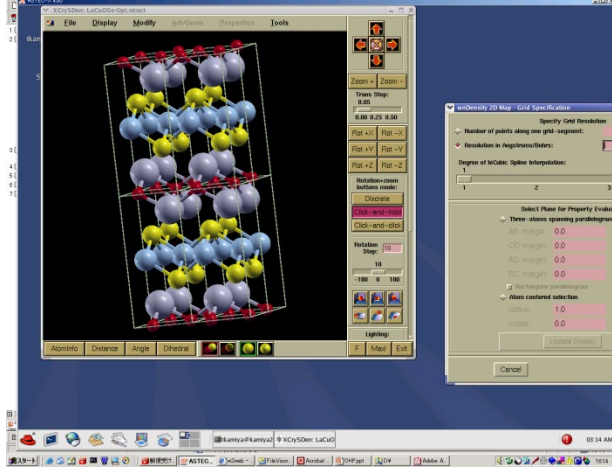
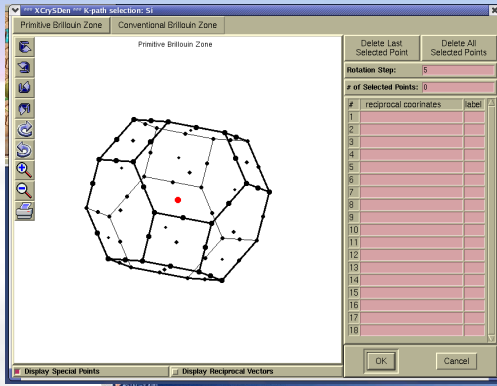
In the meantime, please make sure to download the lecture materials

本講義では、pythonは必須ではありませんが、アルゴリズムの理解と今後の研究に役に立ちますので、余裕のある人は試してみてください。
python is not a requirement for this class, but it will help your understanding about the algorithms to be learned and also assist your future research.

- [python programming](#) (Japanese)

計算材料学特論

Toshio Kamiya
神谷利夫



Class Schedule

Lecture materials (Kamiya's part): http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/D2MatE_programs.html?page=cms

- #01 June 10 (Tue) Kamiya (Fundamentals of computer, Sources of errors (コンピュータの基礎、誤差))
- #02 June 13 (Fri) Kamiya (Numerical differentiation/integration (数値微分/積分))
- #03 June 17 (Tue) Kamiya (Differential equation (微分方程式), Molecular dynamics (分子動力学法),
Interpolation (補間), Smoothing (平滑化))
- #04 June 20 (Fri) Kamiya (Linear least-squares method (線形最小二乗法), Optimization (最適化),
Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化))
- #05 June 24 (Tue) Not decided (to be canceled or video lecture without mandatory attendance)
- #06 June 27 (Fri) Kamiya (Nonlinear optimization (非線形最適化),
Fourier transformation (フーリエ変換))
- #07 July 1 (Tue) Kamiya, Matrix (行列)
- #08 July 4 (Fri) Sasagawa (Review of quantum theory 1: 量子論おさらい1)
- #09 July 8 (Tue) Sasagawa (Review of quantum theory 2: 量子論おさらい2)
- #10 July 11 (Fri) Sasagawa (First principles calculations: basics 1 第一原理計算:基礎1)
- #11 July 15 (Tue) Sasagawa (First principles calculations: basics 2 第一原理計算:基礎2)
- #12 July 18 (Fri) Sasagawa (First principles calc.: applications 1 第一原理計算:応用1)
- #13 July 22 (Tue) Sasagawa (First principles calc.: applications 2 第一原理計算:応用2)
- #14 July 25 (Fri) Sasagawa (Classical and Quantum Computers 古典および量子コンピュータ)

Explanation of the answers

課題解答の解説

PROBLEM, June 13

- **Submit electronic file(s) via LMS until June 15**
(If LMS doesn't work, send the files to kamiya.t.aa@m.titech.ac.jp.
In this case, file name must include your STUDENT ID and FULL NAME)

PROBLEM:

- Calculate $dE(k)/dk$, $d^2E(k)/dk^2$, and effective mass m_e^*/m_0 from $E(k)$ in band.xlsx, and plot m_e^*/m_0 vs k .
Assume the lattice parameter is $a = 4.0 \text{ \AA}$.**
- Compare the results obtained by different h .**

See band_answer.xlsx

Effective mass

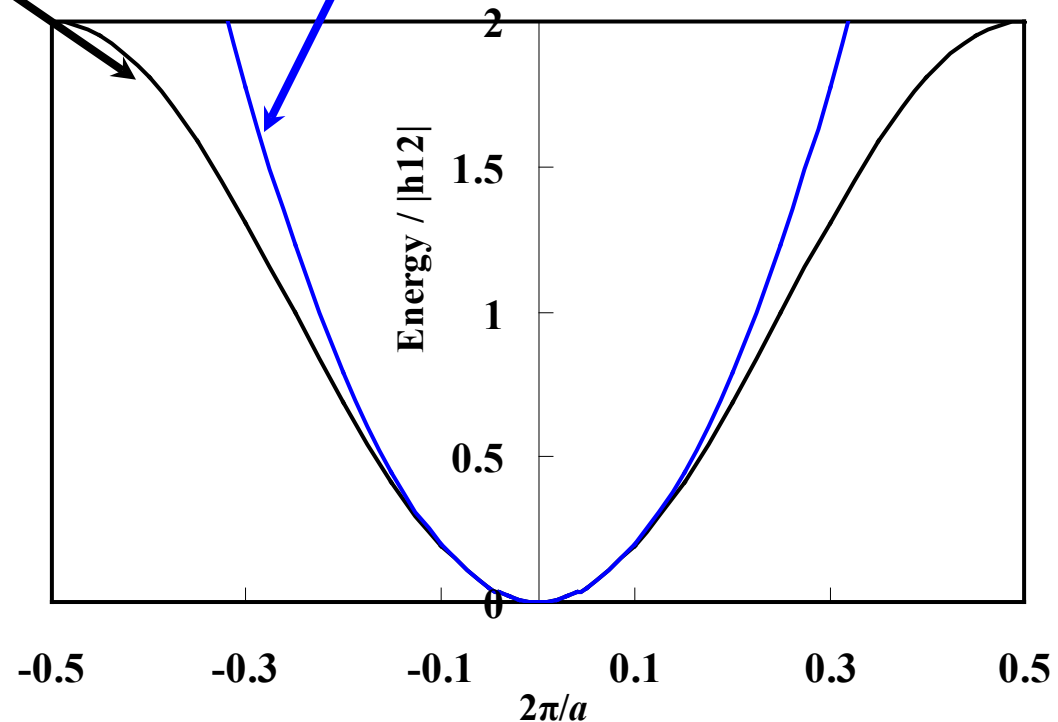
LCAO band

$$E(k) = \varepsilon_1 - 2|h_{12}|\cos(ka) \sim \varepsilon_1 - 2|h_{12}| + |h_{12}|a^2k^2 + O((ka)^4)$$

Free electron model

$$E(k) = E_0 + \frac{|\mathbf{P}|^2}{2m} = E_0 + \frac{\hbar^2}{2m}|\mathbf{k}|^2$$

$$\frac{1}{m^*} = \frac{1}{\hbar^2} \frac{\partial^2 E_n(\mathbf{k})}{\partial k^2}$$



Effective mass

k represents fractional coordinate in reciprocal unit cell:

generally expressed in the range $[-\frac{1}{2} \frac{1}{2}]$

Unit conversion $k_{\text{real}} = (2\pi/a)k$

Note $E(k)$ is in eV

$$m^* = \hbar^2 \left(\frac{\partial^2 E_J(\mathbf{k})}{\partial k_{\text{real}}^2} \right)^{-1} = \hbar^2 \left(\frac{2\pi}{a} \right)^2 \left(\frac{\partial^2 E_{eV}(\mathbf{k})}{\partial k^2} e \right)^{-1}$$

Very often effective mass is given by a ratio to the electron rest mass m_e^0 .

$$m^*/m_e^0 = \hbar^2 \left(\frac{\partial^2 E_J(k)}{\partial k_{\text{real}}^2} \right)^{-1} / m_e^0 = \hbar^2 \left(\frac{2\pi}{a} \right)^2 \left(\frac{\partial^2 E_{eV}(k)}{\partial k^2} e \right)^{-1} / m_e^0$$

Numerical differentiation: Accuracy

$$\frac{df(x)}{dx} \sim \frac{f(x+h) - f(x)}{h}$$

Error: $\frac{f(x+h) - f(x)}{h} = \frac{df(x)}{dx} + \boxed{\frac{1}{2} \frac{d^2 f(x)}{dx^2} h + \frac{1}{3!} \frac{d^3 f(x)}{dx^3} h^2 + O(h^4)}$

$$\frac{df(x)}{dx} \sim \left[\frac{f(x+h) - f(x)}{h} + \frac{f(x) - f(x-h)}{h} \right] / 2 = \frac{f(x+h) - f(x-h)}{2h}$$

$$\begin{aligned} f(x+h) &= f(x) + \frac{df(x)}{dx} h + \frac{1}{2} \frac{d^2 f(x)}{dx^2} h^2 + \frac{1}{3!} \frac{d^3 f(x)}{dx^3} h^3 + O(h^4) \end{aligned}$$

$$\begin{aligned} f(x-h) &= f(x) - \frac{df(x)}{dx} h + \frac{1}{2} \frac{d^2 f(x)}{dx^2} h^2 - \frac{1}{3!} \frac{d^3 f(x)}{dx^3} h^3 + O(h^4) \end{aligned}$$

Error: $\frac{f(x+h) - f(x-h)}{2h} = \frac{df(x)}{dx} + \boxed{\frac{1}{3!} \frac{d^3 f(x)}{dx^3} h^2 + O(h^4)}$

Second differential (二階微分)

If calculate 2nd differential using forward differences with the 1st and the 2nd differentials ... (一階微分を前身差分で計算してから二階微分を前進差分で計算すると・・・)

$$\begin{aligned}\frac{d^2x(t)}{dt^2} &= \frac{\frac{dx}{dt}(t + \Delta t) - \frac{dx}{dt}(t)}{\Delta t} \\ &\sim \frac{\frac{x(t+2\Delta t) - x(t+\Delta t)}{\Delta t} - \frac{x(t+\Delta t) - x(t)}{\Delta t}}{\Delta t} = \frac{x(t+2\Delta t) - 2x(t+\Delta t) + x(t)}{\Delta t^2}\end{aligned}$$

Use central difference

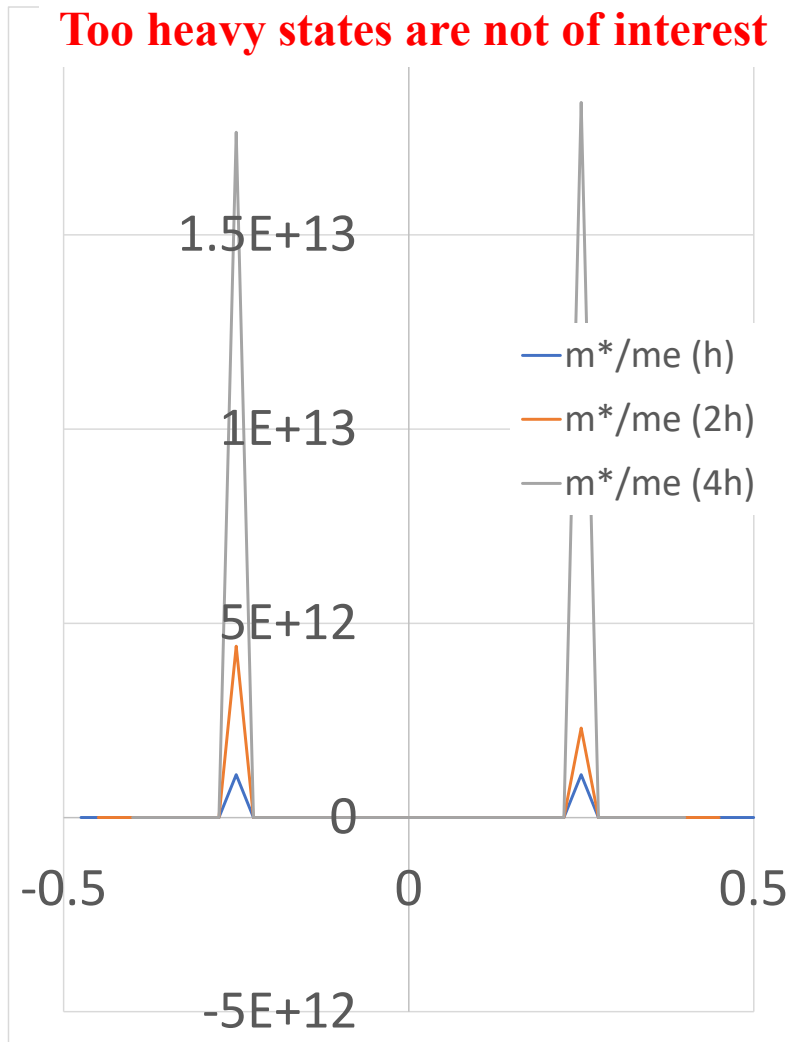
$$\begin{aligned}\frac{d^2x(t)}{dt^2} &= \frac{\frac{dx}{dt}(t + \Delta t/2) - \frac{dx}{dt}(t - \Delta t/2)}{\Delta t} \\ &\sim \frac{\frac{x(t+\Delta t) - x(t)}{\Delta t} - \frac{x(t) - x(t-\Delta t)}{\Delta t}}{\Delta t} = \frac{x(t+\Delta t) - 2x(t) + x(t-\Delta t)}{\Delta t^2}\end{aligned}$$

Note: These two formula are offset in t by Δt
2つの式では、横軸が Δt ずれるので注意！

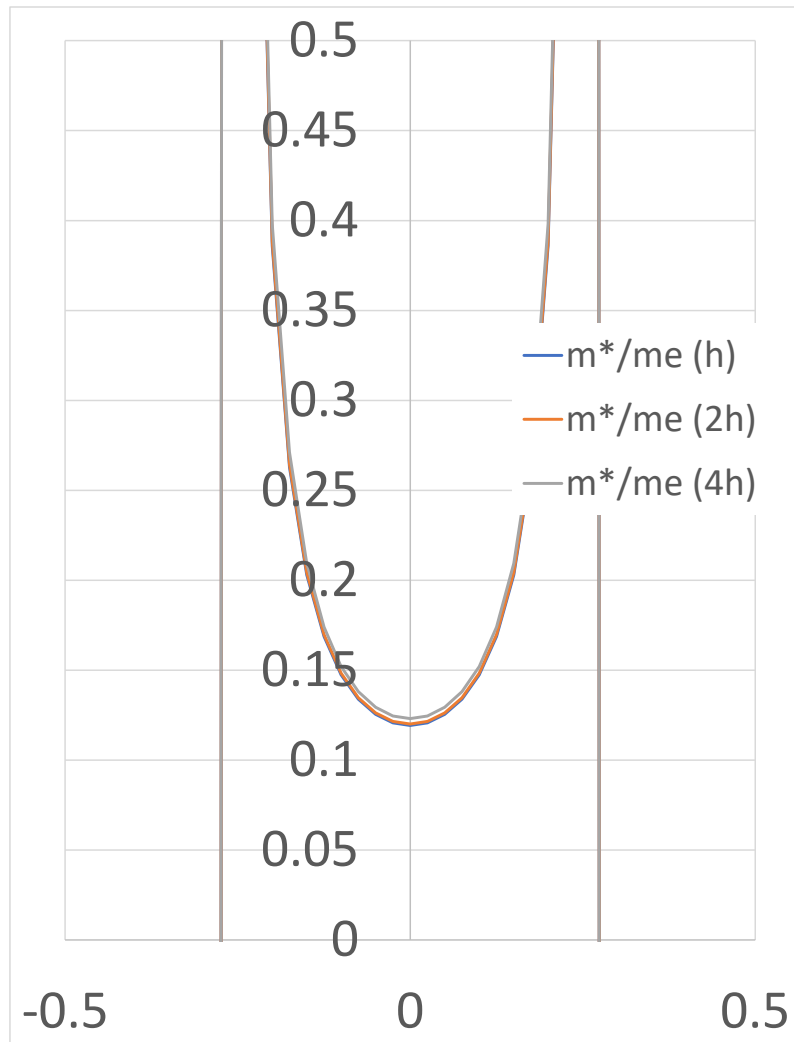
Answer: How to present?

band-answer.xlsx

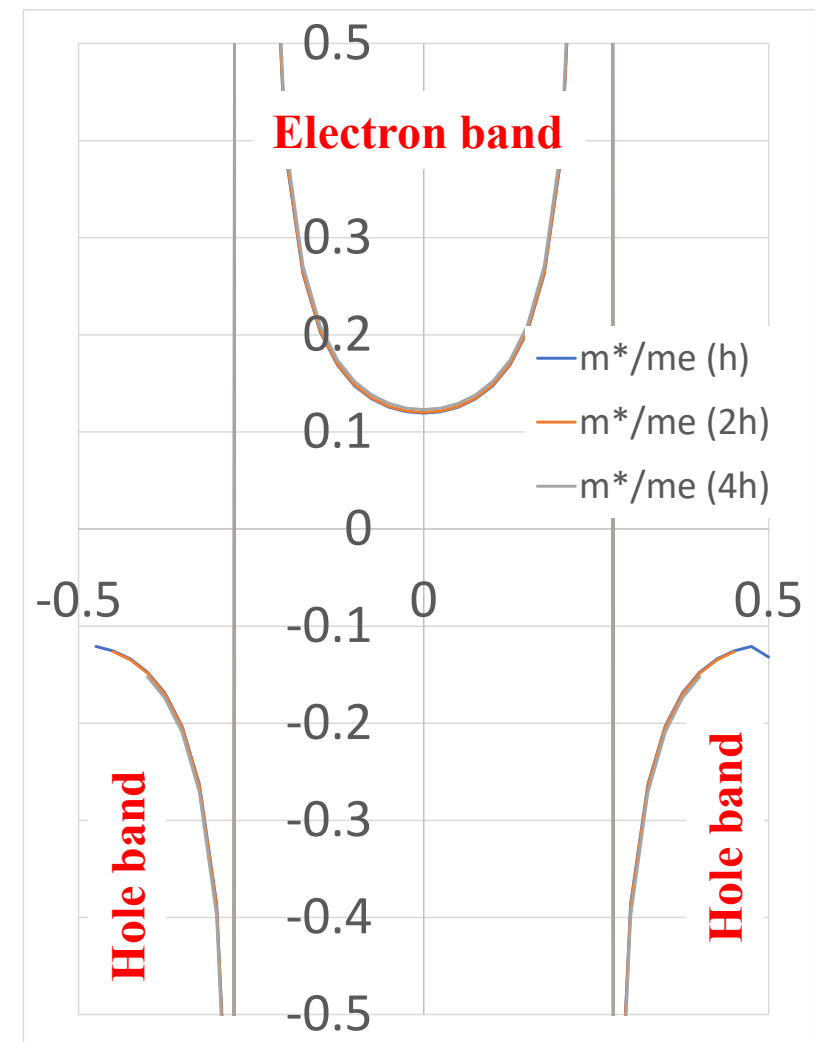
Full range



Electron only ($m^* > 0$)

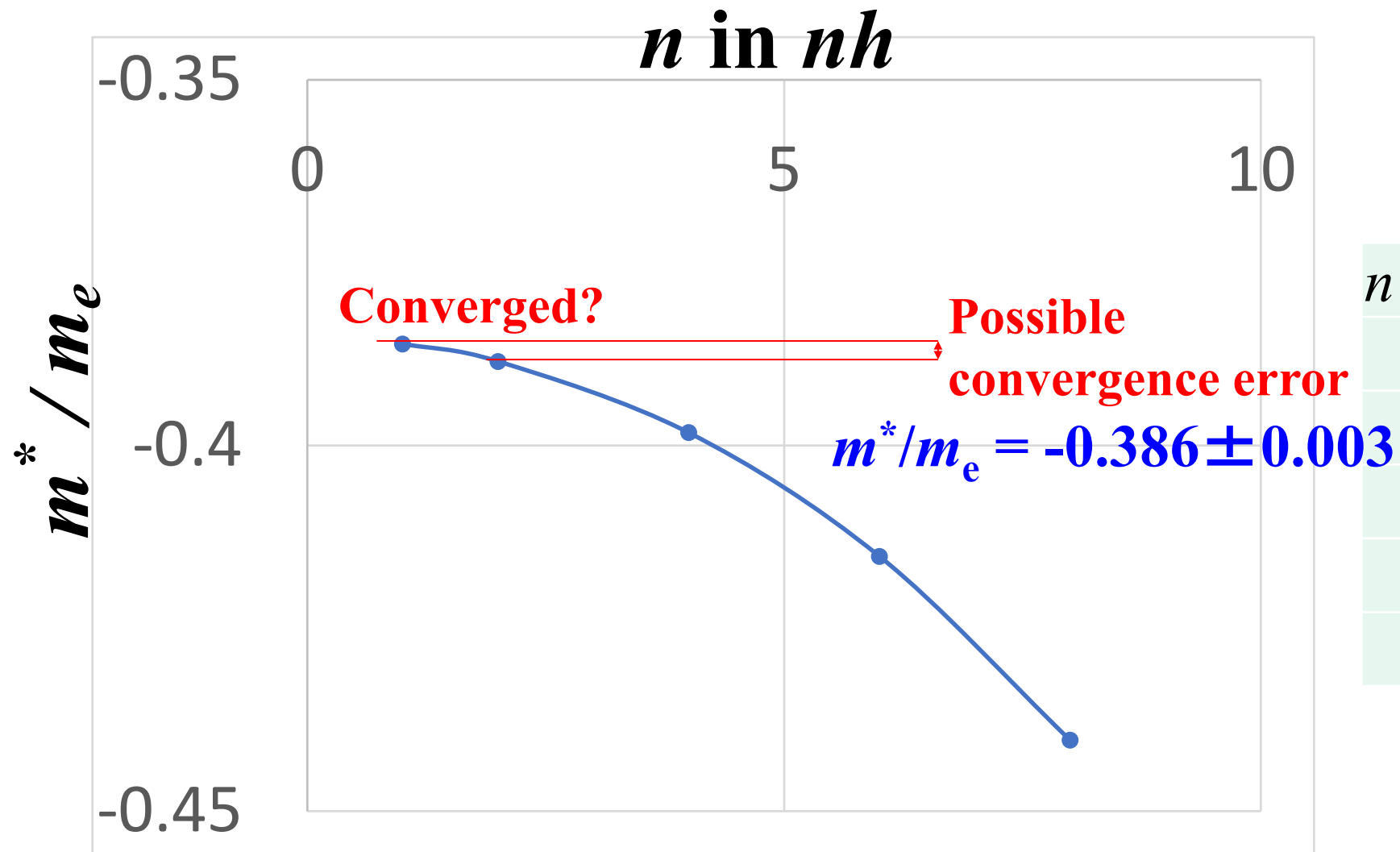


Electron and holes



Accuracy and convergence check

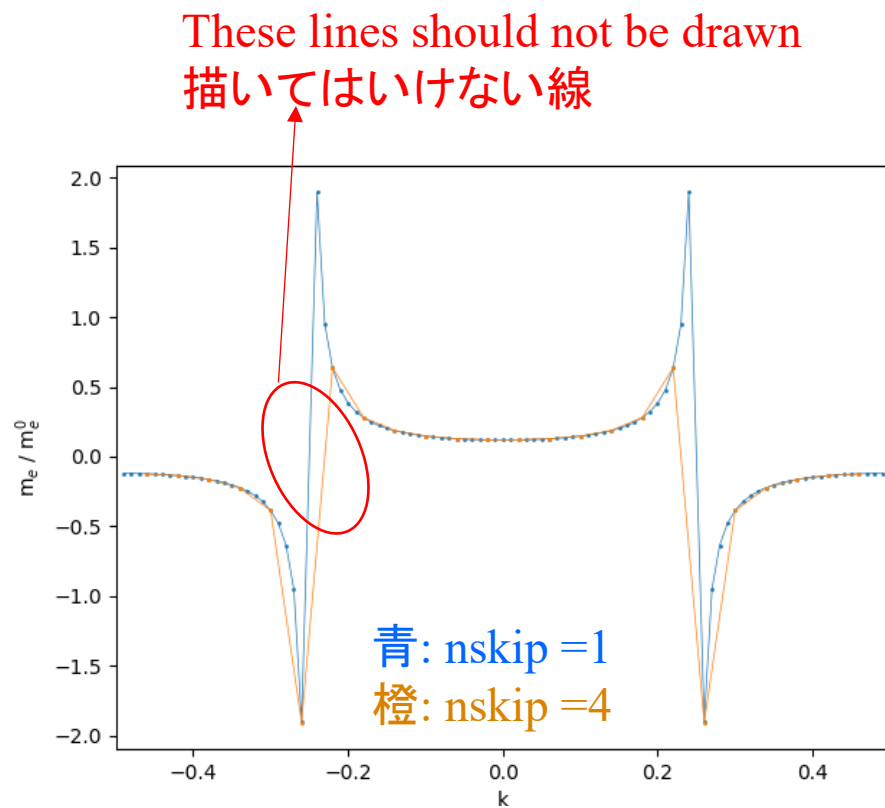
band-answer.xlsx



| <i>n</i> in <i>nh</i> | <i>m</i> [*] / <i>m</i> _e |
|-----------------------|---|
| 1 | -0.386097987 |
| 2 | -0.388489462 |
| 4 | -0.398234965 |
| 6 | -0.415138589 |
| 8 | -0.440277749 |

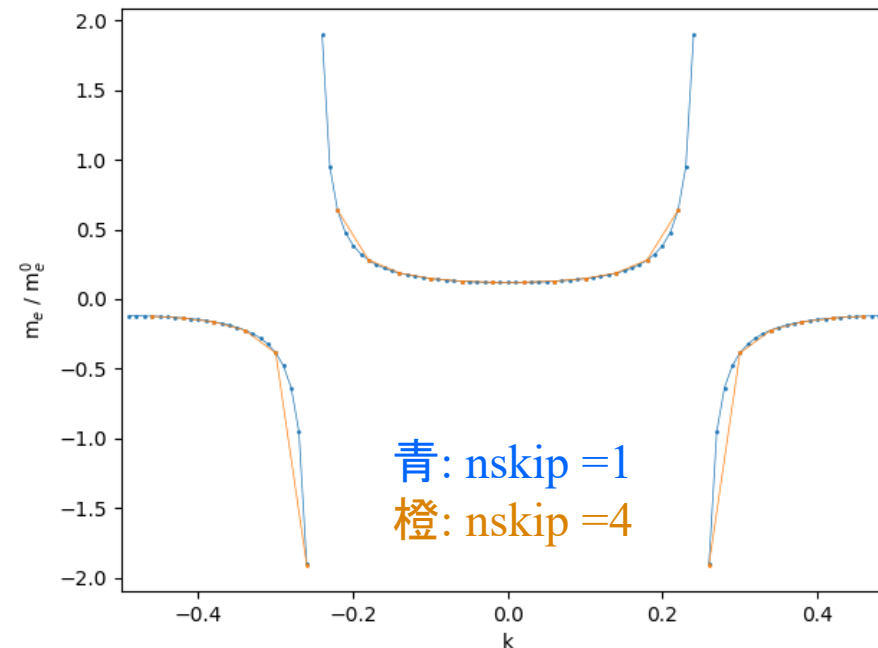
Python program (抜粋)

python EffectiveMass.py



Data points can be disconnected by inserting
None values

データに None (未定義値) を挿入することで
描いてはいけない線を消した



Python program (抜粋)

EffectiveMass.py

#共通の定数は先に計算

```
km = hbar * hbar * (pi2 / a)**2.0
```

#微分の精度を比較するため、h = nskip*dk にする

```
nskip = 1
```

```
xk = []
```

```
ymc = []
```

#符号の変化を検出するため、符号変数を用意

```
signprev = None
```

```
for i in range(nskip, nk - nskip, nskip):
```

#2階微分を計算

```
    d2Edk2c = (E[i+nskip] + E[i-nskip] - 2 * E[i]) * e / pow(nskip *  
dk, 2.0)
```

#2回微分はゼロになることがあるので、まずは1/m*を計算

```
    minv = d2Edk2c / km
```

```
    print(i, E[i-1], E[i], E[i+1], minv)
```

#1/m*が1/meより非常に小さければ、m*は計算しない

```
    if abs(minv) <= 1.0e20: # << 1.0/me ~ 1e30
```

#符号が反転する場所でグラフの線を切断するときは
#Noneデータを追加する。

```
        if cutline:
```

```
            xk.append(k[i])
```

```
            ymc.append(None)
```

#反転した符号を記録

```
            signprev = -signprev
```

```
            continue
```

```
    else:
```

```
        m = km / d2Edk2c
```

#符号が反転する場所でグラフの線を切断するときは
#Noneデータを追加する。

```
    if signprev is None:
```

#signprevが 初期値 None である場合は 符号の最初の値を代入

```
        signprev = m
```

```
    elif signprev * m < 0.0:
```

```
        if cutline:
```

```
            xk.append(k[i])
```

```
            ymc.append(None)
```

#反転した符号を記録

```
            signprev = m
```

```
        xk.append(k[i])
```

```
        ymc.append(m / me)
```

```
plt.plot(xk, ymc, linewidth = 0.5, marker = 'o', markersize = 1.0,  
label = 'nskip = 1')
```

```
plt.xlabel(klabel)
```

```
plt.ylabel("m$_e$ / m$_e^{0}$")
```

```
plt.xlim([-0.5, 0.5])
```

```
# plt.ylim([-0.5, 0.5])
```

```
plt.tight_layout()
```

```
plt.pause(0.1)
```

```
print("Press ENTER to exit>>", end = ")
```

```
input()
```

```
if __name__ == "__main__":
```

```
    main()
```

Read Excel file: openpyxl module

See <http://conf.msl.titech.ac.jp/D2MatE/2023Tutorial/tutorial2023-python-ChatGPT.html>
<http://conf.msl.titech.ac.jp/D2MatE/2023Tutorial/python-tutorial2023-V5.zip>

```
import openpyxl
```

```
# 1. Read data from Excel file
```

```
workbook = openpyxl.load_workbook(input_path) # Open Excel file input_path
```

```
sheet = workbook.active # Assign current worksheet to sheet variable
```

```
T = [] # Initialize T and N lists to read Excel data
```

```
N = []
```

```
for row in sheet.iter_rows(min_row=2, values_only=True):
```

```
    # get row list variable from each row after row# min_row
```

```
    T.append(row[0]) # add data by.append() method
```

```
    N.append(row[2])
```

```
# .iter_rows() returns None when it reaches the last row: iterator
```

Read Excel/CSV file: Easier by pandas

- Pandas:**
- Easy read Excel, CSV, and text files with a table format
 - Often used combined with machine-learning libraries like scikit-learn
 - Array is provided by **DataFrame** type. Data come with labels (columns) and indexes
 - NOTE: DataFrame is “row-like (行優先)”

```
import pandas as pd
```

```
Ex 1: df = pd.read_csv(input_path) # Read DataFrame var df from CSV file
```

```
Ex 2: df = pd.read_excel(input_path) # Read DataFrame var df from Excel file
```

```
Ex A: d = pd.to_dict() # Convert to a dictional variable
```

```
Ex B: labels = df.columns.to_numpy() # Convert to numpy.ndarray
```

```
data_list = df.to_numpy().T # Conver to 2 dimensional numpy.ndarray
```

```
# df.to_numpy() 2D array is of row-like (行優先)
```

```
# data_list[0] corresponds to second row data in Excel file
```

```
# To extract T and N 1D list (column-like, 列優先), take transpose by .T method
```

```
T = data_list[0] # get T and N vars from data_list
```

```
N = data_list[1]
```


PROBLEM, June 17

- **Submit electronic file(s) via LMS until the midnight of June 18**
(If LMS doesn't work, send the files to kamiya.t.aa@m.titech.ac.jp.
In this case, file name must include your STUDENT ID and FULL NAME)

PROBLEM:

- (i) By filling the dx/dt and the $x(t)$ columns in diffeq.xlsx, **solve $dx(t) / dt = -x(t)\sin(\pi t)$ using the Euler method.**

Conditions:

t starts from 0 and ends at 3.0 with the time step of 0.1.

$$x(0) = 1.0$$

Numerical solutions of differential equations

微分方程式の数値解法

Motion of planets – Analytical solution

(惑星の運動 – 解析解)

$$m \frac{d^2 \mathbf{r}}{dt^2} = -G \frac{mM}{r^2} \frac{\mathbf{r}}{r} \quad mr^2 \frac{d\theta}{dt} = l \quad l: \text{a constant, conservation of angular momentum}$$
$$\frac{1}{2} m \left(\frac{dr}{dt} \right)^2 + m \left(\frac{l^2}{2m^2 r^2} - \frac{GM}{r} \right) = E$$
$$r(\theta) = \frac{b}{1 + \varepsilon \cos(\theta - \delta)} \quad b = \frac{l^2}{mc} \quad \varepsilon = \sqrt{1 + 2El^2 / mc^2}$$

Elliptic equations (楕円方程式)

Long radius of ellipse

$$a' = 2b / (1 - \varepsilon^2)$$

Short radius of ellipse

$$b' = 2b / \sqrt{1 - \varepsilon^2}$$

Eccentricity (離心率) 焦点間の距離/長径

$$\varepsilon = \sqrt{1 + 2El^2 / mc^2}$$

Close distance point (近点距離)

$$q = a'(1 - e) = b / (1 + \varepsilon)$$

Long distance point (遠点距離)

$$Q = a'(1 + e) = b / (1 - \varepsilon)$$

Period (周期)

$$T = 2\pi \sqrt{ma^3 / c}$$

Normalization of equation

(方程式の規格化/無次元化)

$$m \frac{d^2 \mathbf{r}}{dt^2} = -G \frac{mM}{r^2} \frac{\mathbf{r}}{r}$$



Convert variables to T and R by representative constants τ_0 and l_0

$$t = \tau_0 T \quad r = l_0 R$$

τ_0, l_0 : Time and length specific to the system
Chose so that T and R will be the order of 1.0

$$m \frac{l_0}{\tau_0^2} \frac{d^2 \mathbf{R}}{dT^2} = -G \frac{1}{l_0^2} \frac{mM}{R^2} \frac{\mathbf{R}}{R}$$

E.g., for planet simulation

τ_0 = Revolution / Rotation period
(公転 / 自転周期)

l_0 = Revolution radius, Astronomy unit

for molecular dynamics (MD)

τ_0 = MD time step

l_0 = Bohr radius (atomic unit)

$$\frac{d^2 \mathbf{R}}{dT^2} = -G' \frac{mM}{R^2} \frac{\mathbf{R}}{R}$$

$$G' = \frac{G \tau_0^2}{l_0^3}$$

First-order diff. eq. : Euler formula (オイラー法)

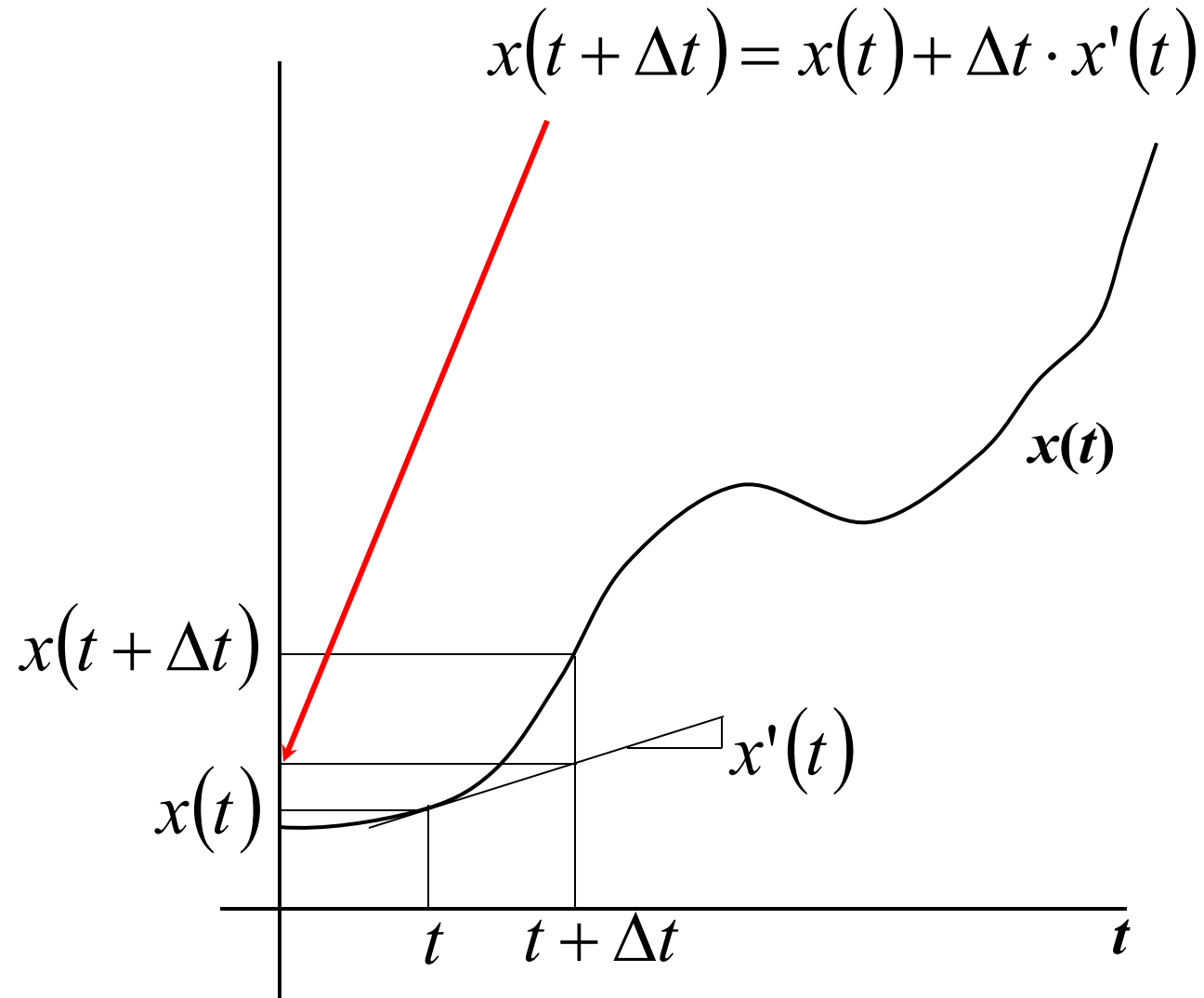
$$\frac{dx}{dt} = f(x, t)$$

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = f(t, x(t))$$

$$x(t + \Delta t) = x(t) + \Delta t \cdot f(t, x(t))$$

- **Accuracy not good**
- **Asymmetric with respect to $t, t + \Delta t$**

Illustrative image of Euler method



First-order diff. eq. : Heun formula (ホイン法)

$$\frac{dx}{dt} = f(t, x(t))$$

- **Average the Euler formula at t and $t+\Delta t$**

$$x(t + \Delta t) = x(t) + \frac{1}{2}\Delta t[f(t, x(t)) + f(t + \Delta t, x(t + \Delta t))]$$

Problem: $x(t+\Delta t)$ and $f(t+\Delta t, x(t+\Delta t))$ are unknown

=> Use $x(t+\Delta t)$ by Euler formula

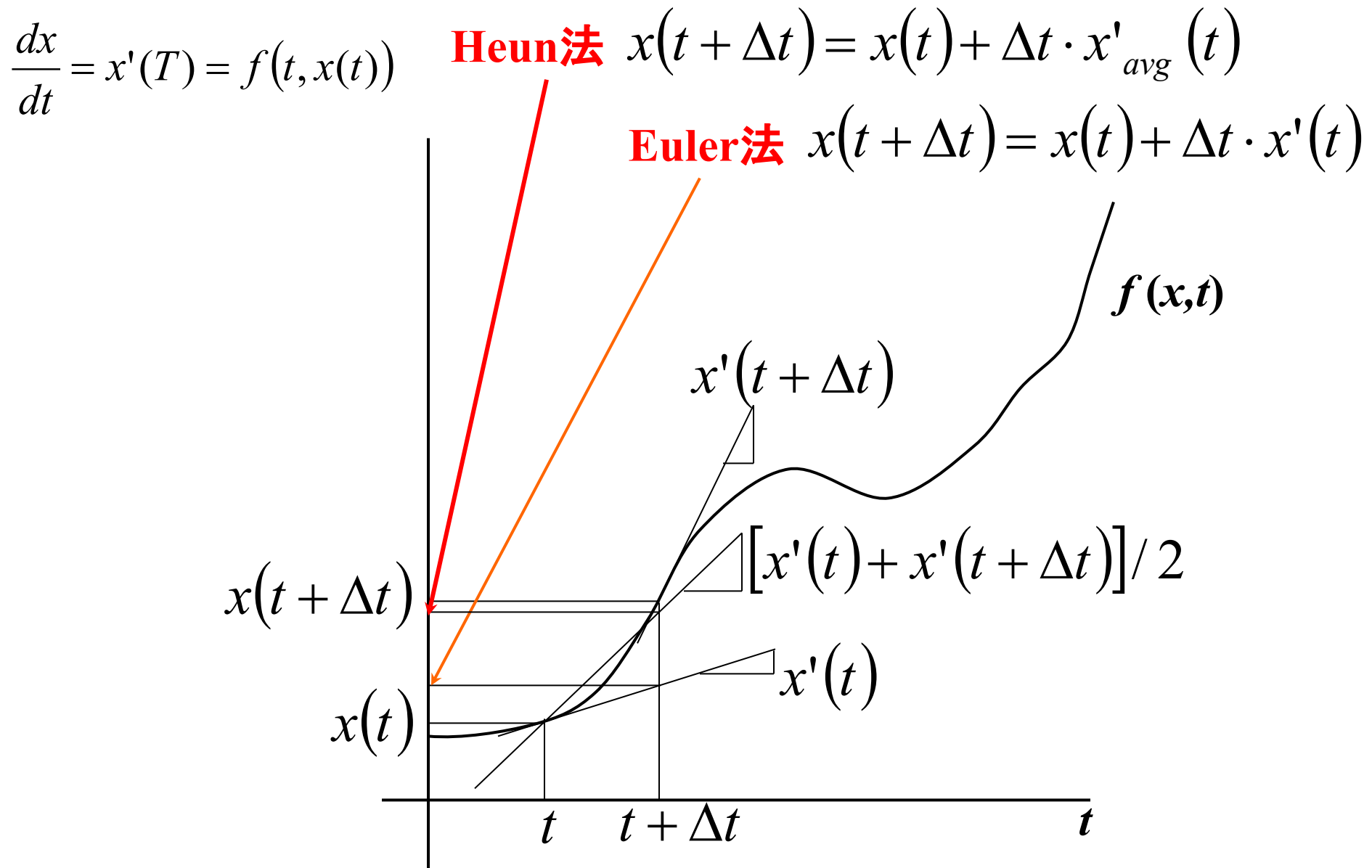
$$x(t + \Delta t) \sim x(t) + \Delta t f(t) = x(t) + k_0$$

$$k_0 = \Delta t \cdot f(t, x(t))$$

$$k_1 = \Delta t \cdot f(t + \Delta t, x(t + \Delta t)) \sim \Delta t \cdot f(t + \Delta t, x(t) + k_0)$$

$$x(t + \Delta t) = x(t) + \frac{k_0 + k_1}{2}$$

Illustrative image of Heun method



First-order differential equation

$$\frac{dx}{dt} = f(x, t)$$

Euler formula: $k_0 = \Delta t \cdot f(x(t), t)$

$$x(t + \Delta t) = x(t) + k_0$$

Heun formula: $k_0 = \Delta t \cdot f(x(t), t)$

$$k_1 = \Delta t \cdot f(x(t) + k_0, t + \Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{k_0 + k_1}{2}$$

Outline of program

```
dt = 0.01
```

```
t0 = 0.0
```

```
x0 = 1.0
```

```
# dx/dt = dxdt(t, x)
```

```
def dxdt(t, x):
```

```
    return -x*x
```

```
# Solve by the Euler formula
```

```
def diffeq_euler(diff1 func, t0, x0, dt):
```

```
    k0 = dt * diff1 func(t0, x0)
```

```
    x1 = x0 + k0
```

```
    return x1
```

```
x1 = diffeq_euler(dxdt, t0, x0, dt)
```

```
# Solve by the Heun formula
```

```
def diffeq_heun(diff1 func, t0, x0, dt):
```

```
    k0 = dt * diff1 func(t0, x0)
```

```
    k1 = dt * diff1 func(t0+dt, x0+k0)
```

```
    x1 = x0 + (k0 + k1) / 2.0
```

```
    return x1
```

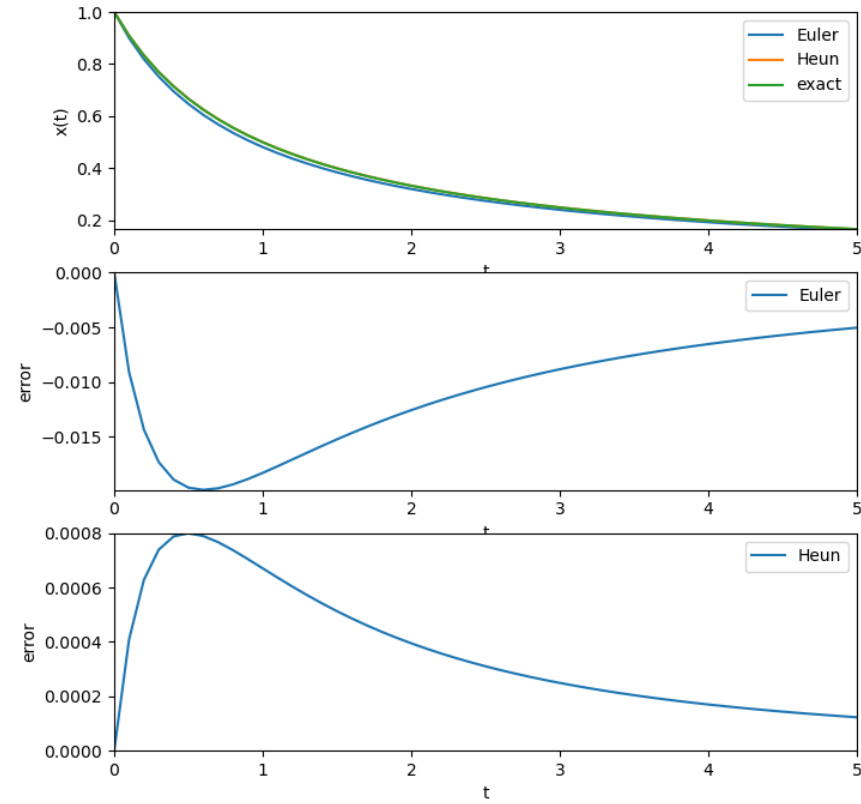
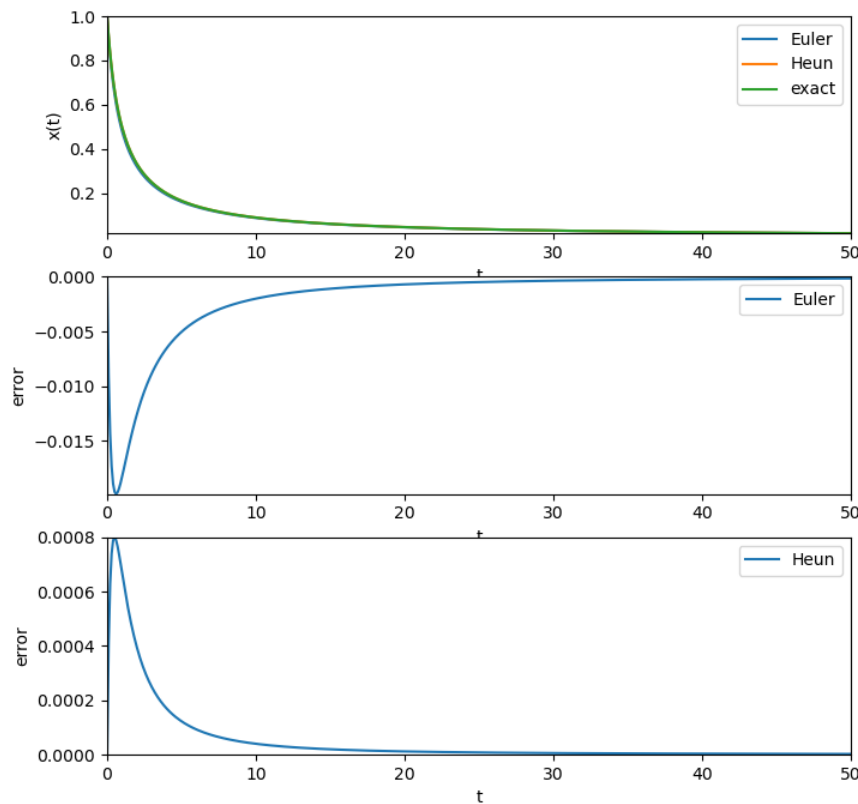
```
x1 = diffeq_heun(dxdt, t0, x0, dt)
```

Program: Euler vs. Heun methods

Usage: `python diffeq_euler_heun.py x0 dt nt iprint_interval`

`python diffeq_euler_heun.py`

$$\frac{dx}{dt} = -x^2 \text{ for } x_0 = 1.0, \Delta t = 0.1, n_t = 501$$



First-order diff. eq. : Simpson formula (シンプソン則)

$$\int_{x_0}^{x_2} g(x') dx' \sim \frac{1}{3} h [g(x_0) + 4g(x_1) + g(x_2)] = f(x_2) - f(x_0)$$

Solution of $\frac{df(x)}{dx} = g(x) \Rightarrow \frac{dx}{dt} = f(t, x)$

$$x(t + 2\Delta t) = x(t) + \frac{1}{3} \Delta t [f(t) + 4f(t + \Delta t) + f(t + 2\Delta t)]$$

Problem: $x(t + \Delta t)$ and $x(t + 2\Delta t)$ are unknown

\Rightarrow Use $x(t + \Delta t)$ by Euler or Heun formula

$$x(t + 2\Delta t) = x(t) + \frac{k_0 + 4k_1 + k_2}{3}$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t, x + k_0)$$

$$k_2 = \Delta t \cdot f(t + 2\Delta t, x + k_0 + k_1)$$

Convert Δt to a half

$$x(t + \Delta t) = x(t) + \frac{k_0 + 4k_1 + k_2}{6}$$

$$k_1 = \Delta t \cdot f(t + \Delta t / 2, x + k_0 / 2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t, x + (k_0 + k_1) / 2)$$

\Rightarrow Runge-Kutta formula

First-order diff. eq. : Runge-Kutta formula

(ルンゲークッタ公式)

$$\frac{dx}{dt} = f(t, x)$$

$$x(t + \Delta t) = x(t) + \frac{dx}{dt} \Delta t + \frac{1}{2!} \frac{d^2x}{dt^2} \Delta t^2 + \frac{1}{3!} \frac{d^3x}{dt^3} \Delta t^3 + \dots$$

$$x(t + \Delta t) = x(t) + \mu_1 k_1 + \mu_2 k_2 + \mu_3 k_3 + \dots$$

$$k_1 = \Delta t \cdot f(t, x)$$

$$k_2 = \Delta t \cdot f(t + \alpha_1 \Delta t, x + \beta_1 k_1)$$

$$k_3 = \Delta t \cdot f(t + \alpha_2 \Delta t, x + \beta_2 k_1 + \beta_3 k_2)$$

Determine μ_i and k_i so as to get minimum error

Number of k_i $n \Rightarrow n$ -stage formula

Formula of $O(\Delta t^p) = 0$ is called ‘order p formula’

3-stage 3-order Runge-Kutta formula

(3段3次のRunge-Kutta公式)

$$x(t + \Delta t) = x(t) + \frac{k_0 + 4k_1 + k_2}{6} + O(h^4)$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t / 2, x + k_0 / 2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t, x + 2k_1 - k_0)$$

Different from Simpson formula

$(k_0 + k_1)/2$

Different μ_i and k_i can provide the same accuracy

(同じ精度で違う取り方もできる)

$$k^* = \Delta t \cdot f(t + \Delta t / 4, x + \Delta x / 4)$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t / 2, x + k^* / 2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t, x + k_1)$$

4-stage 4-order Runge-Kutta formula

(4段4次のRunge-Kutta公式)

$$x(t + \Delta t) = x(t) + \frac{k_0 + 2k_1 + 2k_2 + k_3}{6}$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t/2, x + k_1/2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t/2, x + k_2/2)$$

$$k_3 = \Delta t \cdot f(t + \Delta t, x + k_3)$$

First-order differential equation

$$\frac{dx}{dt} = f(x, t)$$

Euler formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$

$$x(t + \Delta t) = x(t) + k_0$$

Heun formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$

$$k_1 = \Delta t \cdot f(x(t) + k_0, t + \Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{1}{2}(k_0 + k_1)$$

Simson formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$

$$k_1 = \Delta t \cdot f(x(t) + k_0/2, t + \Delta t/2)$$

$$k_2 = \Delta t \cdot f(x(t) + (k_0 + k_1)/2, t + \Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)$$

3-stage 3-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$

$$k_1 = \Delta t \cdot f(x(t) + k_0/2, t + \Delta t/2)$$

$$k_2 = \Delta t \cdot f(x(t) + 2k_1 - k_0, t + \Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)$$

4-stage 4-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$

$$k_1 = \Delta t \cdot f(x(t) + k_1/2, t + \Delta t/2)$$

$$k_2 = \Delta t \cdot f(x(t) + k_2/2, t + \Delta t/2)$$

$$k_3 = \Delta t \cdot f(x(t) + k_3, t + \Delta t)$$

$$x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3)$$

Second-order diff. eq. (二階微分方程式)

$$\frac{d^2 \mathbf{r}_i}{dt^2} = \mathbf{F}_i / m_i$$

- 2nd-order diff eq is divided to two simultaneous 1st-order eqs

(二階微分方程式の場合、一階微分方程式に分解するのが良い)

$$\frac{d^2 x}{dt^2} = f(x, v, t)$$

$$\frac{dv}{dt} = f(x, v, t) \quad \frac{dx}{dt} = v$$

Euler formula: $v(t + \Delta t) \sim v(t) + \Delta t \cdot \frac{dv}{dt}$

$$v(t + \Delta t) = v(t) + \Delta t \cdot f(x(t), v(t), t)$$

$$x(t + \Delta t) = x(t) + \Delta t \cdot v(t)$$

Second-order diff. eq. : Heun formula

(二階微分方程式の解法: ホイン法)

$$\frac{d^2x}{dt^2} = f(x, v, t)$$

$$\frac{dv}{dt} = f(x, v, t)$$

$$(1) k_0 = \Delta t \cdot f(x(t), v(t), t)$$

$$(3) k_1 = \Delta t \cdot f(\mathbf{x}(t) + \mathbf{k}_0', \mathbf{v}(t) + \mathbf{k}_0, t + \Delta t)$$

$$(4) v(t + \Delta t) = v(t) + \frac{1}{2}(k_0 + k_1)$$

Each step needs to calculate k_0 and k_1 :
time-consuming for MD

$$\frac{dx}{dt} = v(x, v, t)$$

$$(2) k_0' = \Delta t \cdot v(t)$$

$$(5) k_1' = \Delta t \cdot v(t + \Delta t)$$

$$(6) x(t + \Delta t) = x(t) + \frac{1}{2}(k_0' + k_1')$$

Second-order diff. eq. : Verlet formula

(二階微分方程式の解法: ベルレ法)

$$\frac{d^2x}{dt^2} = f(x, v, t)$$

$$f(x, v, t) = \frac{d^2x(t)}{dt^2} \sim \frac{x(t + \Delta t) - 2x(t) + x(t - \Delta t)}{\Delta t^2}$$

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + \Delta t^2 f(\mathbf{x}(t), \mathbf{v}(t), t)$$

$$v(t) = \frac{1}{2\Delta t} \{x(t + \Delta t) - x(t - \Delta t)\}$$

Each step only needs to calculate only one $f(\mathbf{x}(t), \mathbf{v}(t), t)$

- Better accuracy than Euler formula, equivalent to Heun formula
- Directly solve 2nd-order differential equation
- Drawback:
The subtraction of similar values, $x(t+n\Delta t)$, may cause roundoff error.

velocity Verlet formula

$$\frac{d^2x}{dt^2} = f(t, x, v)$$

$$\frac{d^2x(t + \Delta t)}{dt^2} \sim \frac{x(t + 2\Delta t) - 2x(t + \Delta t) + x(t)}{\Delta t^2}$$

$$x(t + 2\Delta t) = 2x(t + \Delta t) - x(t) + \Delta t^2 f(t + \Delta t, x(t + \Delta t), v(t + \Delta t))$$

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + \Delta t^2 f(t, x(t), v(t))$$

$$v(t + \Delta t) = v(t) + \frac{1}{2} \{ f(t + \Delta t, x(t + \Delta t), v(t + \Delta t)) + f(t, x(t), v(t)) \}$$

- **Better accuracy than Verlet formula**

Program: diffeq2nd_verlet.py

Usage: python diffeq2nd_verlet.py

| t | x(cal) | x(exact) | v(cal) |
|---------|-----------|-----------|-----------|
| t= 0.00 | 0.000000 | 0.000000 | 1.000000 |
| t= 0.01 | 0.010000 | 0.010000 | 0.999950 |
| t= 0.20 | 0.198673 | 0.198669 | 0.980066 |
| t= 0.40 | 0.389425 | 0.389418 | 0.921060 |
| t= 0.60 | 0.564652 | 0.564642 | 0.825334 |
| t= 0.80 | 0.717367 | 0.717356 | 0.696704 |
| t= 1.00 | 0.841484 | 0.841471 | 0.540299 |
| t= 1.20 | 0.932053 | 0.932039 | 0.362353 |
| t= 1.40 | 0.985463 | 0.985450 | 0.169961 |
| t= 1.60 | 0.999586 | 0.999574 | -0.029206 |
| t= 1.80 | 0.973858 | 0.973848 | -0.227209 |
| t= 2.00 | 0.909305 | 0.909297 | -0.416154 |
| t= 2.20 | 0.808501 | 0.808496 | -0.588509 |
| t= 2.40 | 0.675464 | 0.675463 | -0.737400 |
| t= 2.60 | 0.515499 | 0.515501 | -0.856894 |
| t= 2.80 | 0.334981 | 0.334988 | -0.942226 |
| t= 3.00 | 0.141109 | 0.141120 | -0.989994 |
| t= 3.20 | -0.058388 | -0.058374 | -0.998294 |
| t= 3.40 | -0.255558 | -0.255541 | -0.966795 |
| t= 3.60 | -0.442539 | -0.442520 | -0.896752 |
| t= 3.80 | -0.611878 | -0.611858 | -0.790958 |
| t= 4.00 | -0.756823 | -0.756802 | -0.653631 |
| t= 4.20 | -0.871595 | -0.871576 | -0.490246 |
| t= 4.40 | -0.951620 | -0.951602 | -0.307315 |
| t= 4.60 | -0.993706 | -0.993691 | -0.112133 |

Second-order diff. eq. : Leap Flog formula

(二階微分方程式の解法: かえる跳び法)

Essentially the same as the Verlet formula.

However, Verlet formula

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + \Delta t^2 f(t, x(t), v(t))$$

includes the subtraction of

$x(t)$ terms and may cause roundoff error.

Converting the equation to

$$v(t + \Delta t) = v(t - \Delta t) + 2\Delta t \cdot f(t, x(t), v(t))$$

$$x(t + 2\Delta t) = x(t) + 2\Delta t \cdot v(t + \Delta t)$$

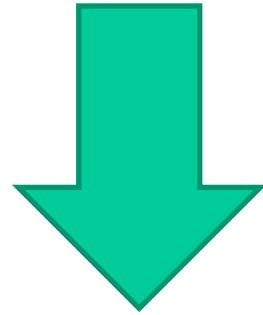
Can reduce the roundoff errors.

Note: Time t calculated for $x(t)$ and $v(t)$ offset by Δt relative to each other.

Leap Flog vs. Verlet

Confirm the Leap Flog formula is identical to the Verlet formula

Leap Flog $x(t + 2\Delta t) = x(t) + 2\Delta t \cdot v(t + \Delta t)$



$$v(t - \Delta t) = \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = \frac{x(t) - x(t - \Delta t)}{\Delta t} + 2\Delta t \cdot f(t, x(t), v(t))$$

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + 2\Delta t \cdot f(t, x(t), v(t))$$

Verlet

Program: diffeq2nd_2d_euler.py

Usage: python diffeq2nd_2d_euler.py

| t | x(cal) | x(exact) | y(cal) | y(exact) |
|---------|-----------|-----------|-----------|-----------|
| t= 0.00 | 0.000000 | 0.000000 | 2.000000 | 2.000000 |
| t= 0.01 | 0.010000 | 0.010000 | 2.000000 | 1.999900 |
| t= 0.20 | 0.198862 | 0.198669 | 1.962097 | 1.960133 |
| t= 0.40 | 0.390186 | 0.389418 | 1.845820 | 1.842122 |
| t= 0.60 | 0.566322 | 0.564642 | 1.655653 | 1.650671 |
| t= 0.80 | 0.720212 | 0.717356 | 1.399036 | 1.393413 |
| t= 1.00 | 0.845671 | 0.841471 | 1.086077 | 1.080605 |
| t= 1.20 | 0.937633 | 0.932039 | 0.729152 | 0.724716 |
| t= 1.40 | 0.992364 | 0.985450 | 0.342415 | 0.339934 |
| t= 1.60 | 1.007603 | 0.999574 | -0.058761 | -0.058399 |
| t= 1.80 | 0.982665 | 0.973848 | -0.458394 | -0.454404 |
| t= 2.00 | 0.918464 | 0.909297 | -0.840535 | -0.832294 |
| t= 2.20 | 0.817482 | 0.808496 | -1.189900 | -1.177002 |
| t= 2.40 | 0.683677 | 0.675463 | -1.492481 | -1.474787 |
| t= 2.60 | 0.522322 | 0.515501 | -1.736110 | -1.713778 |
| t= 2.80 | 0.339800 | 0.334988 | -1.910948 | -1.884445 |
| t= 3.00 | 0.143353 | 0.141120 | -2.009878 | -1.979985 |
| t= 3.20 | -0.059207 | -0.058374 | -2.028803 | -1.996590 |
| t= 3.40 | -0.259811 | -0.255541 | -1.966806 | -1.933596 |
| t= 3.60 | -0.450448 | -0.442520 | -1.826199 | -1.793517 |
| t= 3.80 | -0.623492 | -0.611858 | -1.612436 | -1.581935 |
| t= 4.00 | -0.772001 | -0.756802 | -1.333901 | -1.307287 |
| t= 4.20 | -0.890001 | -0.871576 | -1.001578 | -0.980522 |
| t= 4.40 | -0.972722 | -0.951602 | -0.628623 | -0.614666 |
| t= 4.60 | -1.016792 | -0.993691 | -0.229835 | -0.224305 |

Program: diffeq2nd_2d_verlet.py

Usage: python diffeq2nd_2d_verlet.py

| t | x(cal) | x(exact) | y(cal) | y(exact) |
|---------|-----------|-----------|-----------|-----------|
| t= 0.00 | 0.000000 | 0.000000 | 2.000000 | 2.000000 |
| t= 0.01 | 0.010050 | 0.010000 | 1.999950 | 1.999900 |
| t= 0.20 | 0.199666 | 0.198669 | 1.961126 | 1.960133 |
| t= 0.40 | 0.391372 | 0.389418 | 1.844068 | 1.842122 |
| t= 0.60 | 0.567475 | 0.564642 | 1.653492 | 1.650671 |
| t= 0.80 | 0.720954 | 0.717356 | 1.396995 | 1.393413 |
| t= 1.00 | 0.845691 | 0.841471 | 1.084805 | 1.080605 |
| t= 1.20 | 0.936713 | 0.932039 | 0.729366 | 0.724716 |
| t= 1.40 | 0.990390 | 0.985450 | 0.344850 | 0.339934 |
| t= 1.60 | 1.004584 | 0.999574 | -0.053414 | -0.058399 |
| t= 1.80 | 0.978727 | 0.973848 | -0.449550 | -0.454404 |
| t= 2.00 | 0.913852 | 0.909297 | -0.827762 | -0.832294 |
| t= 2.20 | 0.812544 | 0.808496 | -1.172975 | -1.177002 |
| t= 2.40 | 0.678842 | 0.675463 | -1.471424 | -1.474787 |
| t= 2.60 | 0.518076 | 0.515501 | -1.711211 | -1.713778 |
| t= 2.80 | 0.336656 | 0.334988 | -1.882778 | -1.884445 |
| t= 3.00 | 0.141815 | 0.141120 | -1.979283 | -1.979985 |
| t= 3.20 | -0.058680 | -0.058374 | -1.996880 | -1.996590 |
| t= 3.40 | -0.256836 | -0.255541 | -1.934867 | -1.933596 |
| t= 3.60 | -0.444752 | -0.442520 | -1.795716 | -1.793517 |
| t= 3.80 | -0.614937 | -0.611858 | -1.584975 | -1.581935 |
| t= 4.00 | -0.760607 | -0.756802 | -1.311046 | -1.307287 |
| t= 4.20 | -0.875953 | -0.871576 | -0.984849 | -0.980522 |
| t= 4.40 | -0.956378 | -0.951602 | -0.619389 | -0.614666 |
| t= 4.60 | -0.998674 | -0.993691 | -0.229235 | -0.224305 |

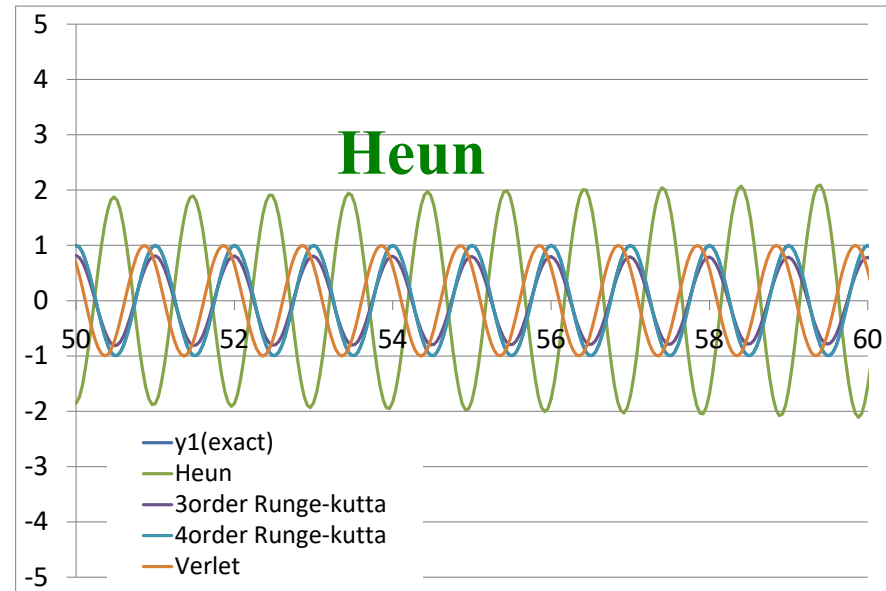
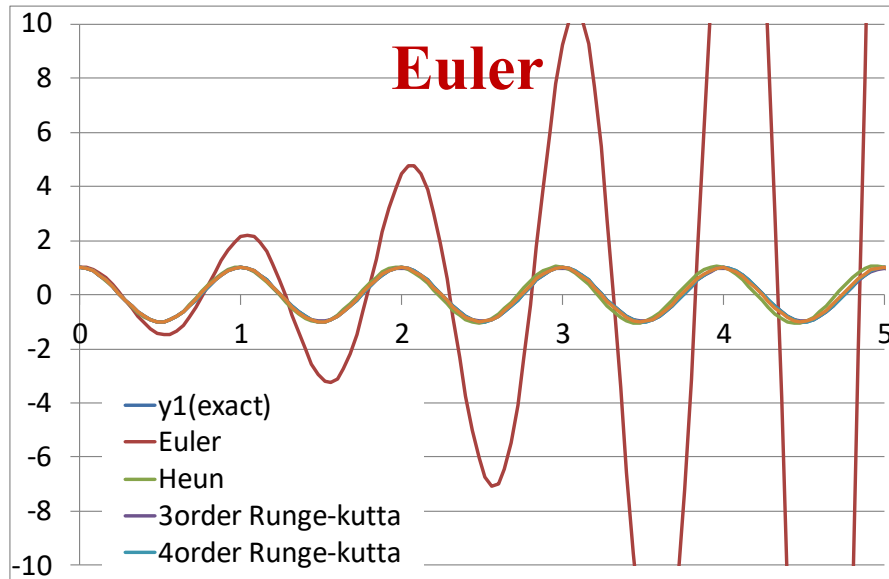
Accuracy of numerical solutions: Diff. eq.

$$\frac{d^2x}{dt^2} = -4\pi^2 x \quad \left(\frac{dx}{dt} = v, \quad \frac{dv}{dt} = -4\pi^2 x \right)$$

Exact ($t = 0$: $x = 1.0$, $v = 0.0$)

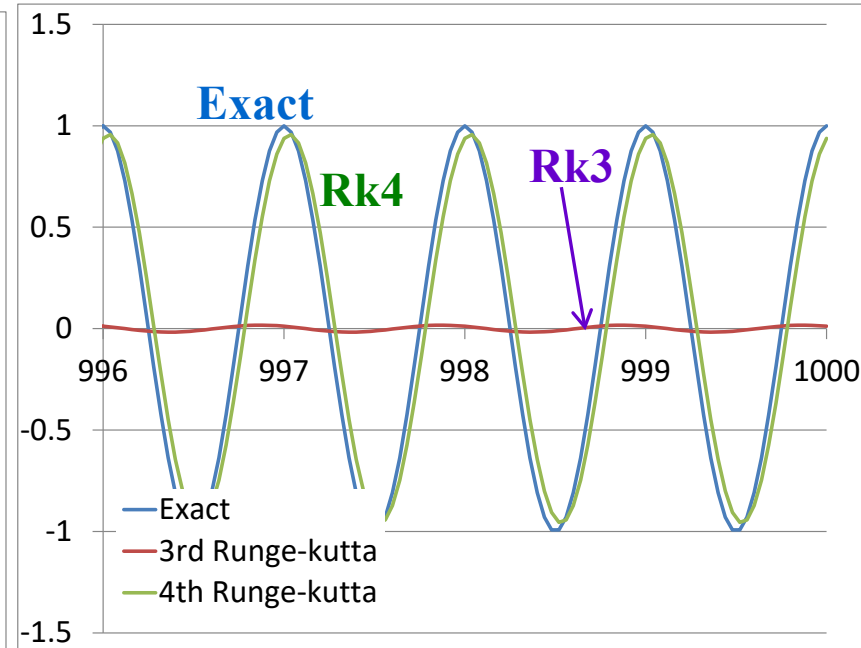
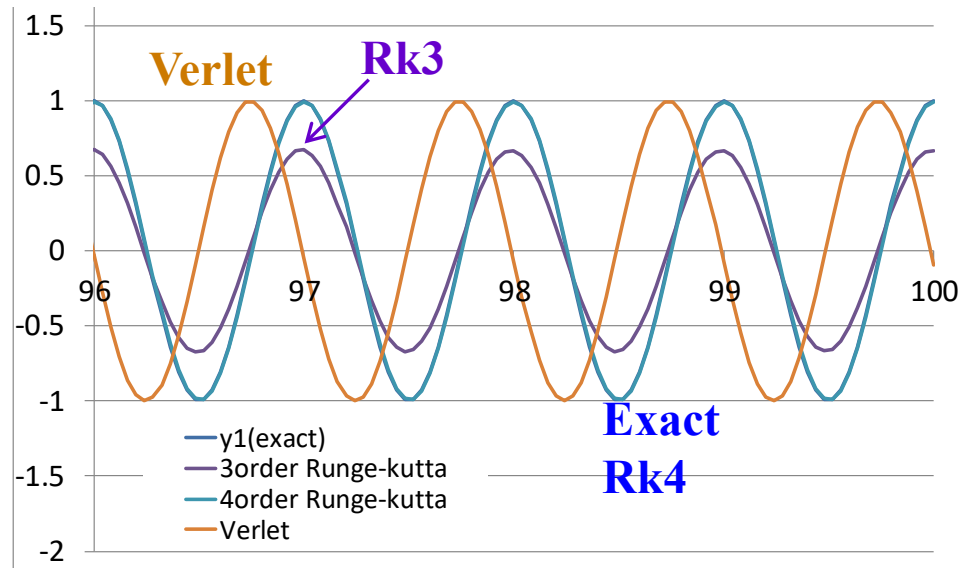
$$x = \cos(2\pi t) \quad v = -2\pi \sin(2\pi t)$$

$\Delta t = 0.04$

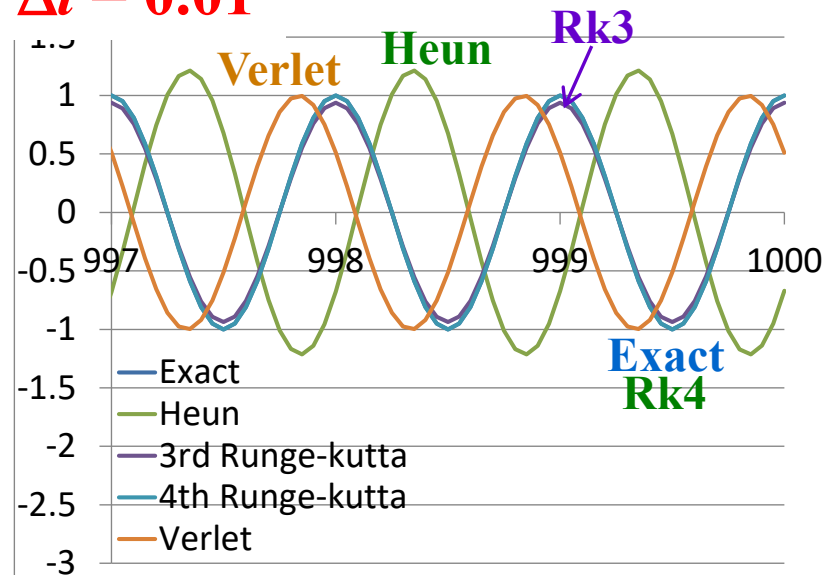


Accuracy of numerical solutions: Diff. eq.

$\Delta t = 0.04$

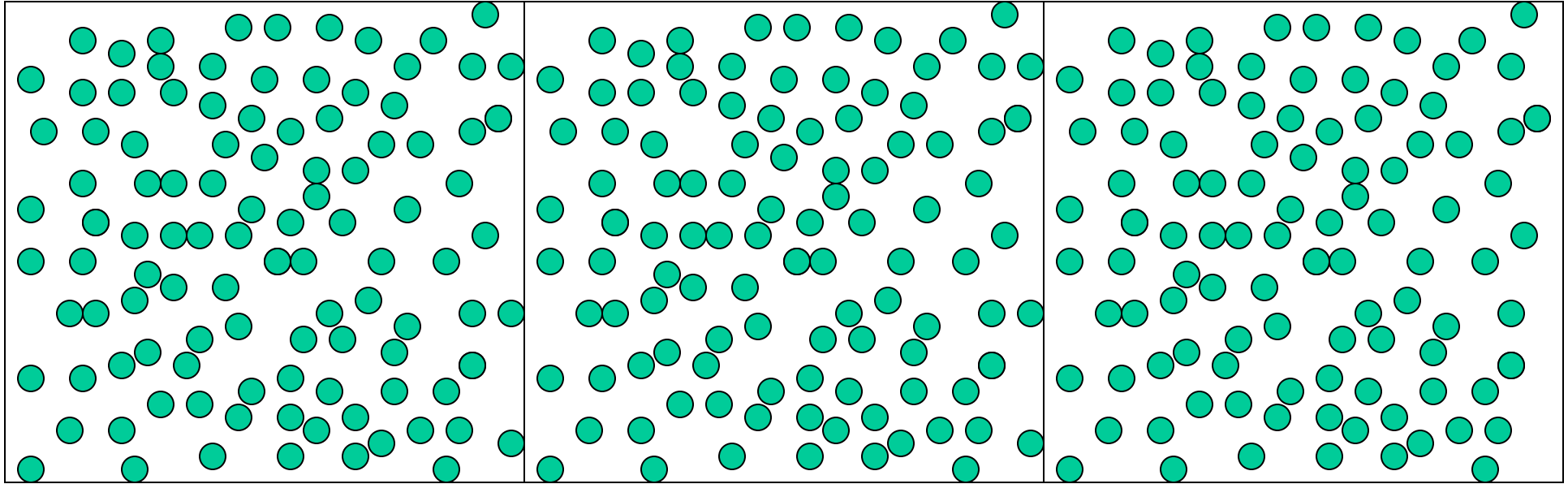


$\Delta t = 0.01$



Molecular dynamics (MD) (分子動力学法)

3D periodic condition: MD cell



$$\mathbf{F}_i = m_i \frac{d^2 \mathbf{r}_i}{dt^2}$$

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i}{m_i}$$

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \cdot \mathbf{v}_i(t)$$

Empirical interatomic potential

(経験的原子間ポテンシャル)

Hard core potential

ハードコア(剛体)ポテンシャル

$$\begin{aligned}\phi(r) &= \infty & r \leq \sigma \\ &= 0 & r > \sigma\end{aligned}$$

Lennard-Jones (LJ) potential

レナード-ジョーンズポテンシャル

$$\phi_{ij}(r) = 4\varepsilon_{ij} \left\{ \left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^6 \right\}$$

Born-Mayer-Huggins (BMH) potential

ボルン-メイヤー-ヒュギンズ

$$\phi_{ij}(r) = \frac{z_i z_j e^2}{r} + A_{ij} b \cdot \exp\left(\frac{\sigma_i + \sigma_j - r}{\rho}\right) - \frac{C_{ij}}{r^6} - \frac{D_{ij}}{r^8}$$

Kawamura potential (MXDOorto/MXDTricl)

河村ポテンシャル

$$\phi_{ij} = \frac{z_i z_j}{r_{ij}} + f_0(b_i + b_j) \exp\left(\frac{a_i + a_j - r_{ij}}{b_i + b_j}\right) + \frac{c_i c_j}{r_{ij}^6}$$

$$\begin{aligned}\phi_{ij}(r) &= \frac{z_i z_j e^2}{r} + f_0(b_i + b_j) \exp\left(\frac{a_i + a_j - r}{b_i + b_j}\right) \\ &+ D_{ij} \left(\exp[-2\beta_{ij}(r - r^*)] - 2 \exp[-\beta_{ij}(r - r^*)] \right)\end{aligned}$$

Morse potential

Empirical interatomic potential

$$U_{ij}(r_{ij}) = \frac{z_i z_j e^2}{4\pi\epsilon_0} \frac{1}{r_{ij}} + f_0(b_i + b_j) \exp\left[\frac{a_i + a_j - r_{ij}}{b_i + b_j}\right] + \frac{c_i c_j}{r_{ij}^6}$$

Coulomb potential

Repulsion term

**Dispersion
(London interaction)**

Example of Parameters for an ion

| | | |
|---------------|---------|-----------------------------|
| Ion charge | : z_i | Fixed to ion formal charge |
| ~Ion radius | : a_i | Adjust to crystal structure |
| ~Ion hardness | : b_i | Adjust to elastic constant |
| Dispersion | : c_i | Fixed |

Potentials and forces for the ion i at r_i

$$U_i(\mathbf{r}_i, t) = \sum_j U_{ij}(\mathbf{r}_j(t) - \mathbf{r}_i(t)), \quad \mathbf{F}_i(\mathbf{r}_i, t) = - \sum_j \frac{\partial}{\partial \mathbf{r}_i} U_{ij}(\mathbf{r}_j(t) - \mathbf{r}_i(t))$$

Most time-consuming term

Better to re-use previous steps,

$\mathbf{F}_i(\mathbf{r}_i, t - \Delta t), \mathbf{F}_i(\mathbf{r}_i, t - 2\Delta t)$ etc

=> Verlet formula is better than Heun and Runge-Kutta formula

Requirements of algorithms used for MD

Requirements

- Enough accuracy (can be checked by energy / momentum conservation laws)
- Fast calculations (note the most time-consuming process is the force calculations, **better to re-use the previous results**)

Runge-Kutta formula: not suitable for MD

High accuracy, but high cost

It **cannot re-use** the previous results

Each step requires three/four new force calculations, high cost

Frequently used formula:

- Verlet formula (Leap Flog formula)
- Beeman formula
- Predictor-Corrector method (予測子－修正子法)

Rahman predictor-corrector method

(ラーマンの予測子－修正子法)

Gear predictor-corrector method (ギアの予測子－修正子法)

Program: Planet simulation

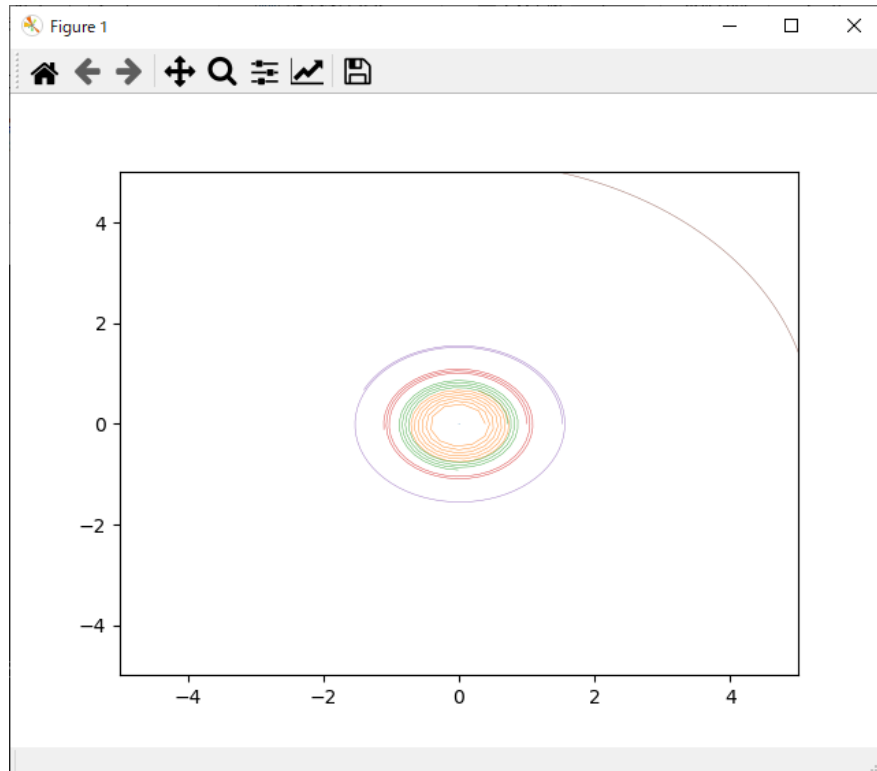
Usage: `python diffeq2nd_planet.py solver dt nt`

solver: 'Euler' or 'Verlet'

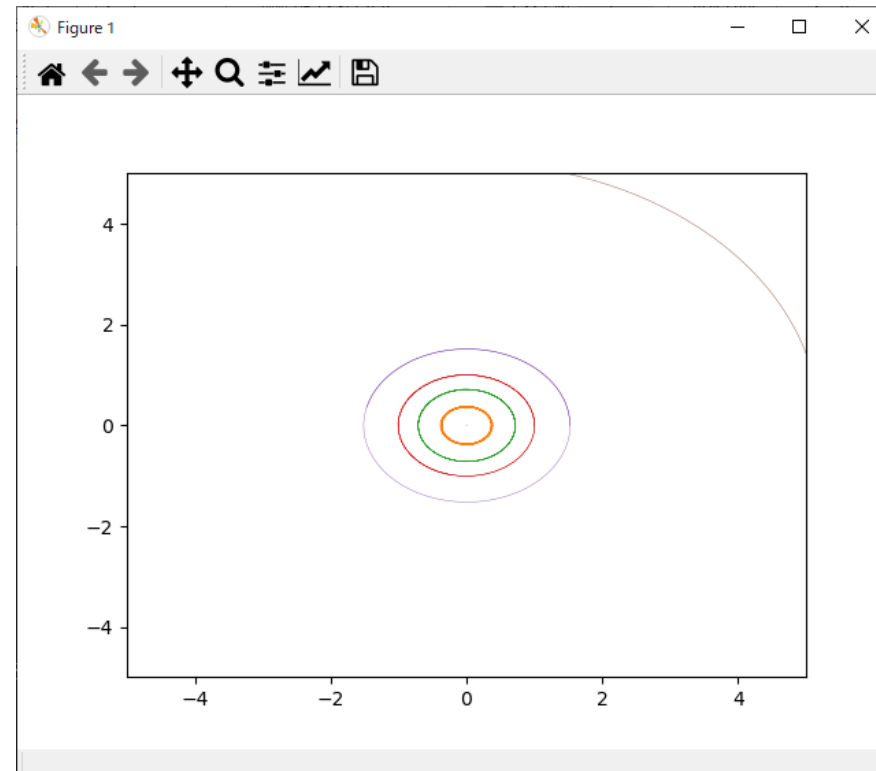
dt: time step in day (time is normalized by a day)

nt: number of steps

`python diffeq2nd_planet.py Euler 0.2 5000`



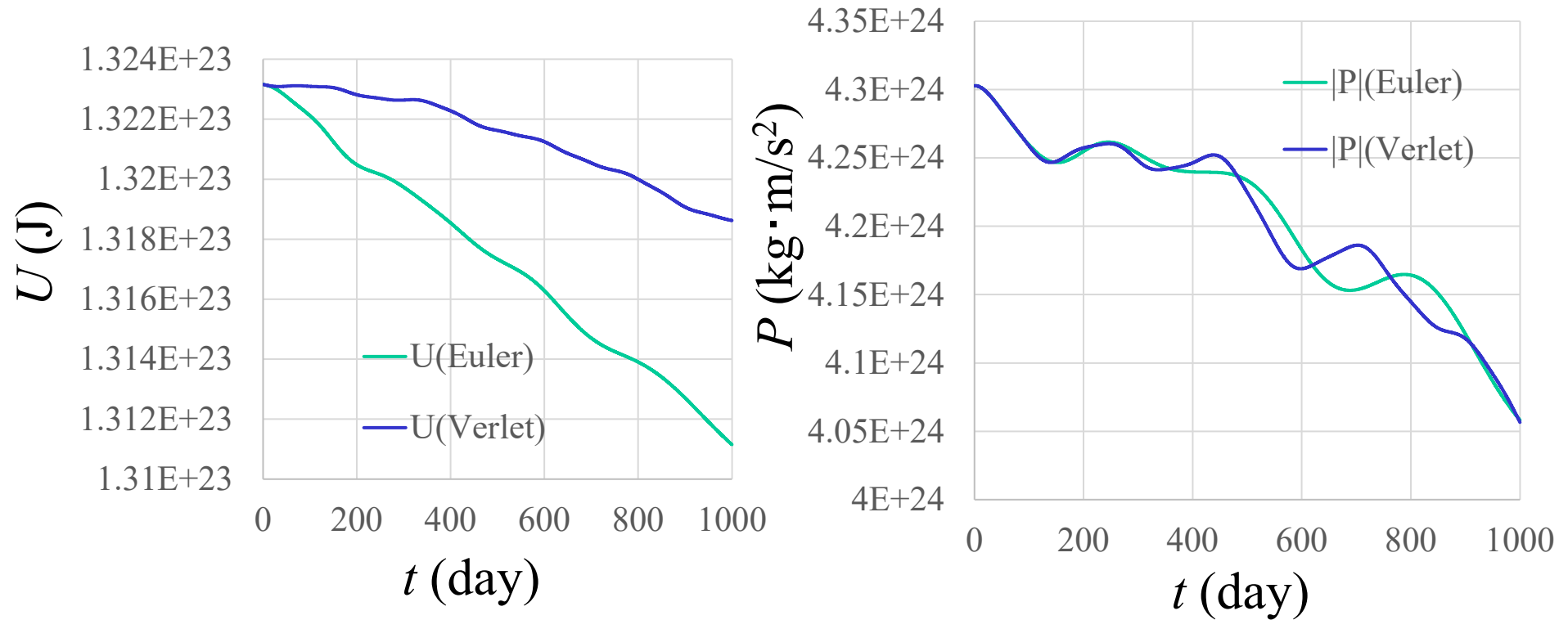
`python diffeq2nd_planet.py Verlet 0.2 5000`



Program: Check by conservation law

python diffeq2nd_planet.py Euler 0.2 5000

python diffeq2nd_planet.py Verlet 0.2 5000



Approximation of discrete data:

Interpolation/Extrapolation

(離散データの近似: 補間/補外)

Interpolation

Pattern 1: Reproduce all sample points (標本点を必ず通る)

n sample points are reproduced by $(n - 1)$ order polynomial.

- Interpolated data might be scattered largely in particular for orders higher than 3 (Runge's phenomenon/oscillation **ルンゲの現象**).
補間点が大きく振動する問題がでる。特に3次以上の多項式

=> To suppress the Runge's phenomenon:

Make the n -th order differentiations continuous at the boundaries between neighboring regions

=> Spline function

n sample points are reproduced by $(n + N - 1)$ order polynomial.

Pattern 2: Smoothing (平滑化)

Scattering of data will be reduced

Pattern 3: Does not reproduce sample points exactly, but the deviation will be minimized

(標本点を通らないが、補間データは標本点から大きく外れない)

- Least-squares method (LSQ, 最小二乗法)
- Minimax approximation (ミニマックス近似)

Polynomial that reproduces sample points

(標本点を通る多項式)

n sample points (x_i, y_i) ($i = 1, \dots, n$) are reproduced by $(n - 1)$ order polynomial.

$$y_i = \sum_{k=0}^{n-1} a_k x_i^k \quad (i = 1, \dots, n)$$
$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & & x_2^{n-1} \\ 1 & x_3 & x_3^2 & & x_3^{n-1} \\ \vdots & & & \ddots & \\ 1 & x_n & x_n^2 & & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$|x_i| > 1$ might cause overflow,

$|x_i| < 1$ might cause underflow errors.

=> Normalize (正規化) the x range e.g. to $[-1, 1]$: $x'_i = 2 \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$

Standardize (標準化) by average and standard deviation: $x'_i = 2 \frac{x_i - x_{\text{average}}}{\sigma_x}$

Lagrange interpolation formula

(ラグランジの補間公式)

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)

$(n - 1)$ order polynomial that reproduces n sample points

(x_i, y_i) ($i = 0, \dots, n - 1$) is determined uniquely.

Lagrange interpolation formula

$$P_{n-1}(x) = f(x_0)\phi_0(x) + f(x_1)\phi_1(x) + \dots f(x_{n-1})\phi_{n-1}(x)$$

$$\phi_i(x) = \frac{\prod_{k \neq i}^{n-1} (x - x_k)}{\prod_{k \neq i}^{n-1} (x_i - x_k)} = \prod_{k \neq i}^{n-1} \frac{(x - x_k)}{(x_i - x_k)}$$

$n = 2$:

$$P_1(x) = f(x_0) \frac{(x - x_1)}{(x_0 - x_1)} + f(x_1) \frac{(x - x_0)}{(x_1 - x_0)}$$

$n = 3$:

$$P_2(x) = f(x_0) \frac{(x - x_1)}{(x_0 - x_1)} \frac{(x - x_2)}{(x_0 - x_2)} + f(x_1) \frac{(x - x_0)}{(x_1 - x_0)} \frac{(x - x_2)}{(x_1 - x_2)} + f(x_2) \frac{(x - x_0)}{(x_2 - x_0)} \frac{(x - x_1)}{(x_2 - x_1)}$$

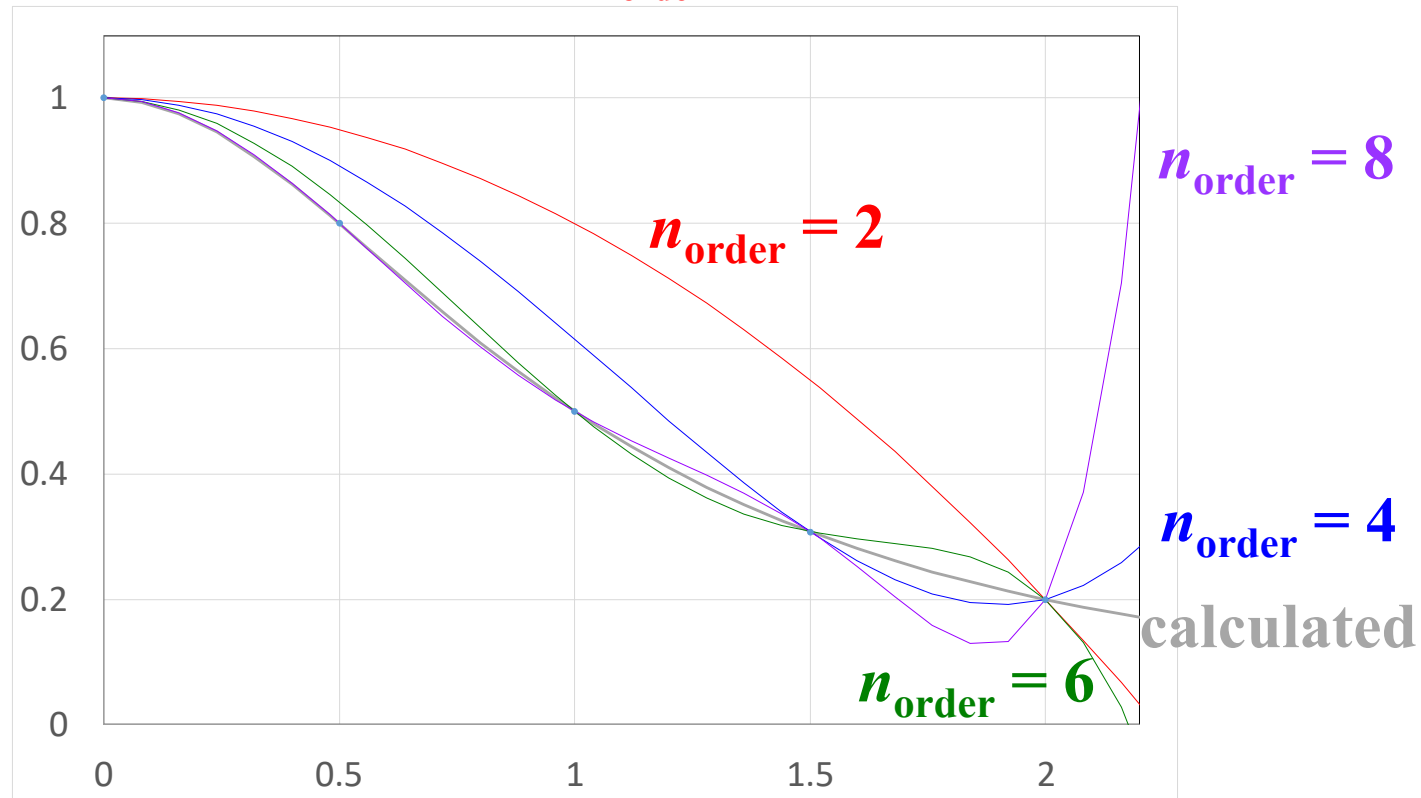
Problem of such polynomials

- Increasing the sample points will change the coefficients of polynomial completely.

- Runge's phenomenon / oscillation** (ルンゲの現象)

High order (e.g. >3) polynomial will cause large oscillations at points other than the sample points (高次の多項式では標本点以外で大きく振動することがある)

Ex. Interpolate $f(x) = 1 / (1 + x^2)$ for $(n_{\text{order}}+1)$ points in the range $x = [-2, 2]$



In the machine learning (機械学習):

Overfitting (過適合), Overlearning (過学習)

Interpolation: Piecewise polynomial interpolation

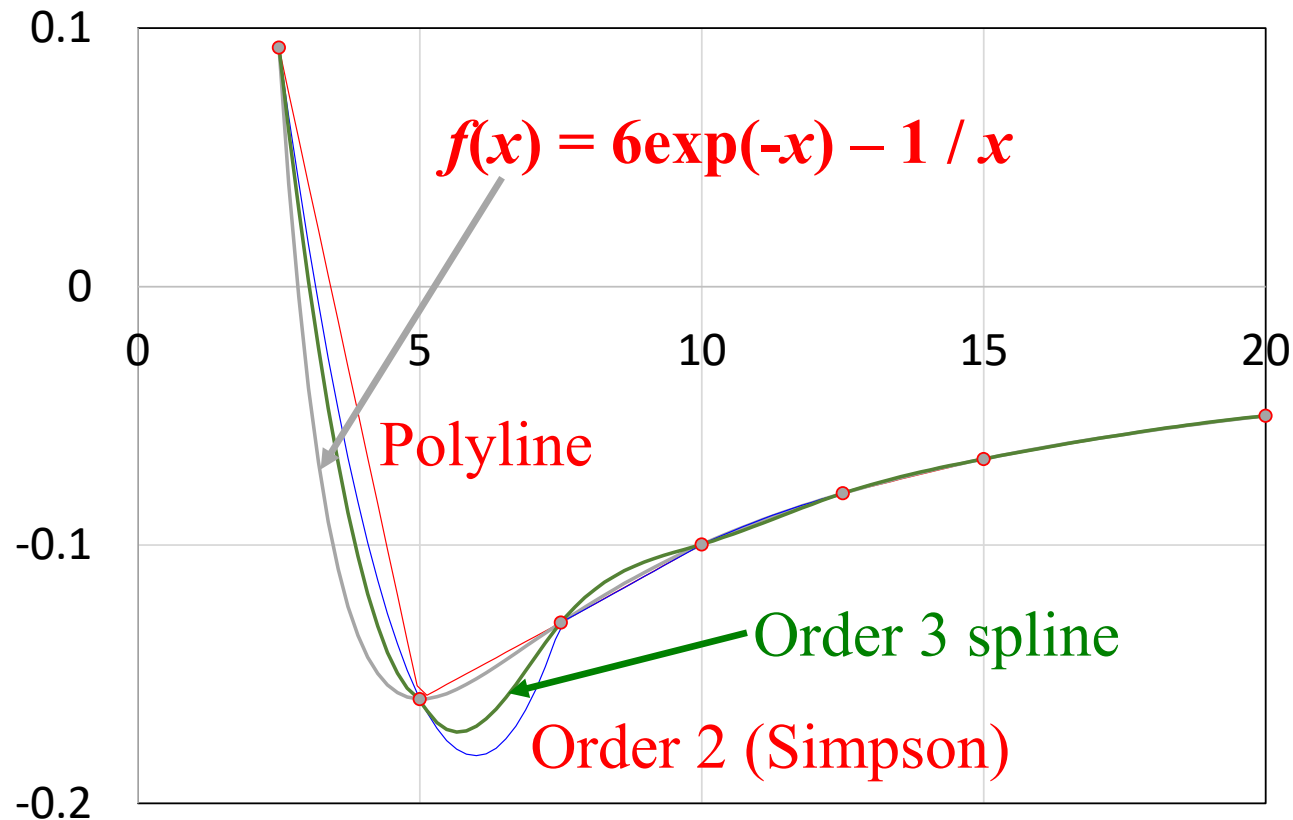
(区分多項式補間)

Connect divided sections by polylines (折れ線)

=> First derivatives will be discontinuous at the boundaries

=> $(n - 1)$ -th derivatives are continuous for whole range:

Order n spline functions (n 次のスプライン関数)

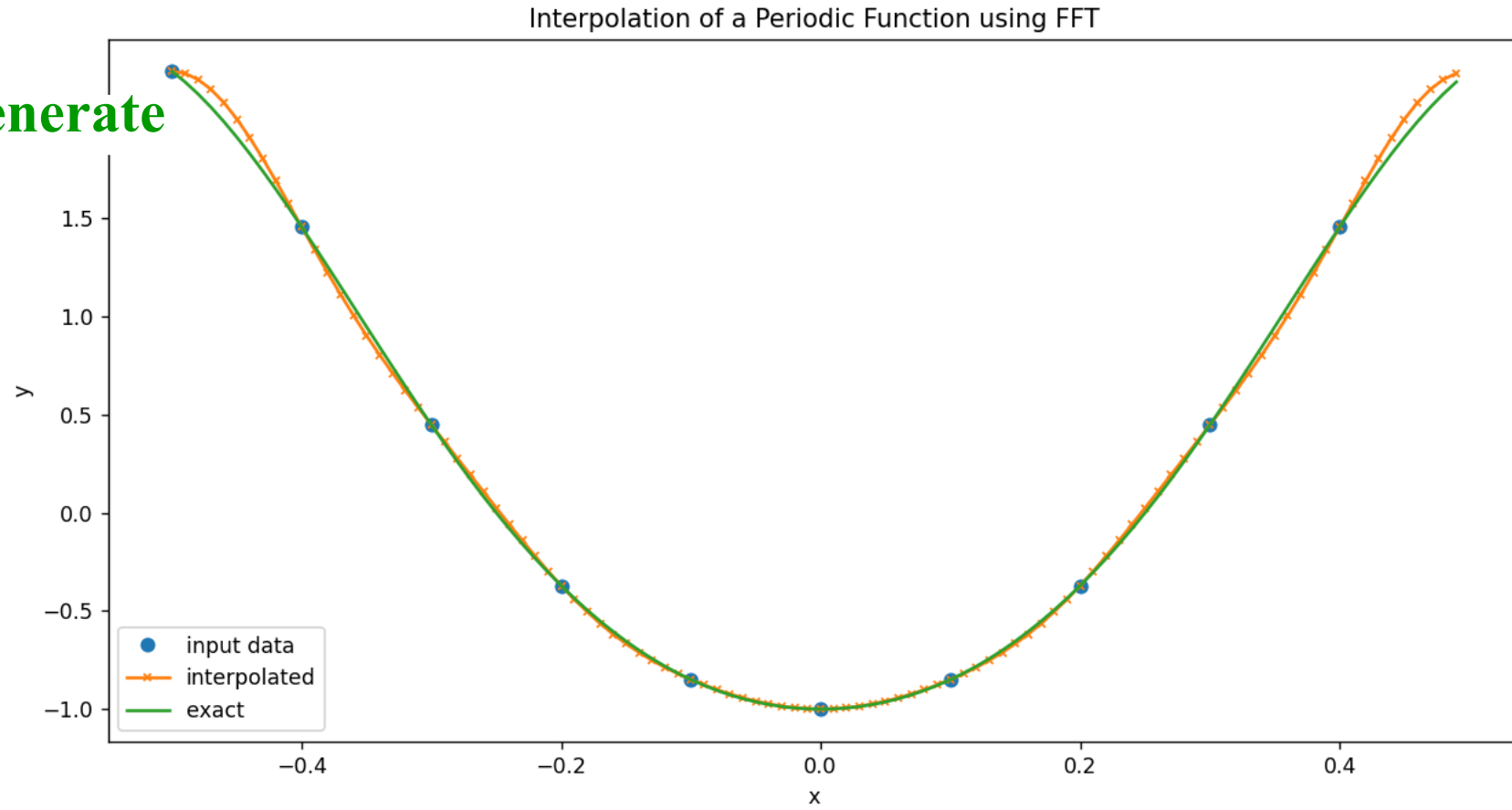


Interpolation by FFT (Fast Fourier Transform)

For a periodic function $E(k)$ (like an energy function $E(k)$ of crystal),
Interpolation can be carried out by:

1. Fourier transform $E(k)$ to $E^*(x)$
2. Increase the number of data by adding zeros in the high $|x|$ region (padding)
3. Inverse Fourier transform

> **python interpolate_fft.py generate**



Smoothing

平滑化

Smoothing (平滑化)

Take some average for sample points

- **Moving average (移動平均)**
 - **Simple moving average (単純移動平均):**
Average of sequential data with the uniform weight
 - **Weighted moving average (加重移動平均):**
Average of sequential data with weight
Weight: Linear, Triangular, Exponential, Gauss, etc...

Approximate sample points by some function

- **Polynomial smoothing (多項式による平滑化)**
- **Smoothing spline (スプライン平滑化)**
- **Least-squares method (最小二乗法)**

Other

- **Fourier transformation (フーリエ変換)**

Calculation

Simple moving average (2m+1 points)

$$y_{i,smoothed} = \frac{1}{2m+1} \sum_{j=i-m}^{i+m} y_j$$

Weighted moving average (2m+1 points)

$$y_{i,smoothed} = \sum_{j=i-m}^{i+m} w_j y_j / \sum_{j=i-m}^{i+m} w_j$$

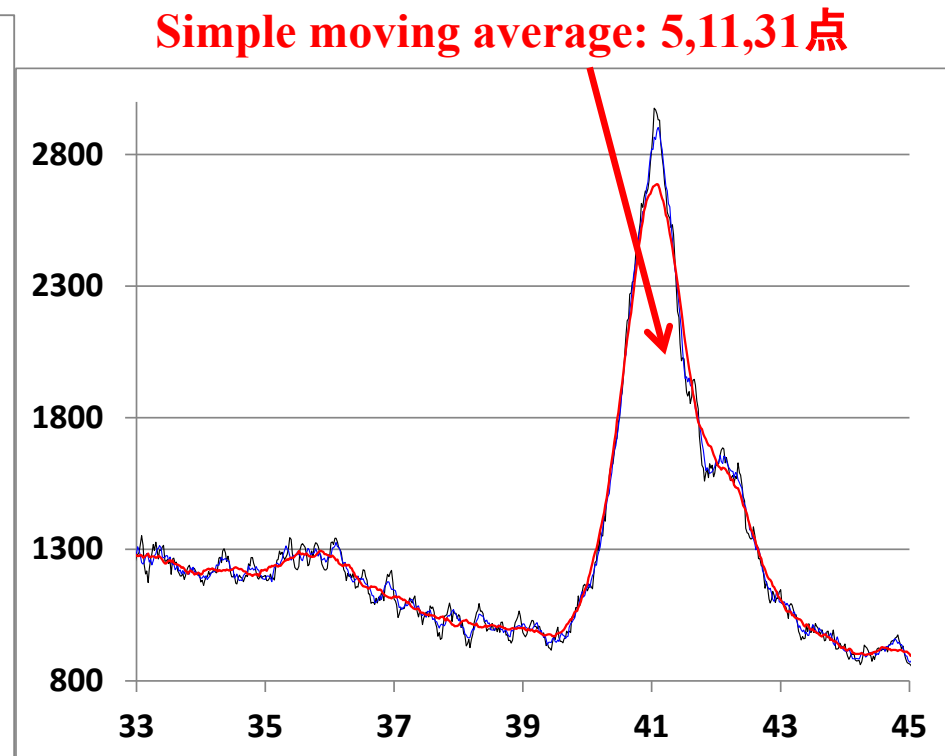
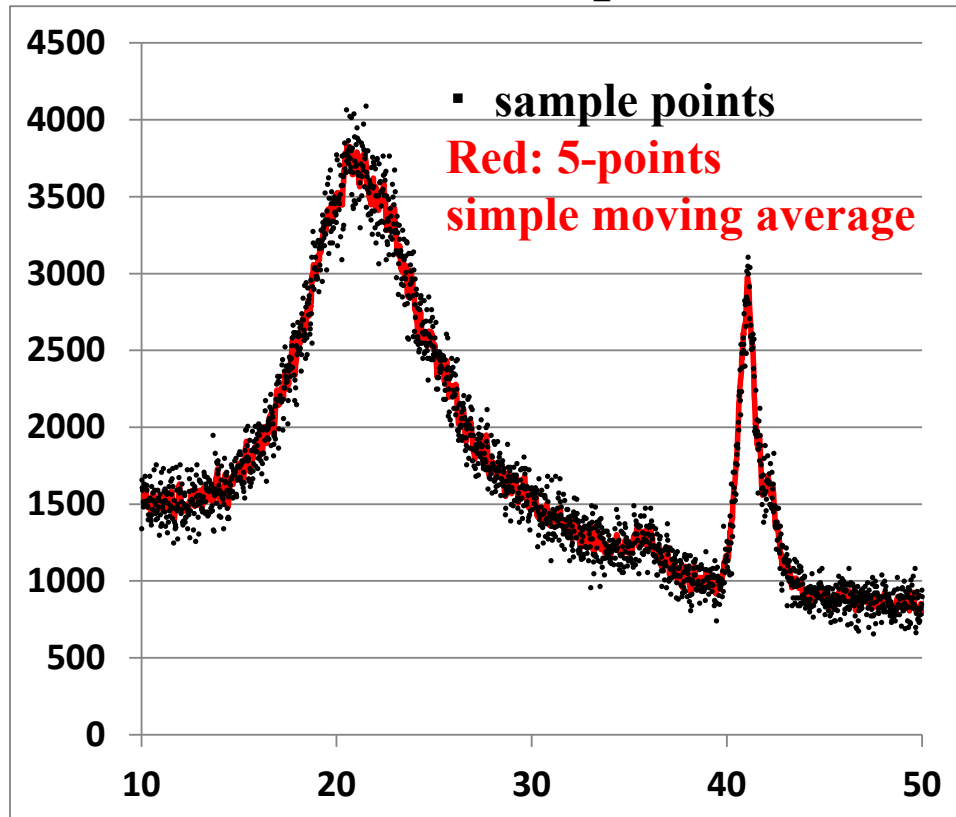
Smoothing (平滑化)

- **Moving average (移動平均法)**

Smoother with more sample points for average,
but would alter the function shape if the function is not monotonic.
=> Affect peak height, valley depth, peak width etc...

The range of averaged sample points larger than the peak width
=> split peaks might become difficult to be separated.

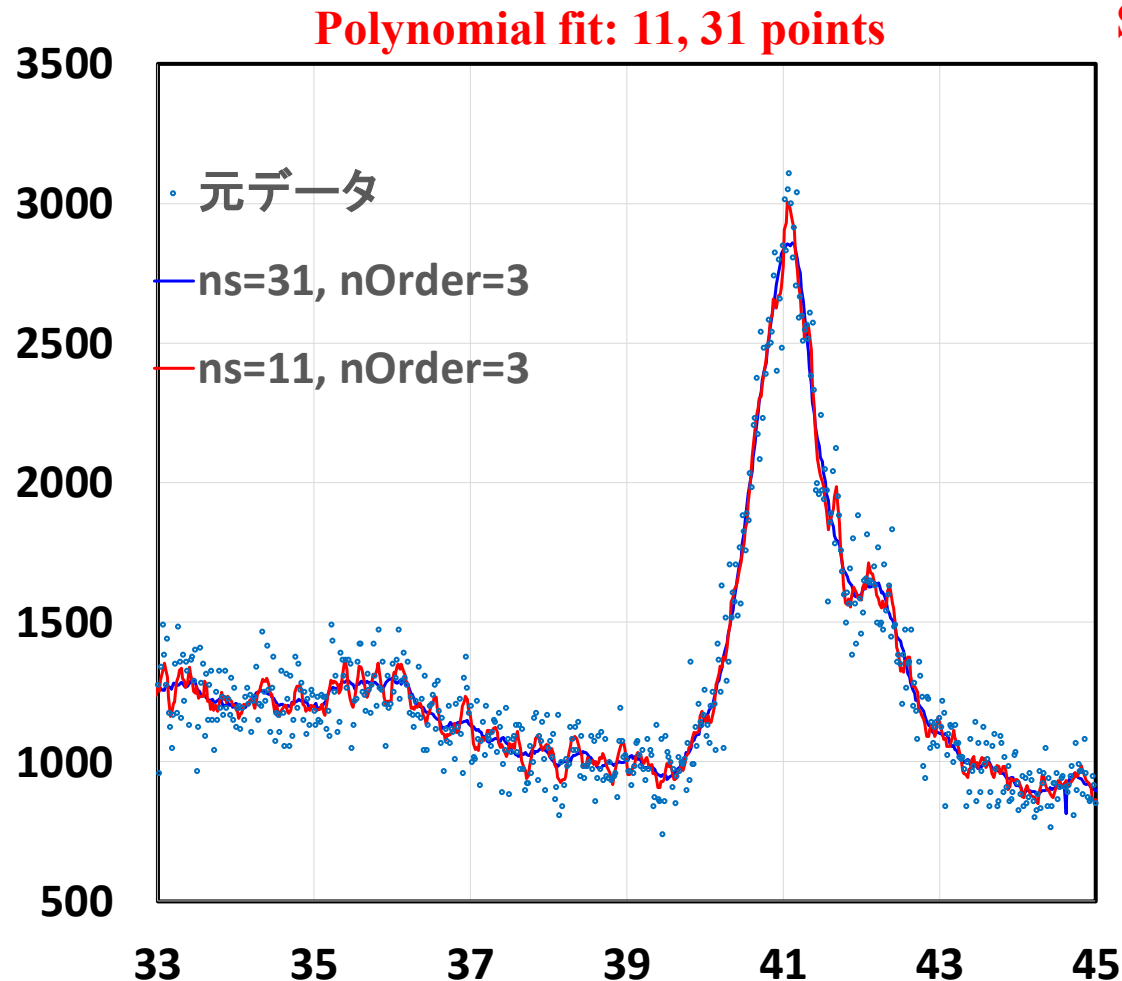
Poor S/N ratio XRD pattern



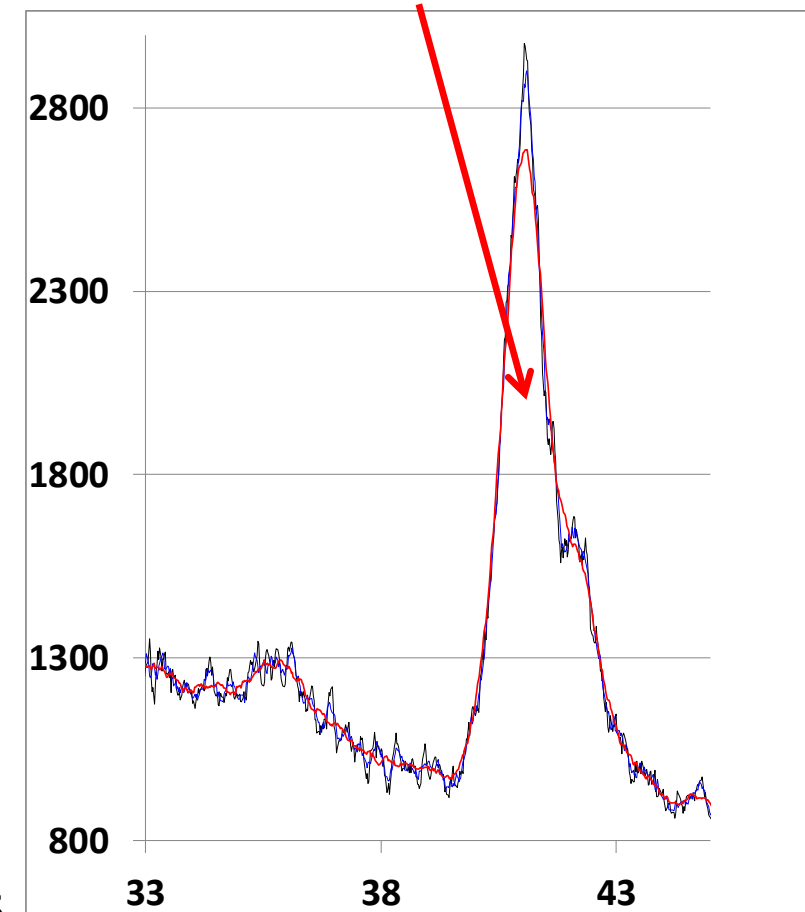
Smoothing: Polynomial fit method (多項式適合法)

Adopt n_{order} order polynomial to n_s sample points among the given n sample points, determined by LSQ

データに n_{order} 次多項式を最小自乗法で求め、標本点の値を内挿する



Simple moving average: 5, 11, 31 points



Weights of polynomial fit (Savizky-Golay method)

多項式適合法 (Savizky-Golay法) の重み

南茂夫, 科学計測のための波形データ処理, CQ出版社 (1986)

**Table 5.1 Weights for
order 2 and 3 polynomial fit**

Order 1: Simple moving average
Orders 2 and 3 have the same weights

| # of points N | 25 | 23 | 21 | 19 | 17 | 15 | 13 | 11 | 9 | 7 | 5 |
|----------------------|------|------|------|------|------|------|-----|-----|-----|-----|----|
| m=int(N/2) | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| -12 | -253 | | | | | | | | | | |
| -11 | -138 | -210 | | | | | | | | | |
| -10 | -33 | -105 | -171 | | | | | | | | |
| -9 | 62 | -10 | -76 | -136 | | | | | | | |
| -8 | 147 | 75 | 9 | -51 | -105 | | | | | | |
| -7 | 222 | 150 | 84 | 24 | -30 | -78 | | | | | |
| -6 | 287 | 215 | 149 | 89 | 35 | -13 | -55 | | | | |
| -5 | 342 | 270 | 204 | 144 | 90 | 42 | 0 | -36 | | | |
| -4 | 387 | 315 | 249 | 189 | 135 | 87 | 45 | 9 | -21 | | |
| -3 | 422 | 350 | 284 | 224 | 170 | 122 | 80 | 44 | 14 | -10 | |
| -2 | 447 | 375 | 309 | 249 | 195 | 147 | 105 | 69 | 39 | 15 | -3 |
| -1 | 462 | 390 | 324 | 264 | 210 | 162 | 120 | 84 | 54 | 30 | 12 |
| 0 | 467 | 395 | 329 | 269 | 215 | 167 | 125 | 89 | 59 | 35 | 17 |
| 1 | 462 | 390 | 324 | 264 | 210 | 162 | 120 | 84 | 54 | 30 | 12 |
| 2 | 447 | 375 | 309 | 249 | 195 | 147 | 105 | 69 | 39 | 15 | -3 |
| 3 | 422 | 350 | 284 | 224 | 170 | 122 | 80 | 44 | 14 | -10 | |
| 4 | 387 | 315 | 249 | 189 | 135 | 87 | 45 | 9 | -21 | | |
| 5 | 342 | 270 | 204 | 144 | 90 | 42 | 0 | -36 | | | |
| 6 | 287 | 215 | 149 | 89 | 35 | -13 | -55 | | | | |
| 7 | 222 | 150 | 84 | 24 | -30 | -78 | | | | | |
| 8 | 147 | 75 | 9 | -51 | -105 | | | | | | |
| 9 | 62 | -10 | -76 | -136 | | | | | | | |
| 10 | -33 | -105 | -171 | | | | | | | | |
| 11 | -138 | -210 | | | | | | | | | |
| 12 | -253 | | | | | | | | | | |
| Normalization factor | 5175 | 4025 | 3059 | 2261 | 1615 | 1105 | 715 | 429 | 231 | 105 | 35 |

Weights for order 2 and 3 using (2m+1) points ((2m+1)点を用いた2,3次多項式適合の重み)

$$w_{23}(j) = 3m(m+1) - 1 - 5j^2$$

$$j = -m, \dots, -1, 0, 1, \dots, m$$

$$W_{23} = (4m^2 - 1)(2m + 3)/3$$

Calculation

Simple moving average (2m+1 points)

$$y_{i,smoothed} = \frac{1}{2m+1} \sum_{j=i-m}^{i+m} y_j$$

Weighted moving average (2m+1 points)

$$y_{i,smoothed} = \sum_{j=i-m}^{i+m} w_j y_j / \sum_{j=i-m}^{i+m} w_j$$

Order 2 and 3 polynomial fit using (2m+1) points

$$w_{23}(j) = 3m(m+1) - 1 - 5j^2 \quad j = -m, \dots, -1, 0, 1, \dots, m$$

$$W_{23} = (4m^2 - 1)(2m + 3)/3$$

$$y_{i,smoothed} = \frac{1}{W_{23}} \sum_{j=i-m}^{i+m} w_{23}(j) y_j$$

Program: smoothing.py

Usage: python smoothing.py

