Computational Materials Science (計算材料学特論)

Lecture materials updated (this morning, 8:45)

http://conf.msl.titech.ac.jp/Lecture/ComputationalMaterialsScience/index-numericalanalysis.html

2024年度Q2 計算材料学特論 (資料: 英語+日本語版)

Computational Materials Science 2023 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 on numerical analysis (by Kamiya)

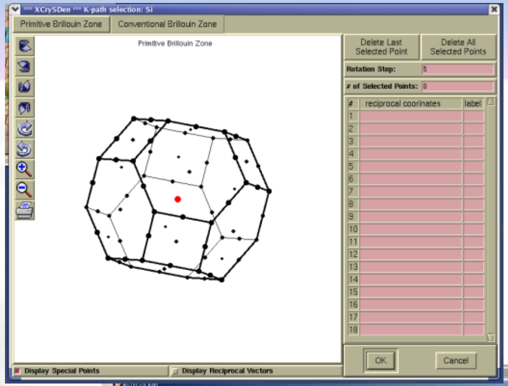
講義で使うプレゼン資料は、Other related programsの下にあります
Lecture presentation slides will be found after the python tips section.

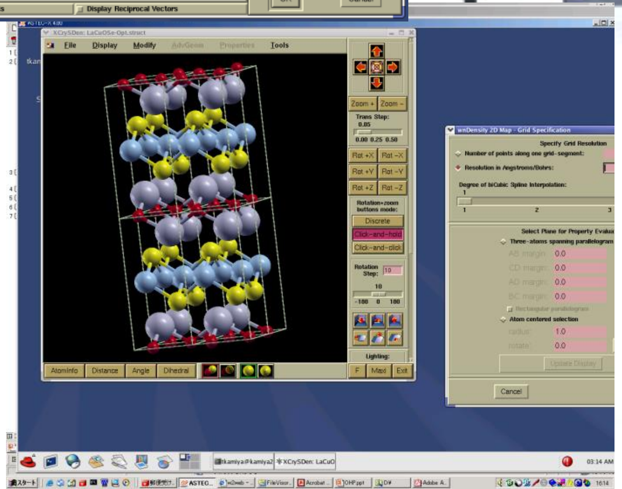
Update News:

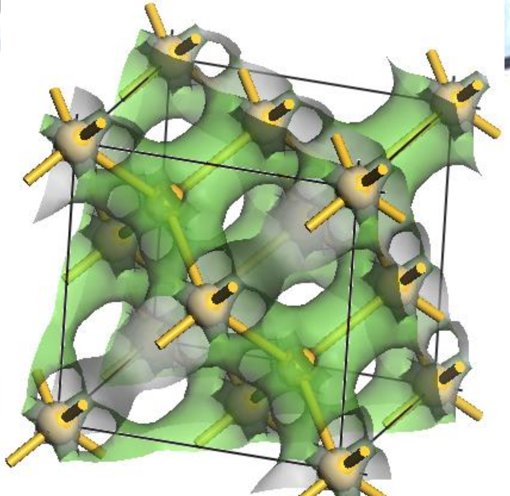
- June 18, 7:06 Lecture materials on June 18 have been uploaded (20240618Diffeq.zip)
- June 18, 9:06 Lecture materials on June 18 have been uploaded (20240616Diffeq.zip)
- June 14, 13:07 Lecture materials on June 14 have been updated (20240614DifferentialIntegration2.zip)
- June 14, 8:45 Lecture materials on June 14 have been updated
- June 13, 11:48 Lecture materials on June 14 have been uploaded
- June 12, 14:13 Lecture materials on June 11 have been updated (20240612ComputerAndErorrSouorces.zip)
- June 11, 8:24 Lecture materials on June 11 have been updated
- June 07, 9:23 Lecture materials on June 11 have been uploaded.

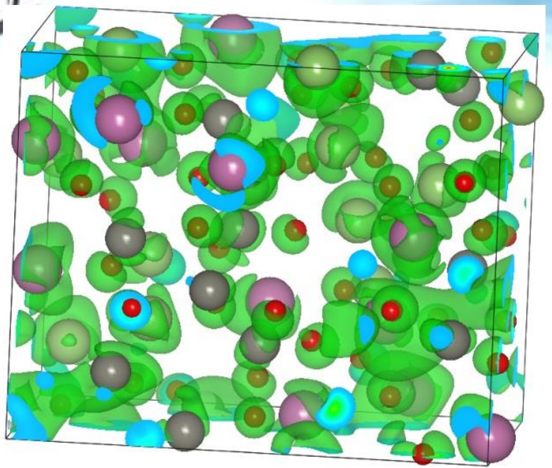
Computational Materials Science
計算材料学特論

Toshio Kamiya
神谷利夫









Class Schedule

Lecture materials (Kamiya's part): http://conf.msl.titech.ac.jp/Lecture/

http://conf.msl.titech.ac.jp/Lecture/ComputationalMaterialsScience/index-numericalanalysis.html

#01	June 11 (Tue)	Kamiya (Fundamental of computer, Sources of errors (コンピュータの基礎、誤差))
#02	June 14 (Fri)	Kamiya (Numerical differentiation/integration (数値微分/積分))
#03	June 18 (Tue)	Kamiya (Numerical integration (数値積分), Differential equation (微分方程式), Molecular dynamics (分子動力学法))
#04	June 21 (Fri)	Kamiya (Interpolation (補間), Smoothing (平滑化), Linear least-squares method (線形最小二乗法), Optimization (最適化))
#05	June 25 (Tue)	Kamiya (Numerical solutions of equations (方程式の数値解法), Nonlinear optimization (非線形最適化))
#06	June 28 (Fri)	Kamiya, Matrix (Fourier transformation (フーリエ変換), 行列)
	July 2 (Tue)	**No lecture (休講)**
#07	July 5 (Fri)	Kamiya, Review (復習)
#08	July 9 (Tue)	Sasagawa (Review of quantum theory 1: 量子論おさらい1)
#09	July 12 (Fri)	Sasagawa (Review of quantum theory 2: 量子論おさらい2)
#10	July 16 (Tue)	Sasagawa (First principles calculations: basics 1 第一原理計算:基礎1)
#11	July 19 (Fri)	Sasagawa (First principles calculations: basics 2 第一原理計算:基礎2)
#12	July 23 (Tue)	Sasagawa (First principles calc.: applications 1 第一原理計算:応用1)
#13	July 26 (Tue)	Sasagawa (First principles calc.: applications 2 第一原理計算:応用2)
#14		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 14

- **Submit electronic file(s) via T2SCHOLAR until June 16**

  (If T2SCHOLAR 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) Calculate d$E(k)$/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 Å.

(ii) Compare the results obtained by different h.

See band_answer.xlsx

Effective mass

LCAO band

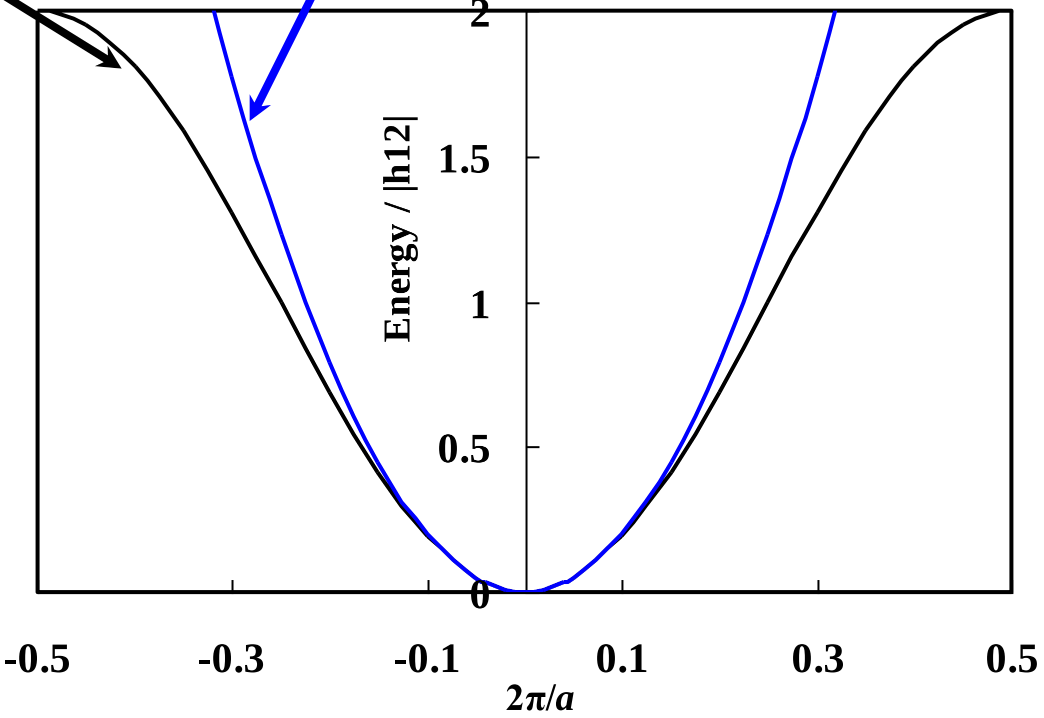
$$E(k)=\varepsilon_1-2|h_{12}|\cos(ka)\sim\varepsilon_1-2|h_{12}|+|h_{12}|a^2k^2+O((ka)^4)$$

Free electron model

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

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

$$\boxed{m^*=\frac{\hbar^2}{2|h_{12}|a^2}}$$



Effective mass

k represents fractional coordinate in reciprocal unit cell:
generally expressed in the range **[-½ ½]**

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

Note ***E*(*k*) is in eV**

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

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

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

Numerical differentiation: Accuracy

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

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

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

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

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

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

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・・・)

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

$$\sim \frac{\frac{x(t+2\Delta t)-x(t+\Delta t)}{\Delta t} - \frac{x(t+\Delta t)-x(t)}{\Delta t}}{\Delta t} = \frac{x(t+2\Delta t)-2x(\boldsymbol{t+\Delta t})+x(t)}{\Delta t^2}$$

Use central difference

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

$$\sim \frac{\frac{x(t+\Delta t)-x(t)}{\Delta t} - \frac{x(t)-x(t-\Delta t)}{\Delta t}}{\Delta t} = \frac{x(t+\Delta t)-2x(\boldsymbol{t})+x(t-\Delta t)}{\Delta t^2}$$

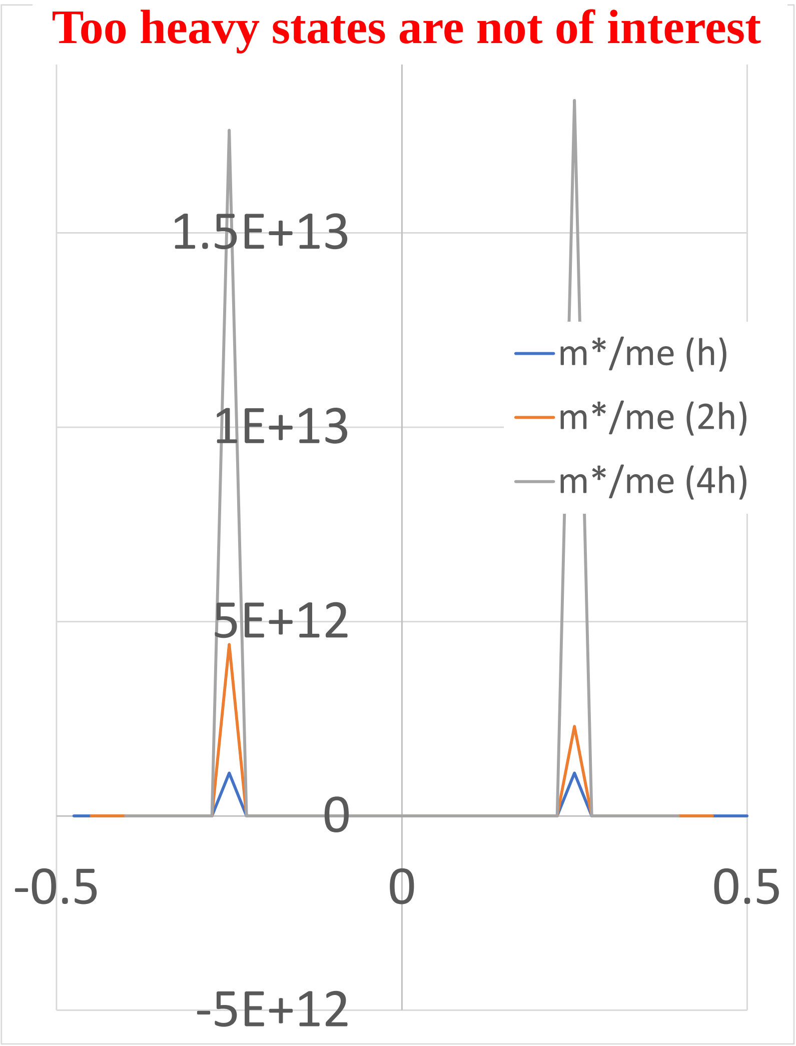
Note: These two formula are offset in *t* by *Δt*

２つの式では、横軸が*Δt*ずれるので注意！

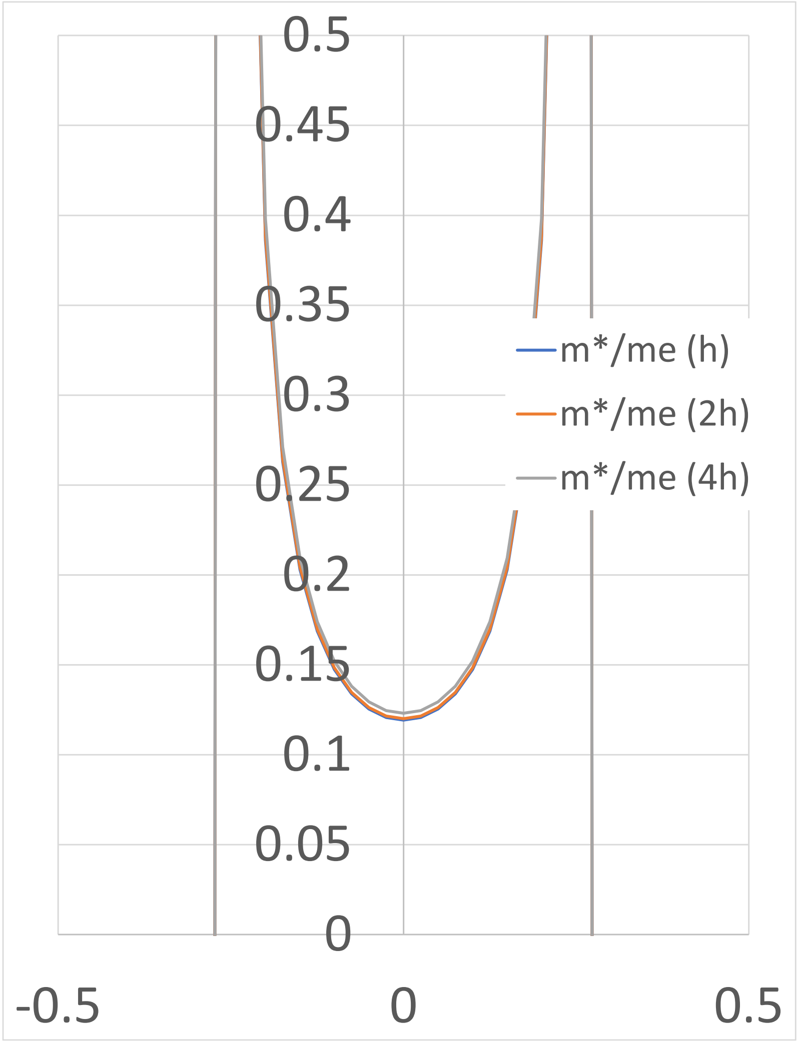
Answer: How to present?

band-answer.xlsx

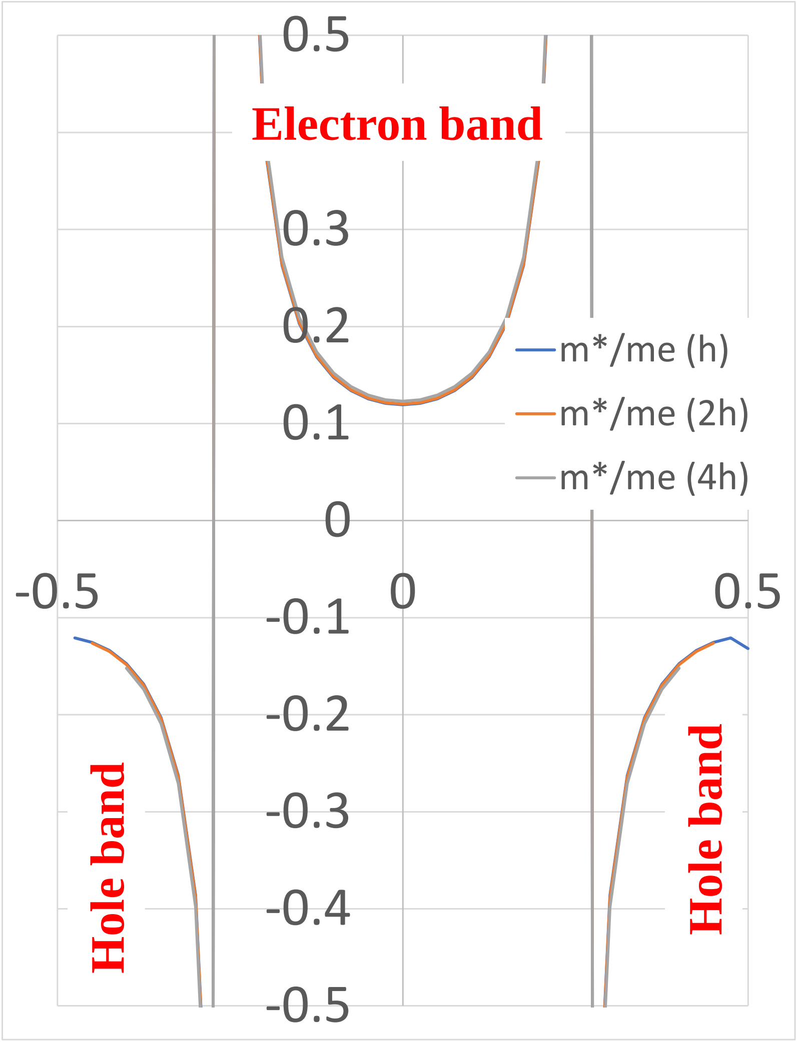
Full range



Electron only ($m^* > 0$)

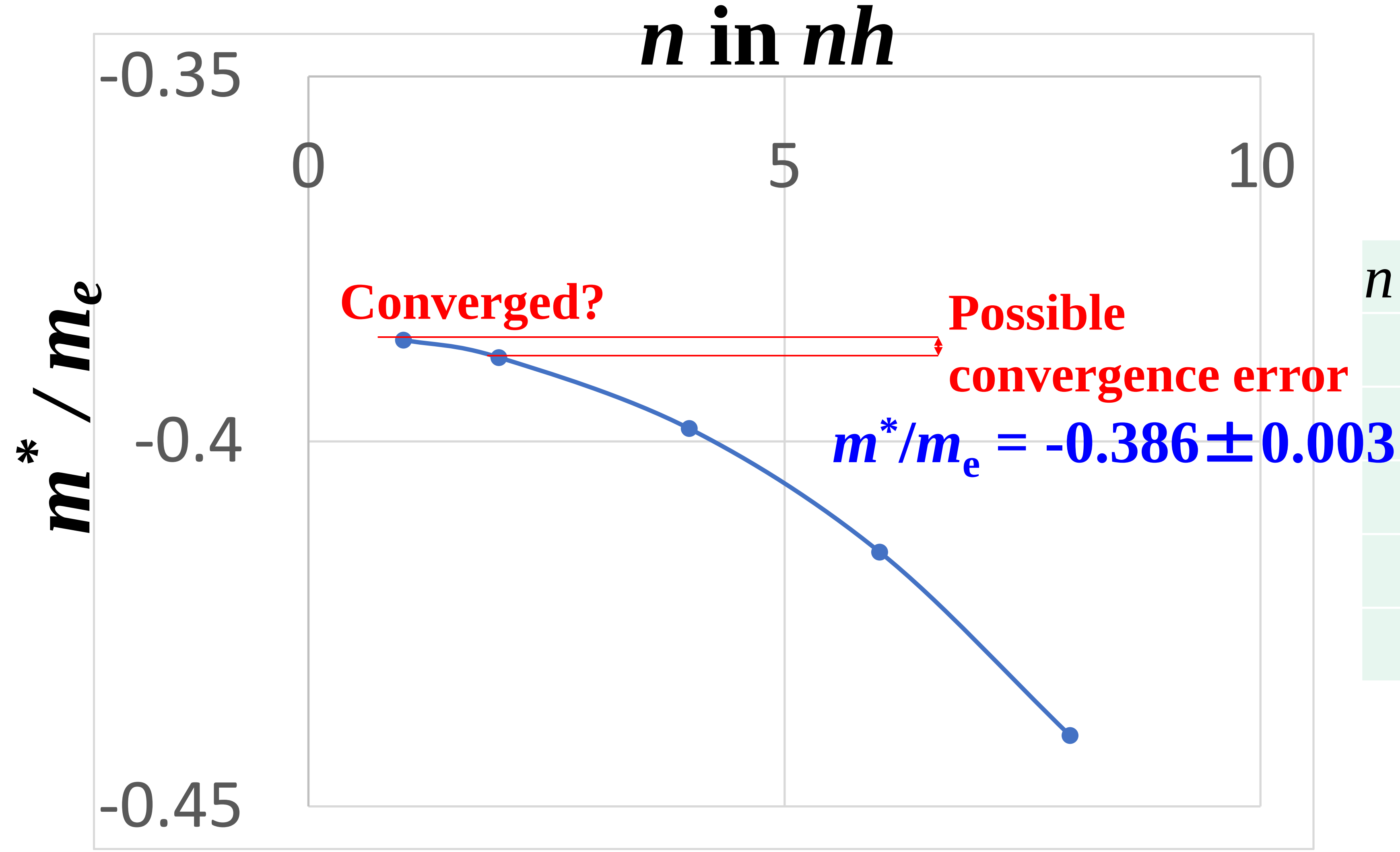


Electron and holes



Accuracy and convergence check

band-answer.xlsx

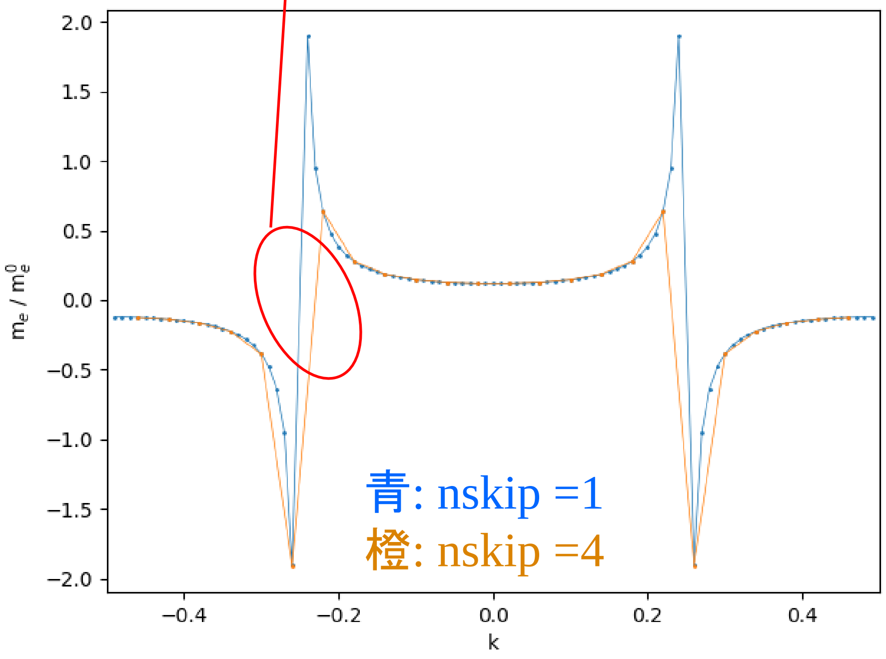


n in *nh*	m^*/m_e
1	-0.386097987
2	-0.388489462
4	-0.398234965
6	-0.415138589
8	-0.440277749

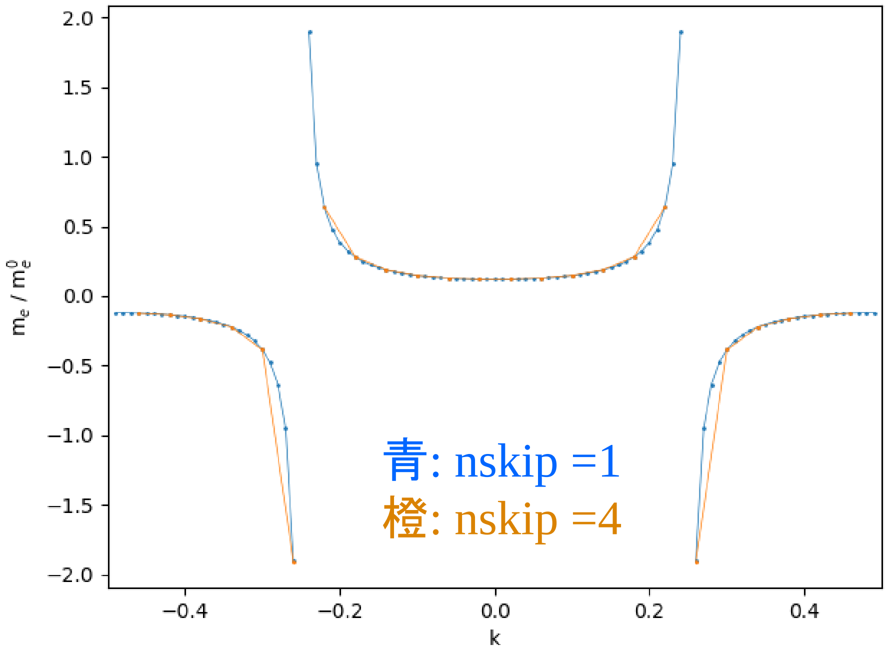
Python program (抜粋)

python EffectiveMass.py

These lines should not be drawn
描いてはいけない線



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 18

- **Submit electronic file(s) via T2SCHOLAR until June 20**

  (If T2SCHOLAR 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.

$x(0) = 1.0$

Numeral integration (quadrature)

数値積分 (求積)

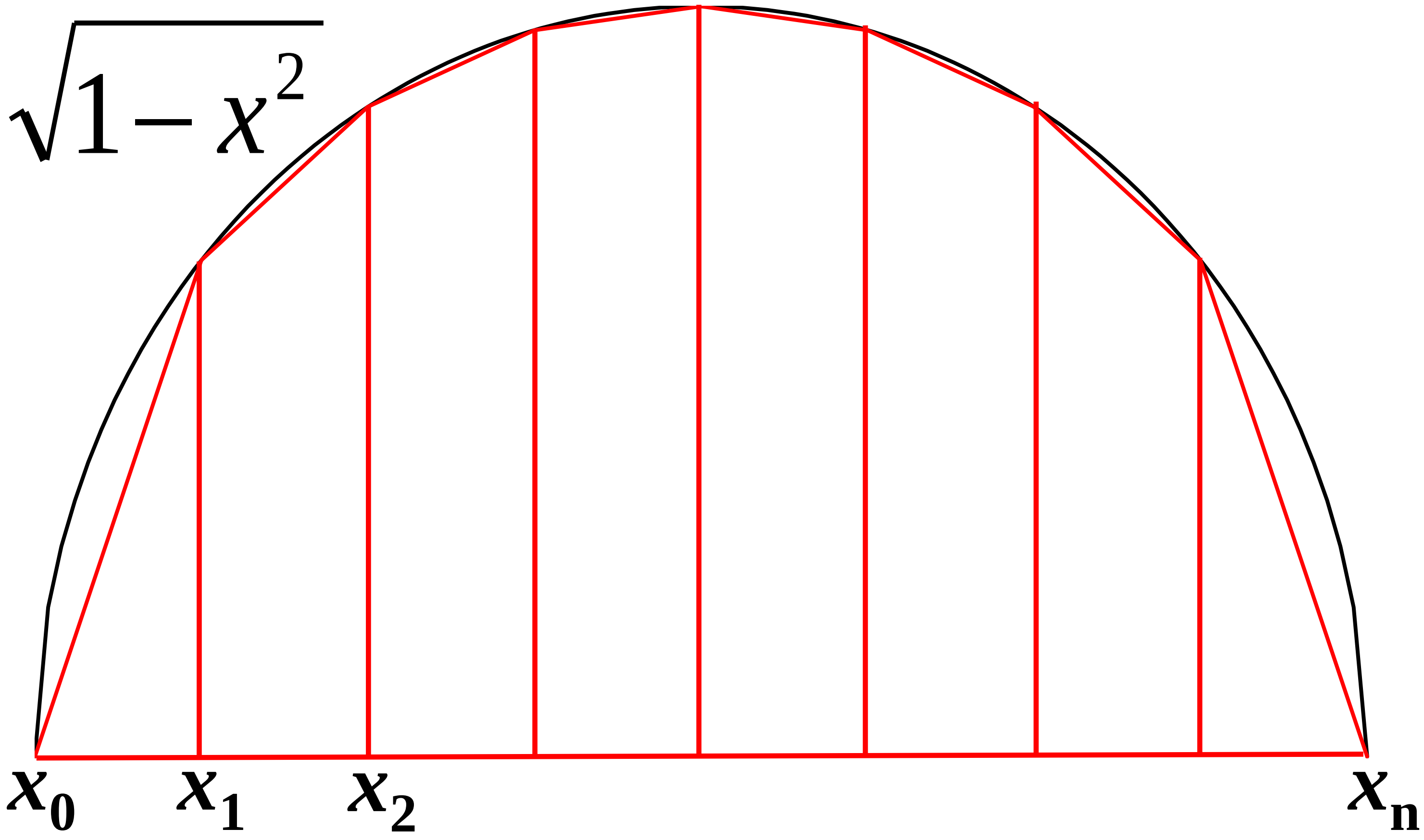
Variable-Conversion type algorithms

Problem for integration with anomaly points

(特異点を含む場合の問題)

$$F(x) = \int_{x_0}^{x} g(x')dx'$$

$$g(x) = \sqrt{1-x^2}$$



Very large errors for large $|f'(x)| / |f''(x)|$

Variable conversion type: Double exponential type formula (変数変換型: 二重指数関数型公式)

戸田英雄, 小野令美，入門 数値計算, オーム社 (昭和58年)

Good for integral including anomaly points at the ends and for infinite range

端点に特異点のある積分や、無限積分に有効

e.g., finite range integral is converted to the infinite range (-∞, ∞) by variable conversion

有限区間積分の場合は、変数変換により無限積分にする

Variable conversion *x* => *u*: $\boldsymbol{x = \varphi(u)}$

Calculate by the Trapezoid formula

$$S = \int_{x_0}^{x_1} f(x)dx = \int_{u_0}^{u_1} f\big(\varphi(u)\big)\frac{d\varphi(u)}{du}du = h_u \sum_{n=-\infty}^{\infty} f\big(\varphi(u_n = nh)\big)\varphi'(u_n)$$

Iri-Moriguchi-Takasawa (IMT) formula

伊理・森口・高沢(IMT)の公式

戸田英雄, 小野令美，入門 数値計算, オーム社 (昭和58年)

Good for finite range integral including anomaly points at the ends and for infinite range

By variable conversion (変数変換)

$$x = \phi(u) = \frac{1}{Q}\int_0^u \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt \qquad \phi'(u) = \frac{1}{Q}\exp\left(-\frac{1}{t} - \frac{1}{1-t}\right)$$

$$Q = \int_0^1 \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt \quad = 0.00702985841$$

an integral of *f*(*x*) is converted to

$$\int_0^1 f(x)dx = \int_0^1 f(\phi(u))\phi'(u)du$$

, and then calculate the integral by the Trapezoid formula

1. **Convert the integration range to [0, 1] by $x = (x' - a) / (b - a)$**

$$\int_a^b f(x')dx' = (b-a)\int_0^1 f(x)dx$$

2. **Calculate integration points $x_k = \phi(k/n)$ and weights $w_k = \phi'(k/n)$**

3. **Calculate** $I = h\sum_{k=1}^{n-1} f(x_k)w_k$ **($h = (b - a)/n$)**

Iri-Moriguchi-Takasawa (IMT) formula

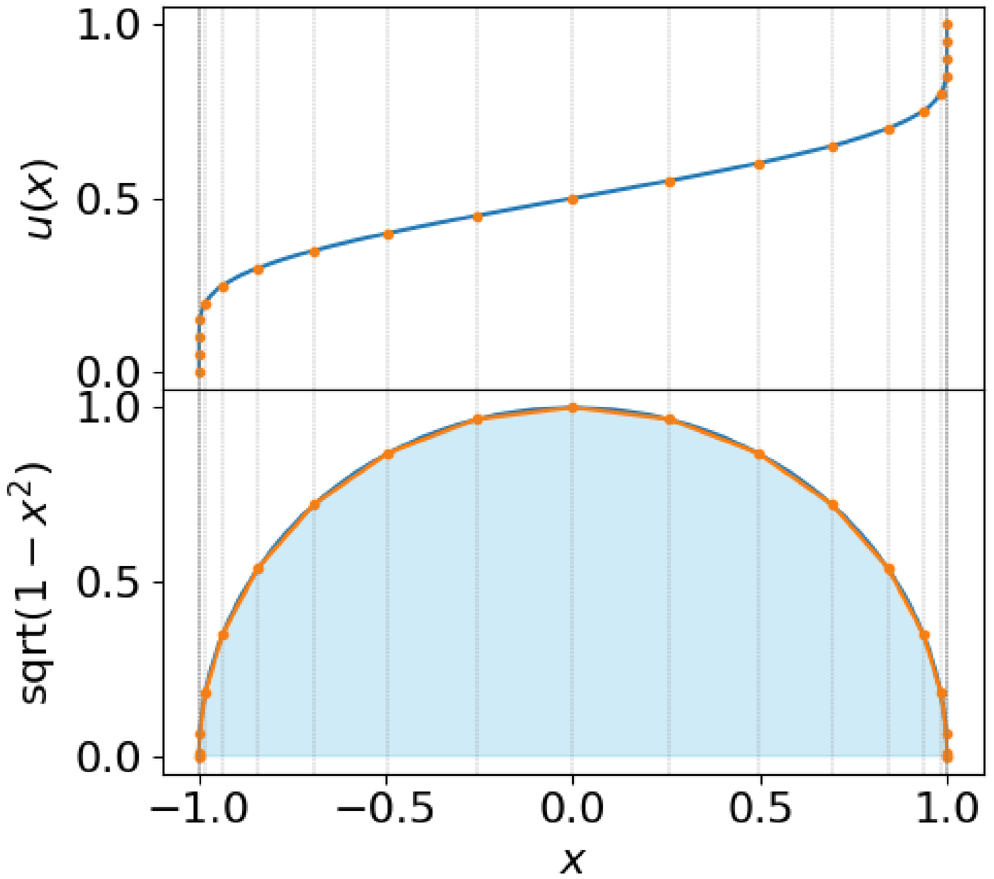
伊理･森口･高沢(IMT)の公式

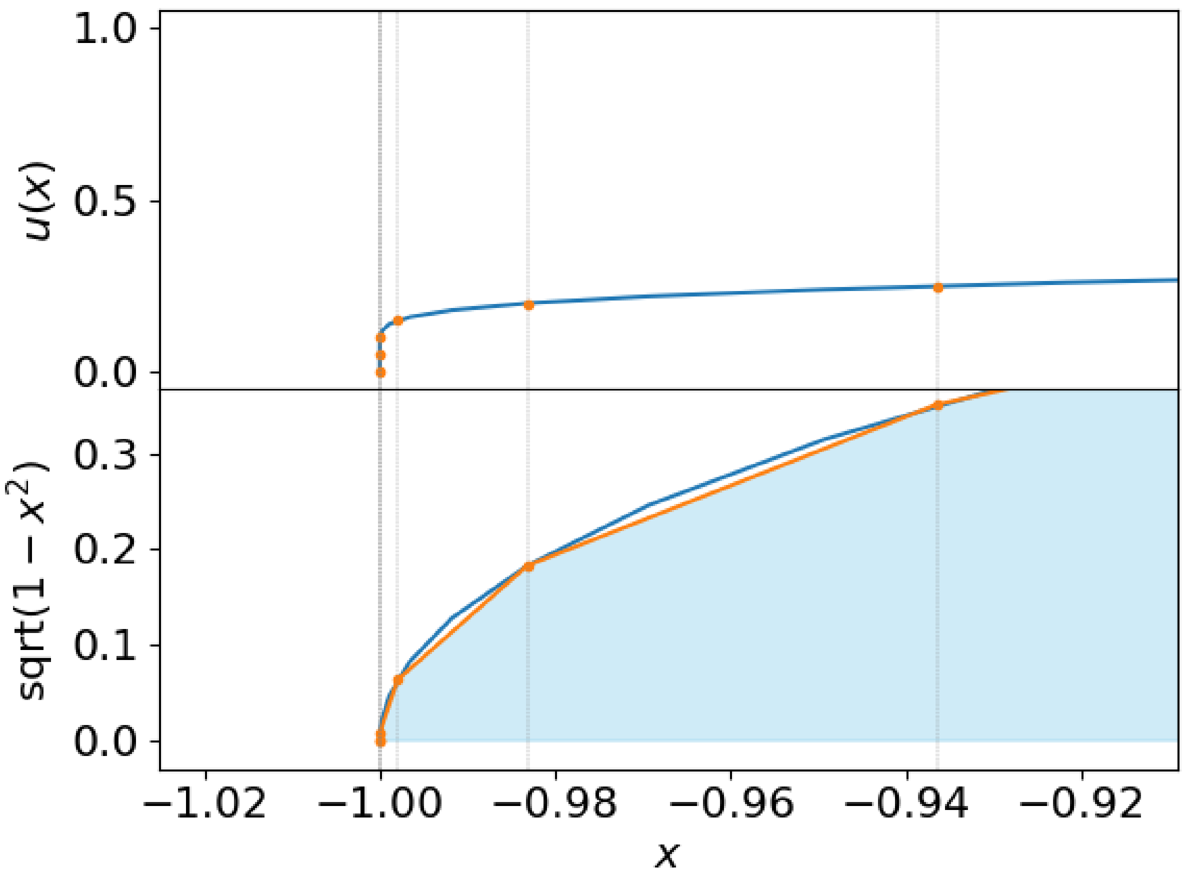
> python integ_imt.py

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)

$$x_n = \varphi(u_n = nh) = \frac{1}{Q}\int_0^{nh} \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt$$

Q = 0.00702985841





Variable conversion type: Double exponential type formula

(変数変換型: 二重指数関数型公式)

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)

For $\int_{-1}^{1} f(x)dx$

$$x_n = \varphi(u) = \tanh\left[\frac{\pi}{2}\sinh(u)\right] \qquad \varphi'(u) = \frac{\pi}{2}\frac{\cosh u}{\cosh^2\left((\pi/2)\sinh u\right)}$$

For $\int_{0}^{\infty} f(x)dx$

$$x_n = \varphi(u) = \exp\left[\frac{\pi}{2}\sinh(u)\right] \qquad \varphi'(u) = \frac{\pi}{2}\cosh u \exp\left(\frac{\pi}{2}\sinh u\right)$$

For $\int_{0}^{\infty} f(x)dx$ **where** ***f***(***x***) **includes exp(-*x*) type factor**

$$x_n = \varphi(u) = \exp\left[\frac{\pi}{2}(u - \exp(-u))\right] \qquad \varphi'(u) = \frac{\pi}{2}(1 + \exp(-u))\exp\left(\frac{\pi}{2}(u - \exp(-u)\right)$$

For $\int_{-\infty}^{\infty} f(x)dx$

$$x_n = \varphi(u) = \sinh\left[\frac{\pi}{2}\sinh(u)\right] \qquad \varphi'(u) = \frac{\pi}{2}\cosh u \cosh\left(\frac{\pi}{2}\sinh(u)\right)$$

Variable conversion type: Double exponential type formula

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)

For $\int_{-1}^{1} f(x)dx = \int_{-1}^{1} f(u)\varphi'(u)du = h_u \sum_i f\big(\varphi(u_i = ih_u)\big)\varphi'(u_i)$

x range: $[-1,1]$

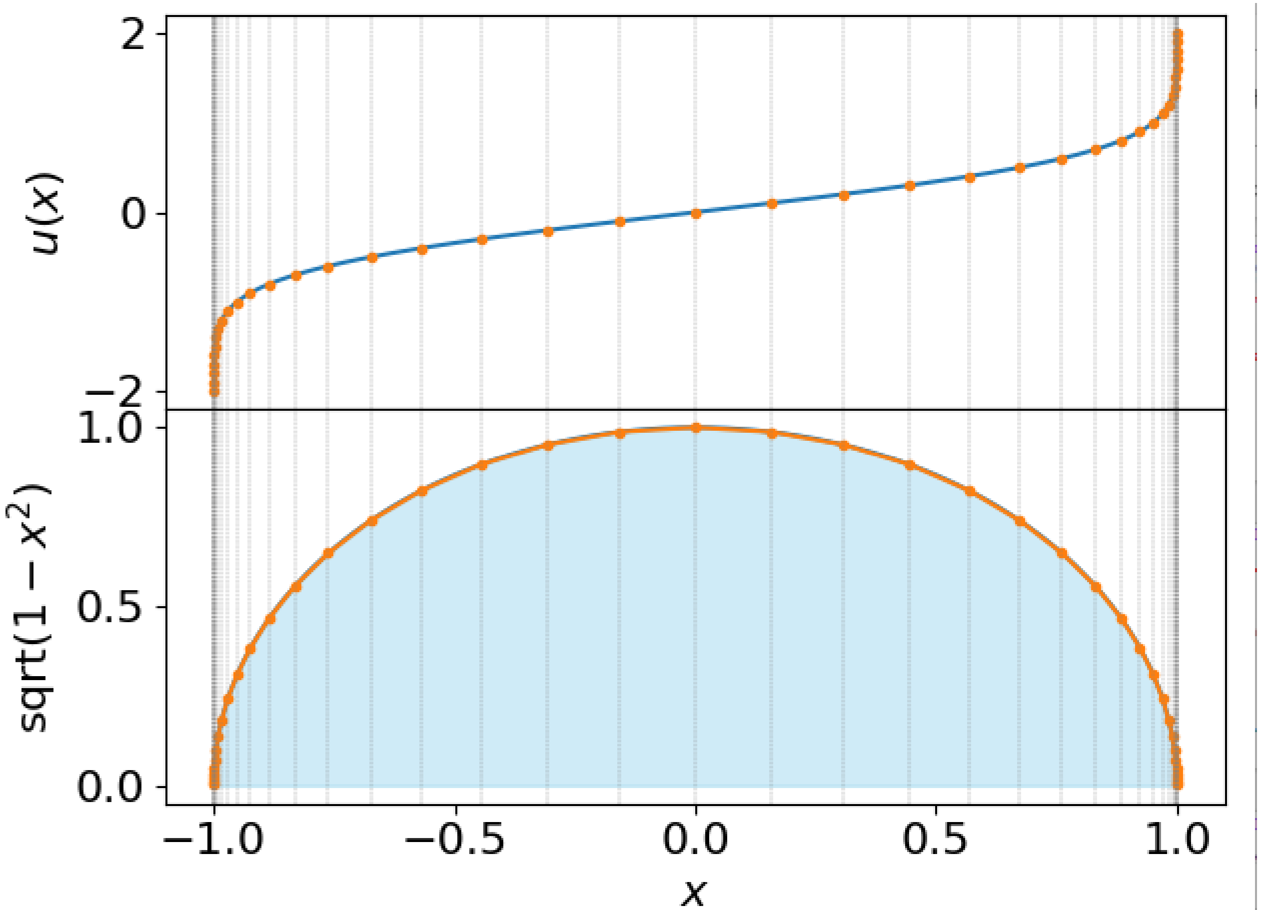
> python integ_double_exp_-1_1.py

u range: $[-2,2]$ => x range ~ $[-1,1]$

$$f(x) = \sqrt{1-x^2}$$

$$x = \varphi(u) = \tanh\left[\frac{\pi}{2}\sinh(u)\right]$$

$$\varphi'(u) = \frac{\pi}{2}\frac{\cosh u}{\cosh^2((\pi/2)\sinh u)}$$



Variable conversion type: Double exponential type formula

戸田英雄, 小野令美，入門 数値計算, オーム社 (昭和58年)

For $\int_0^\infty f(x)dx = \int_0^\infty f(u)\varphi'(u)du = h_u \sum_i f\big(\varphi(u_i = ih_u)\big)\varphi'(u_i)$

x range: $[0, \infty]$

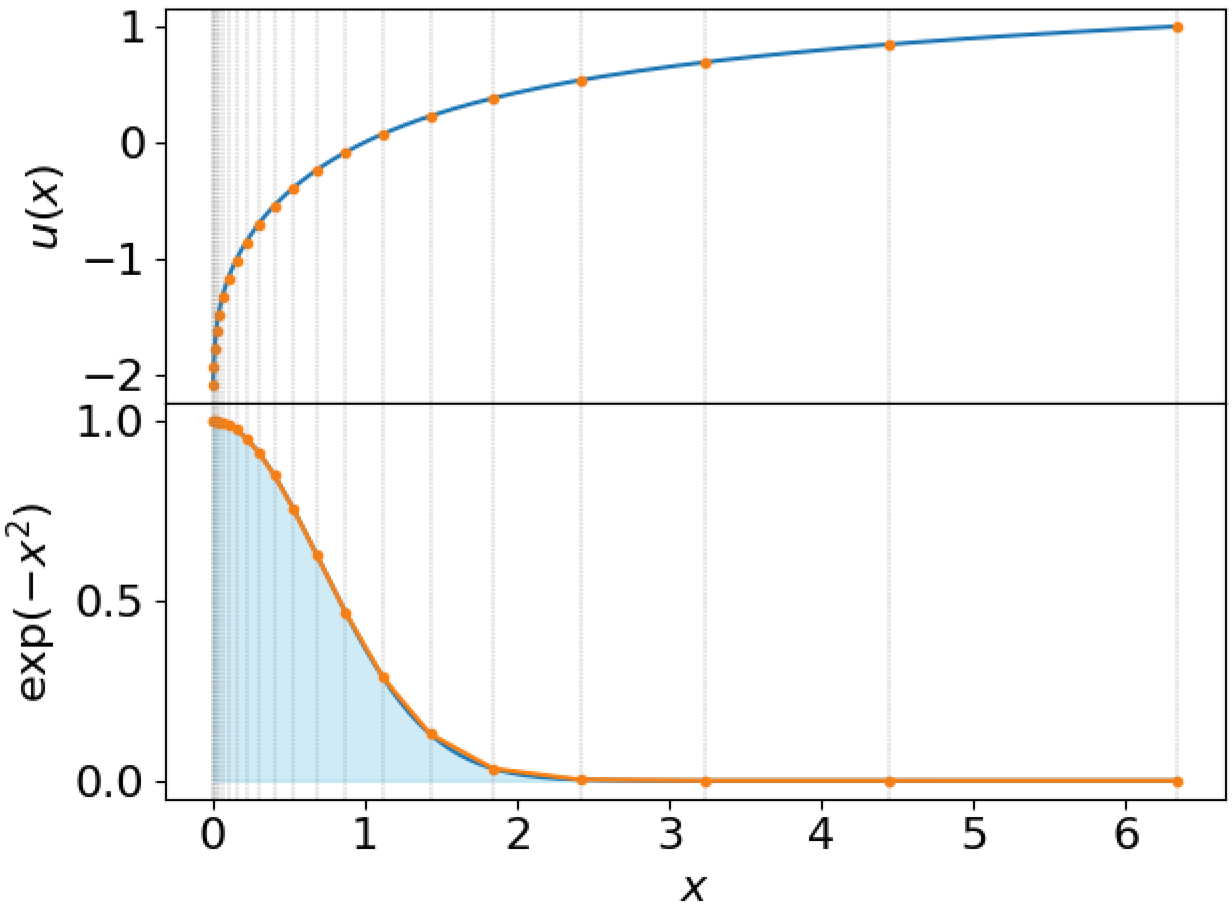
> python integ_double_exp_0_inf.py

u range: $[-2{,}1]$ => x range ~ [0,6]

$f(x) = \exp(-x^2)$

$x_n = \varphi(u) = \exp\left[\frac{\pi}{2}\sinh(u)\right]$

$\varphi'(u) = \frac{\pi}{2}\cosh u \exp\left(\frac{\pi}{2}\sinh u\right)$



Variable conversion type: Double exponential type formula

戸田英雄, 小野令美，入門 数値計算, オーム社 (昭和58年)

For $\int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty} f(u)\varphi'(u)du = h_u \sum_{n=-\infty}^{\infty} f\big(\varphi(u_n = nh_u)\big)\varphi'(u_n)$

x range: $[-\infty, \infty]$

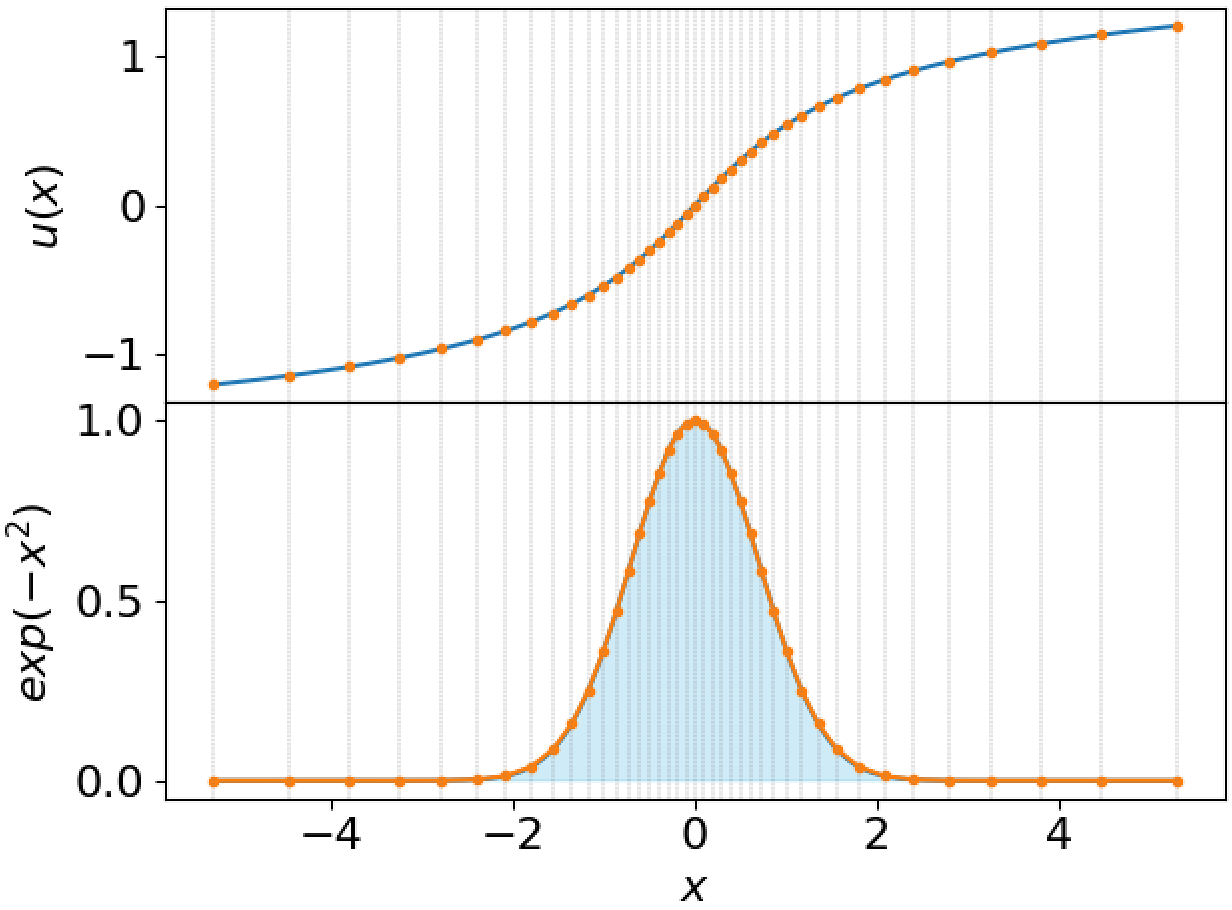
u range: $[-1.2, 1.2]$ => x range ~ $[-5, 5]$

$f(x) = \exp(-x^2)$

$x = \varphi(u) = \sinh\left[\frac{\pi}{2}\sinh(u)\right]$

$\varphi'(u) = \frac{\pi}{2}\cosh u \cosh\left(\frac{\pi}{2}\sinh(u)\right)$

> python integ_double_exp_inf_inf.py



Error for integration with anomaly points

$$S = \int_{-1}^{1} \sqrt{1 - x^2}\, dx$$ **Exact: π/2 = 1.5707963**

nDivided	Rieman	Trapezoid	Simpson	Simpson 3/8	Bode	Romberg	Cubic Spline	Order 3 Gauss-Legendre	IMT	Double exp*
2	5.71E-01	5.71E-01	2.37E-01			2.37E-01		2.08E-02	1.03E+00	1.5708035
3	3.14E-01	3.14E-01		1.57E-01					1.74E-01	-0.52993
4	2.05E-01	2.05E-01	8.28E-02		7.24E-02	7.24E-02	6.93E-02	7.24E-03	2.72E-02	0.1417235
5	1.47E-01	1.47E-01					5.26E-02		3.40E-03	-0.0288253
6	1.12E-01	1.12E-01	4.48E-02	5.47E-02			3.97E-02	3.92E-03	2.99E-03	0.0050382
7	8.90E-02	8.90E-02					3.17E-02		8.70E-04	-0.0007911
8	7.29E-02	7.29E-02	2.90E-02		2.54E-02	2.47E-02	2.60E-02	2.54E-03	2.37E-05	0.0001138
9	6.12E-02	6.12E-02		2.96E-02			2.18E-02		7.98E-05	-1.55E-05
10	5.23E-02	5.23E-02	2.07E-02				1.87E-02	1.81E-03	4.90E-05	1.99E-06
11	4.53E-02	4.53E-02					1.62E-02		1.32E-05	-2.49E-07
12	3.98E-02	3.98E-02	1.57E-02	1.92E-02	1.38E-02		1.42E-02	1.38E-03	4.53E-06	2.82E-08
13	3.53E-02	3.53E-02					1.26E-02		8.86E-06	-6.05E-09
14	3.16E-02	3.16E-02	1.25E-02				1.13E-02	1.09E-03	6.87E-06	-3.54E-09
15	2.85E-02	2.85E-02		1.37E-02			1.02E-02		2.03E-06	-5.65E-09
16	2.59E-02	2.59E-02	1.02E-02		8.95E-03	8.62E-03	9.25E-03	8.93E-04	1.23E-05	-7.57E-09
17	2.36E-02	2.36E-02					8.45E-03		2.22E-06	-9.79E-09
18	2.17E-02	2.17E-02	8.54E-03	1.04E-02			7.76E-03	7.48E-04	1.05E-05	-1.22E-08
19	2.00E-02	2.00E-02					7.15E-03		1.21E-05	-1.48E-08
20	1.85E-02	1.85E-02	7.29E-03		6.40E-03		6.63E-03	6.38E-04	1.12E-05	-1.75E-08
32						3.04E-03				-5.18E-08

* 変換積分範囲は u = [-2.0, 2.0]

Density of states and carrier density in metal

Fermi-Dirac function

$$f(e) = \frac{1}{\exp(\beta(E-E_F))+1}$$

Density of states (DOS)

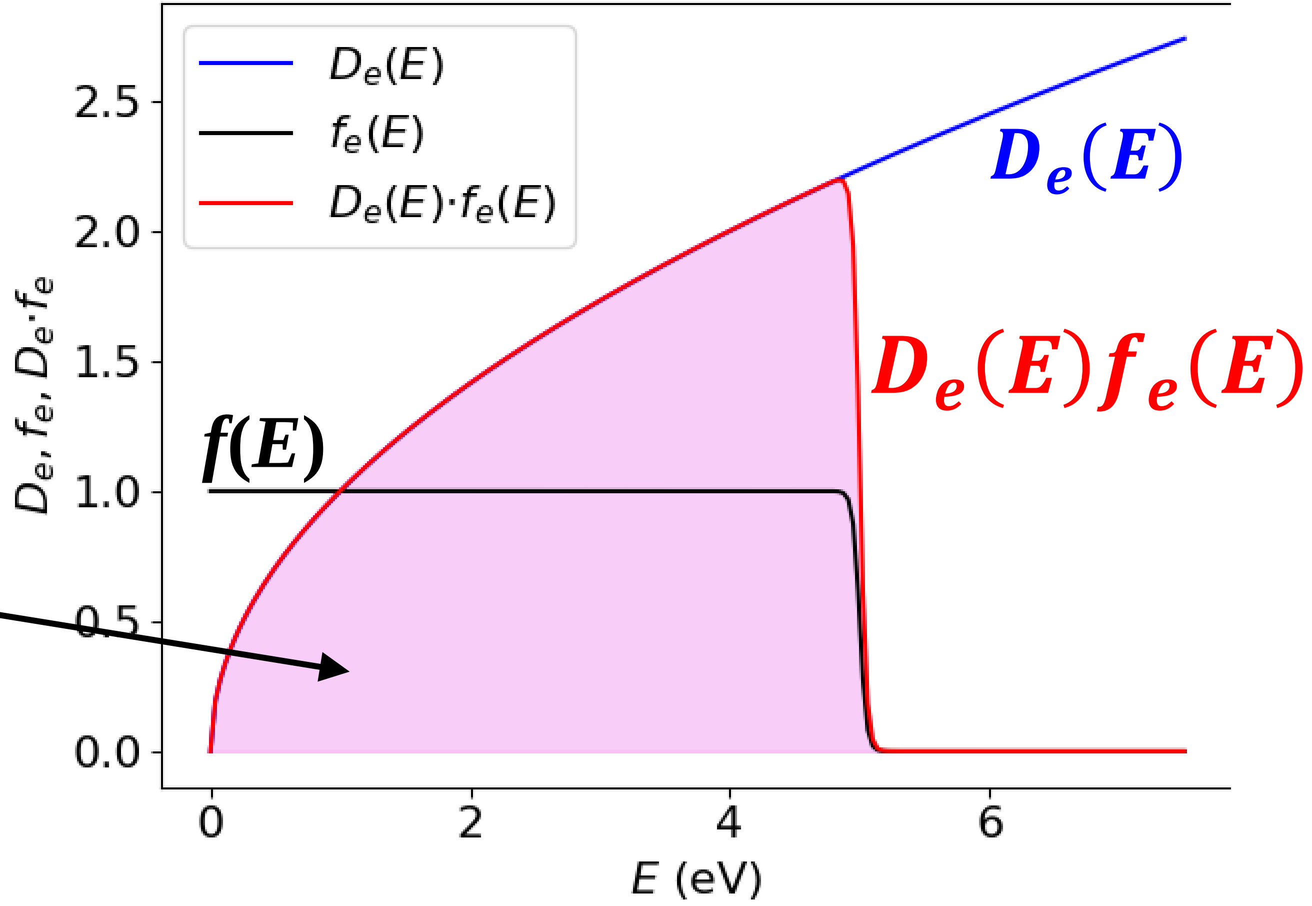
$$\boldsymbol{D_e(E) = D_{e0}\sqrt{E}}$$

$$\boldsymbol{D_{e0} = (2S+1)V\frac{2\pi(2m)^{3/2}}{h^3}}$$

Number of electrons in CB

$$\boldsymbol{N = \int_0^{\infty} D_e(E) f(E)\, dE}$$

$\boldsymbol{D_e(E)}$ is normalized by $\boldsymbol{D_{e0}}$



Program: Calculate N_e in metal

Issue: How to integrate $N(e)f(e)$ efficiently

- Wide integration range $E = 0 \sim E_F + \alpha k_B T \sim$ several eV (if precision is ~exp(-α))
- The range that needs precise calc is only around E_F with a range $\alpha k_B T \sim 0.1$ eV
- Function changes sharply around E_F, so integration mesh ΔE should be fine enough

(e.g., $\Delta E < \alpha k_B T / 100$, 1 meV)

=> We should not the same ΔE throughout the entire integration range $E = 0 \sim E_F + \alpha k_B T$

=> **Divide integration range**

(We can use the analytical form for the range $0 \sim E_F - \alpha k_B T$)

Usage: python N-integration-metal.py cal 300 5.0

Temperature at 300K, $E_F = 5.0$ eVで

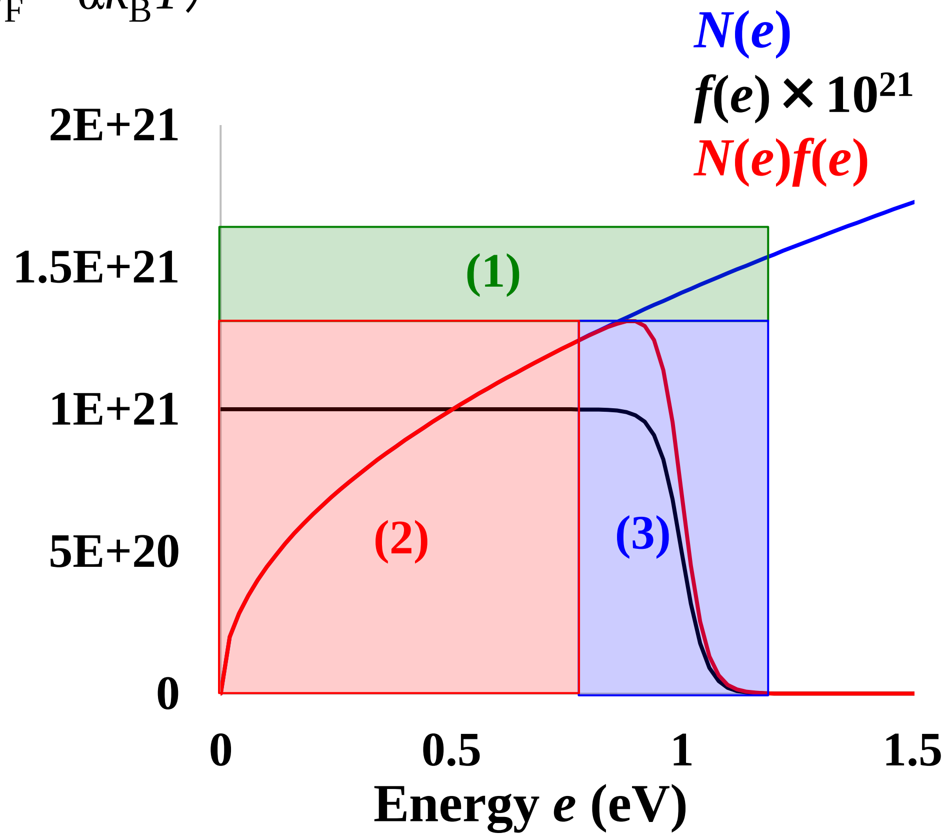
Measure time by repeating for 300 times

Precision 8 digits (epsrel = 1e-8), α = 6:

Integ. range	Time for 300 repetition
(1) $0 \sim E_F + \alpha k_B T$	0.109 秒
(2) $0 \sim E_F - \alpha k_B T$	0.063 秒
(3) $E_F - \alpha k_B T \sim E_F + \alpha k_B T$	0.016 秒

30% faster for (2) + (3)

Using analytic form for (2) is 10 times faster



Program: Debye model of heat capacity

$$C_V = 3Rf_D\left(\frac{\Theta_D}{T}\right) \qquad f_D(y) = \frac{3}{y^3}\int_0^y \frac{x^4 e^x}{\left(e^x - 1\right)^2}dx$$ **Debye function**

数値積分を使って計算: python の scipyモジュールの quad 関数 (適応積分法) を使ってみる

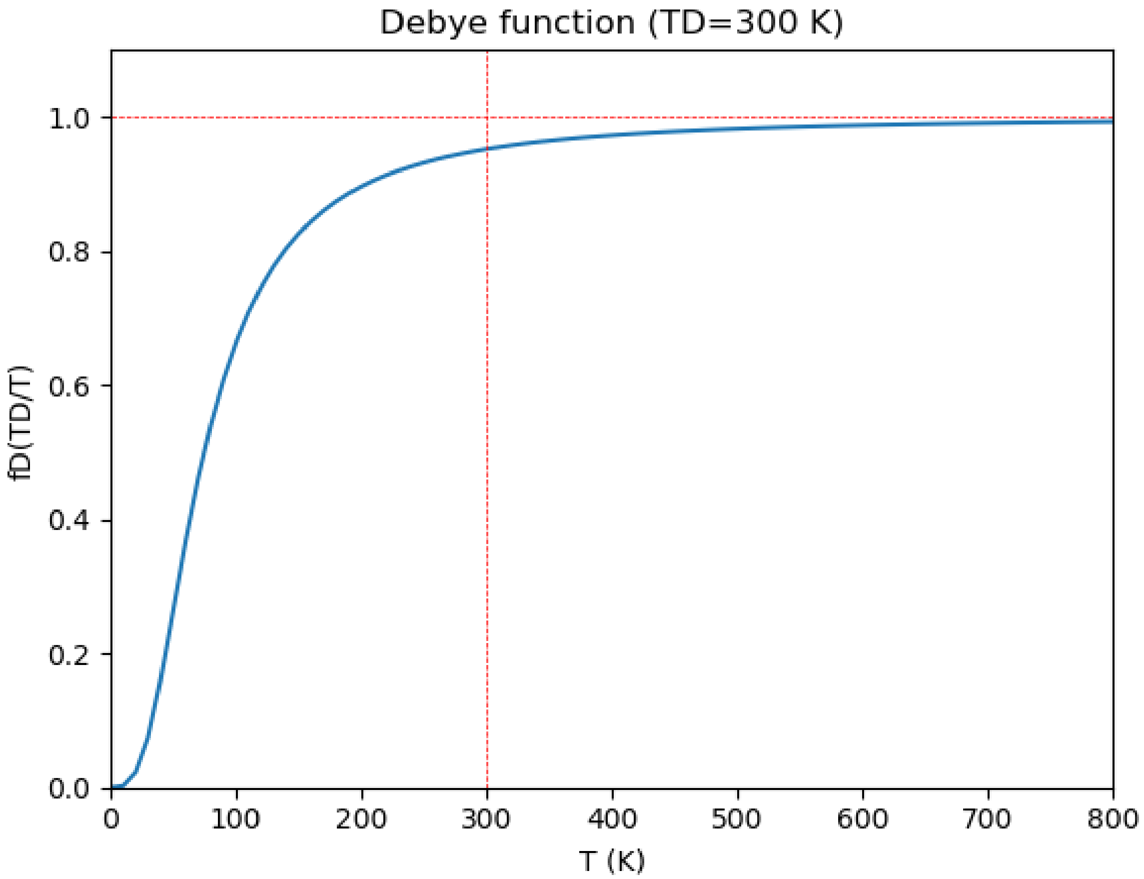
参考例 : https://org-technology.com/posts/integrate-function.html

数値積分の講義資料: http://conf.msl.titech.ac.jp/Lecture/python/index-numericalanalysis.html

python debye_function.py 300 0 500 10

Debye temperature 300 K

Temperate range 0 – 500 K, 10 K step



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)

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

$$mr^2\frac{d\theta}{dt} = l$$

l:a constant, conservation of angular momentum

$$\frac{1}{2}m\left(\frac{dr}{dt}\right)^2 + m\left(\frac{l^2}{2m^2r^2} - \frac{GM}{r}\right) = E$$

$$r(\theta) = \frac{b}{1+\varepsilon\cos(\theta-\delta)}$$

$$b = \frac{l^2}{mc}$$

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

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

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

τ_0, l_0: Time and length specific to the system
Chose so that T and R will the 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)

$$\frac{d^2\mathbf{R}}{dT^2} = -G'\frac{mM}{R^2}\frac{\mathbf{R}}{R} \qquad G' = \frac{G\tau_0^2}{l_0^3}$$

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

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

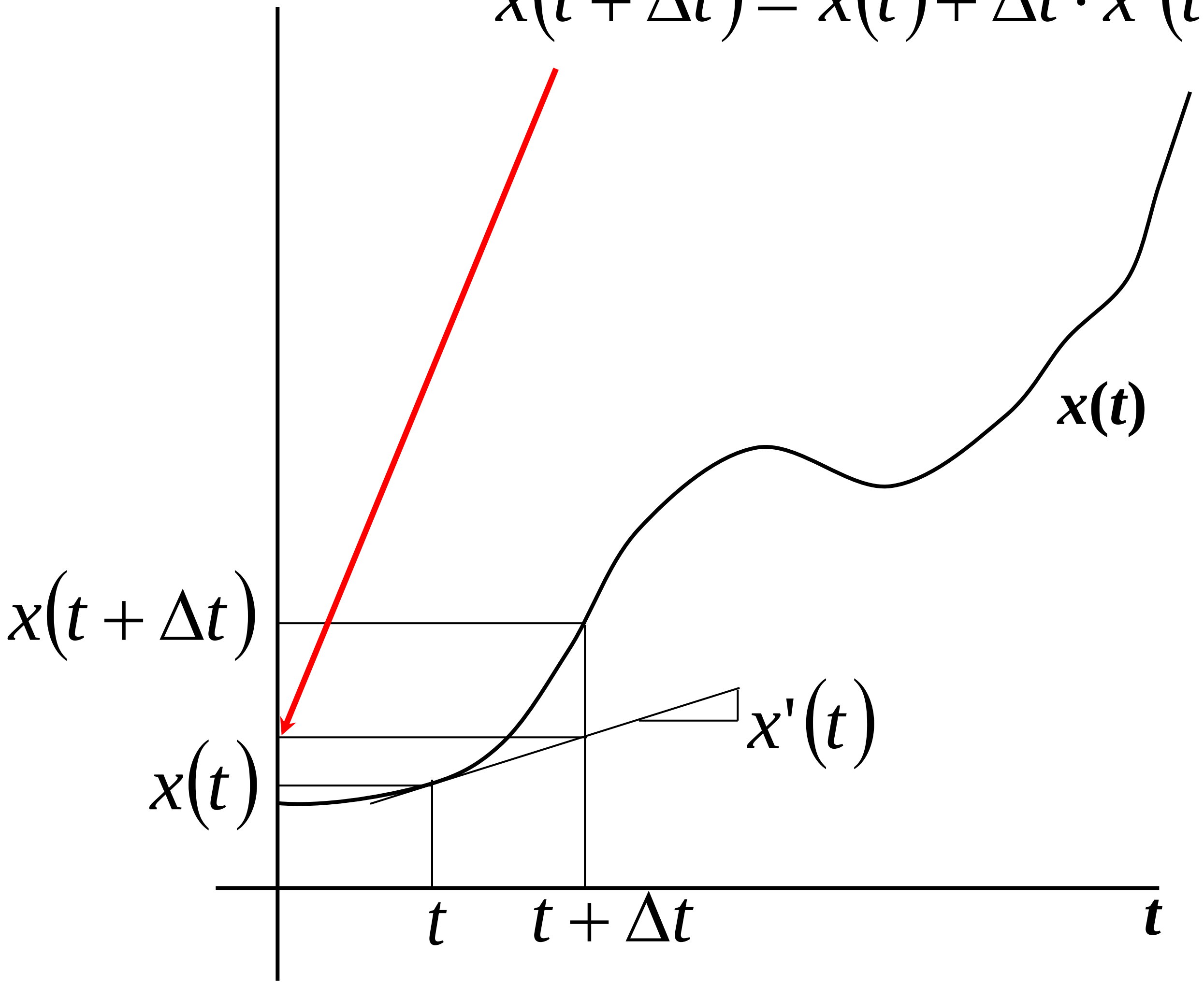
$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = f(t, x(t))$$

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

- **Accuracy not good**
- **Asymmetric with respect to *t*, *t*+Δ*t***

Illustrative image of Euler method

$$x(t+\Delta t) = x(t) + \Delta t \cdot x'(t)$$



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

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

- **Average the Euler formula at *t* and *t*+Δ*t***

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

Problem: *x*(*t*+Δ*t*) and *f*(*t*+Δ*t*, *x*(*t*+Δ*t*)) are unknown

=> Use *x*(*t*+Δ*t*) by Euler formula

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

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

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

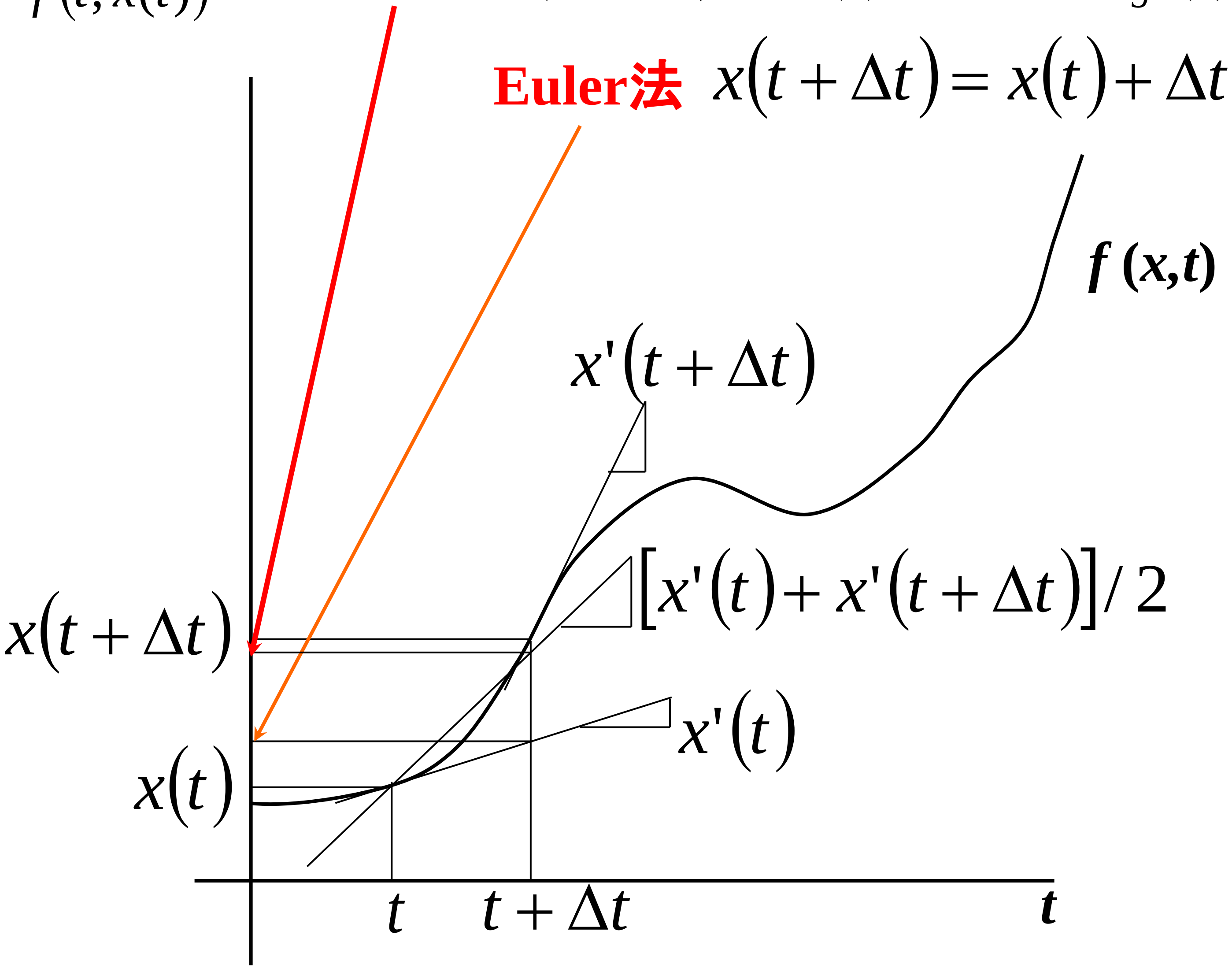
$$\boxed{x(t+\Delta t) = x(t) + \frac{k_0 + k_1}{2}}$$

Illustrative image of Heun method

$$\frac{dx}{dt} = x'(T) = f\left(t, x(t)\right)$$

Heun法 $x(t+\Delta t) = x(t) + \Delta t \cdot x'_{avg}(t)$

Euler法 $x(t+\Delta t) = x(t) + \Delta t \cdot x'(t)$



First-order differential equation

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

Euler formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + k_0}$$

Heun formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_0, t + \Delta t)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + \frac{k_0 + k_1}{2}}$$

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(diff1func, t0, x0, dt):
   k0 = dt * diff1func(t0, x0)
   x1 = x0 + k0
   return x1

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

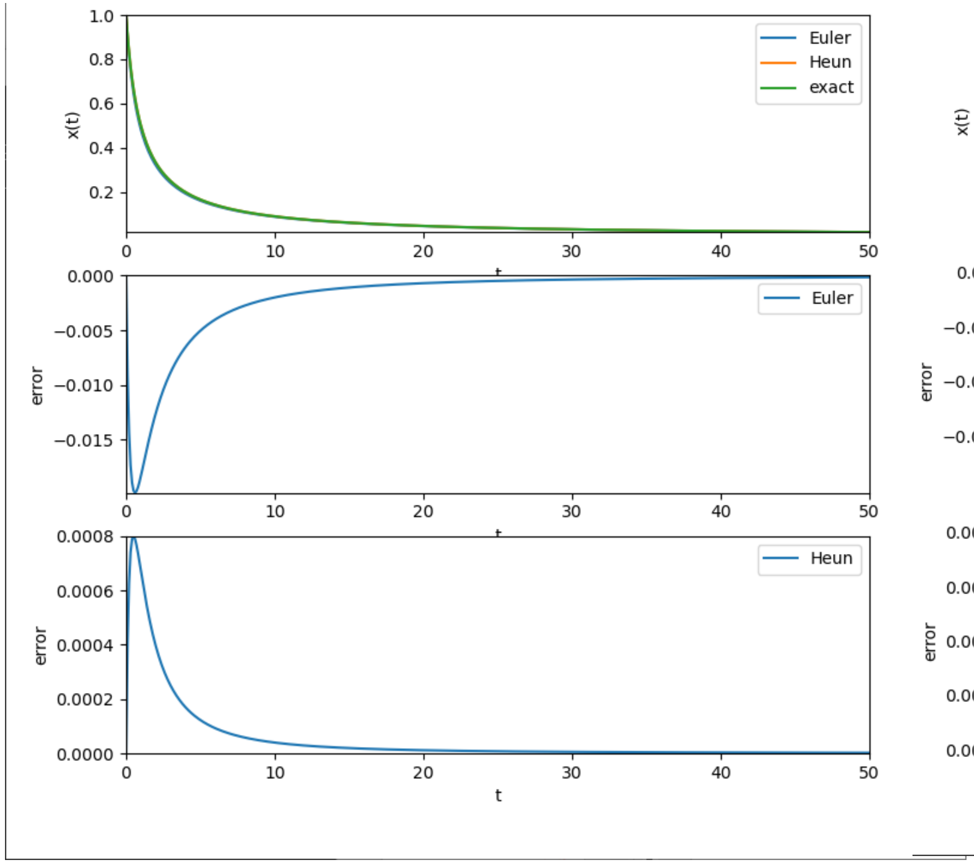
```
# Solve by the Heun formula
def diffeq_heun(diff1func, t0, x0, dt):
   k0 = dt * diff1func(t0, x0)
   k1 = dt * diff1func(t0+dt, x0+k0)
   x1 = x0 + (k0 + k1) / 2.0
   return x1

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

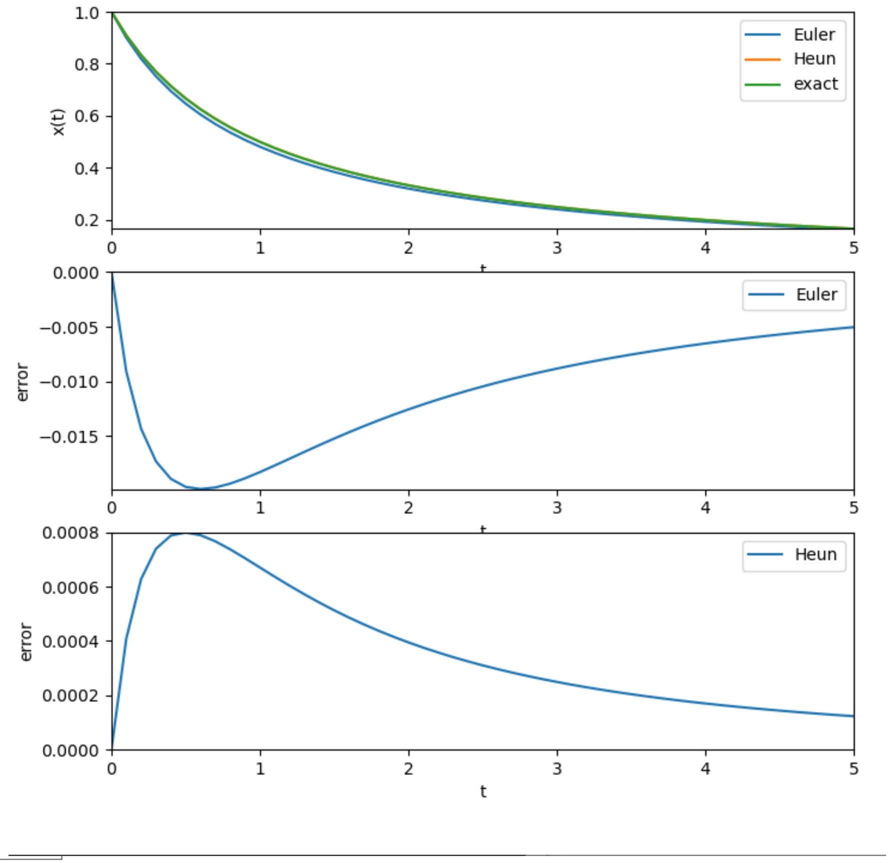
Program: Euler vs. Heun methods

Usage: python diffeq_euler_heun.py x0 dt nt iprint_interval

python diffeq_euler_heun.py

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





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

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

Solution of $\dfrac{df(x)}{dx} = g(x)$ **=>** $\dfrac{dx}{dt} = f(t,x)$

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

Problem: *x*(*t*+Δ*t*) and *x*(*t*+2Δ*t*) are unknown

=> Use *x*(*t*+Δ*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 Δ*t* to a half

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

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

=> Runge-Kutta formula

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

(ルンゲークッタ公式)

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

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

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

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

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

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

Determine μ_i and k_i so as to get minimum error

Number of k_i n => 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)

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

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

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

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

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:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$\boldsymbol{x(t+\Delta t) = x(t) + k_0}$$

Heun formula:

$$k_0 = \Delta t \cdot f(x(t), \qquad t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_0, t + \Delta t)$$
$$\boldsymbol{x(t+\Delta t) = x(t) + \frac{1}{2}(k_0 + k_1)}$$

Simson formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_0/2, \qquad t + \Delta t/2)$$
$$k_2 = \Delta t \cdot f(x(t) + (k_0 + k_1)/2, t + \Delta t)$$
$$\boldsymbol{x(t+\Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)}$$

3-stage 3-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_0/2, \qquad t + \Delta t/2)$$
$$k_2 = \Delta t \cdot f(x(t) + 2k_1 - k_0, t + \Delta t)$$
$$\boldsymbol{x(t+\Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)}$$

4-stage 4-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(x(t), t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_1/2, t + \Delta t/2)$$
$$k_2 = \Delta t \cdot f(x(t) + k_2/2, t + \Delta t/2)$$
$$k_3 = \Delta t \cdot f(x(t) + k_3, \quad t + \Delta t)$$
$$\boldsymbol{x(t+\Delta t) = x(t) + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3)}$$

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)

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

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

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

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

$$\boldsymbol{x(t + \Delta t) = x(t) + \Delta t \cdot v(t)}$$

Second-order diff. eq. : Heun formula

(二階微分方程式の解法: ホイン法)

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

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

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

$$(3)\ k_1 = \Delta t \cdot f(\boldsymbol{x(t) + k_0'}, \boldsymbol{v(t) + k_0}, t + \Delta t)$$

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

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)

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

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

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

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

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

- **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

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

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

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

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

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

- **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

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

includes the subtraction of
***x*(*t*) terms and may cause roundoff error.**

Converting the equation to

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

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

Can reduce the roundoff errors.
Note: Time calculated for *x*(*t*) and *v*(*t*) are shifted by Δt

Leap Flog vs. Verlet

Confirm the Leap Flog formula is identical to the Verlet formula

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

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

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

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

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

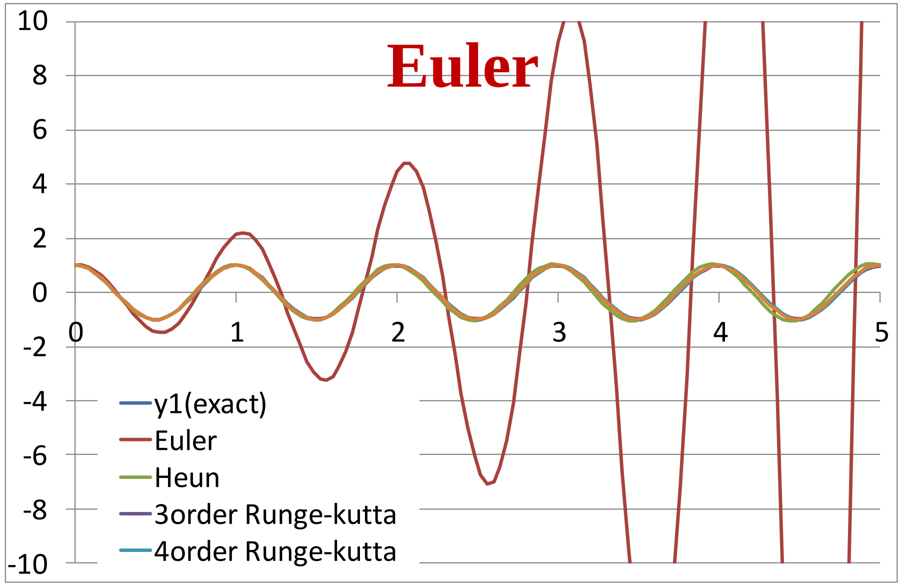
Accuracy of numerical solutions: Diff. eq.

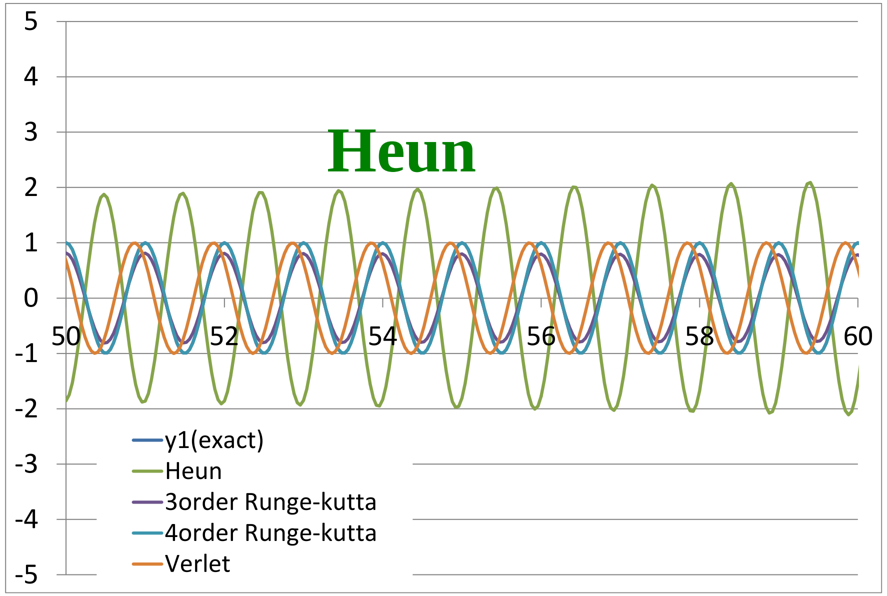
$$\frac{d^2x}{dt^2} = -4\pi^2 x \qquad \left(\frac{dx}{dt} = v, \ \frac{dv}{dt} = -4\pi^2 x \right)$$

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

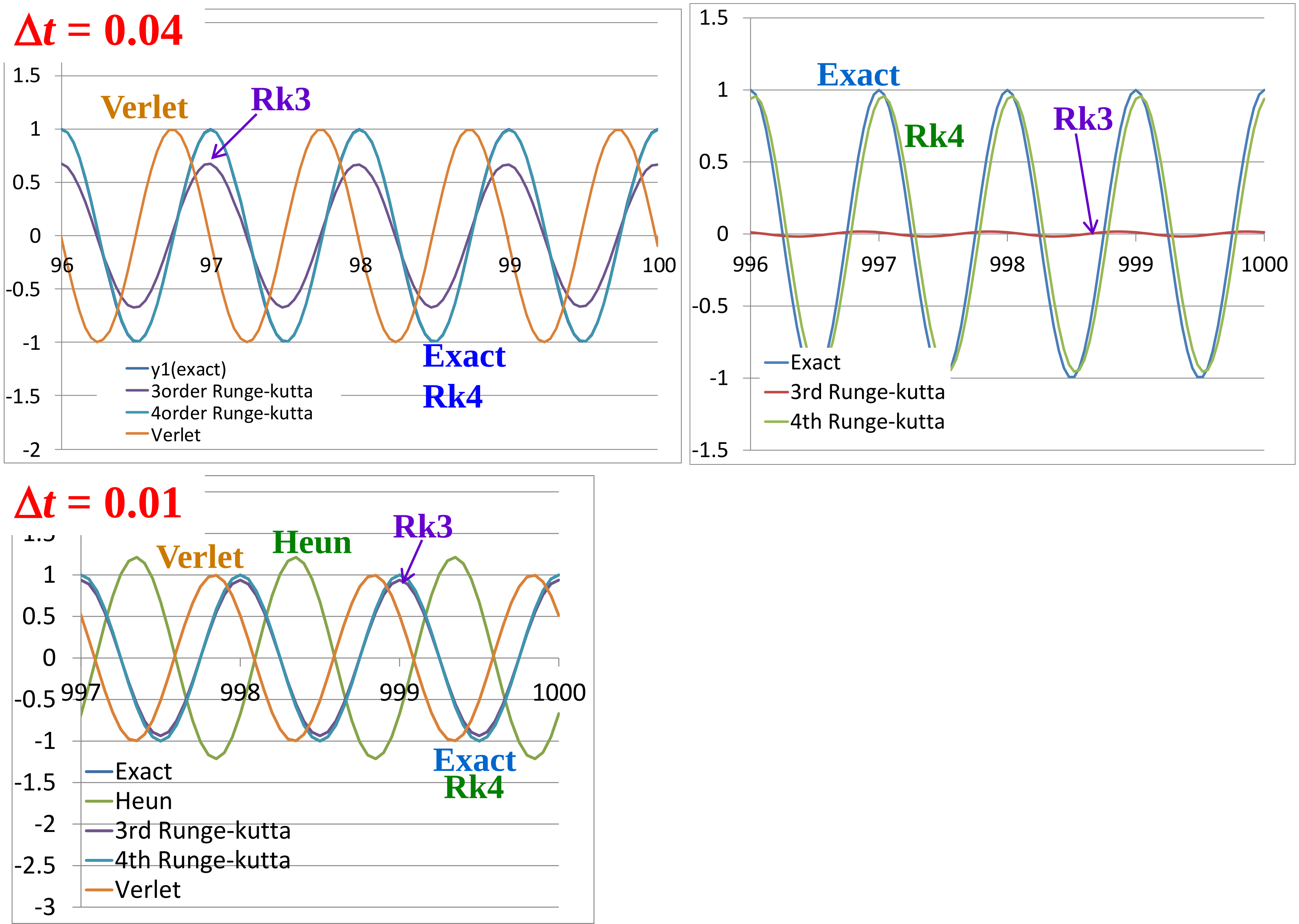
$$x = \cos(2\pi t) \quad v = -2\pi \sin(2\pi t)$$

Δt = 0.04



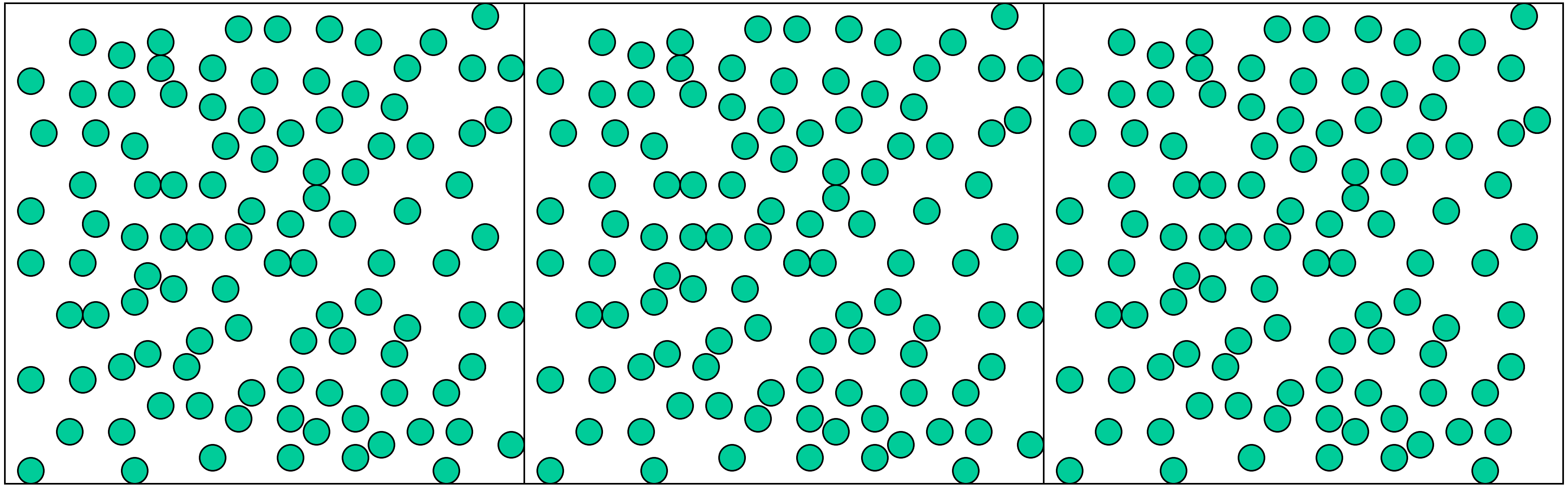


Accuracy of numerical solutions: Diff. eq.



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

$$\phi(r) = \infty \qquad r \le \sigma$$
$$= 0 \qquad r > \sigma$$

Lennard-Jones (LJ) potential

レナードー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

ボルンー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（MXDOrto/MXDTricl）

河村ポテンシャル

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

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

Morse potential

Empirical interatomic potential

$$U_{ij}(r_{ij}) = \frac{z_i z_j e^2}{4\pi\varepsilon_0}\frac{1}{r_{ij}} + f_0(b_i + b_j)\exp\left[\frac{a_i + a_j - r_{ij}}{b_i + b_j}\right] + \frac{c_i c_j}{r_{ij}{}^6}$$

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

$$U_i(\boldsymbol{r_i}, t) = \sum_j U_{ij}(\boldsymbol{r_j}(t) - \boldsymbol{r_i}(t)), \boldsymbol{F_i}(\boldsymbol{r_i}, t) = -\sum_j \frac{\partial}{\partial \boldsymbol{r_i}} U_{ij}(\boldsymbol{r_j}(t) - \boldsymbol{r_i}(t))$$

Most time-consuming term

Better to re-use previous steps,

$\boldsymbol{F_i}(\boldsymbol{r_i}, \boldsymbol{t} - \Delta \boldsymbol{t}), \boldsymbol{F_i}(\boldsymbol{r_i}, \boldsymbol{t} - 2\Delta \boldsymbol{t})$ **etc**

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

Requirements of algorisms 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 (予測子ー修正子法)

  Rahman predictor-corrector method
  (ラーマンの予測子ー修正子法)

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

Program: Planet simulation

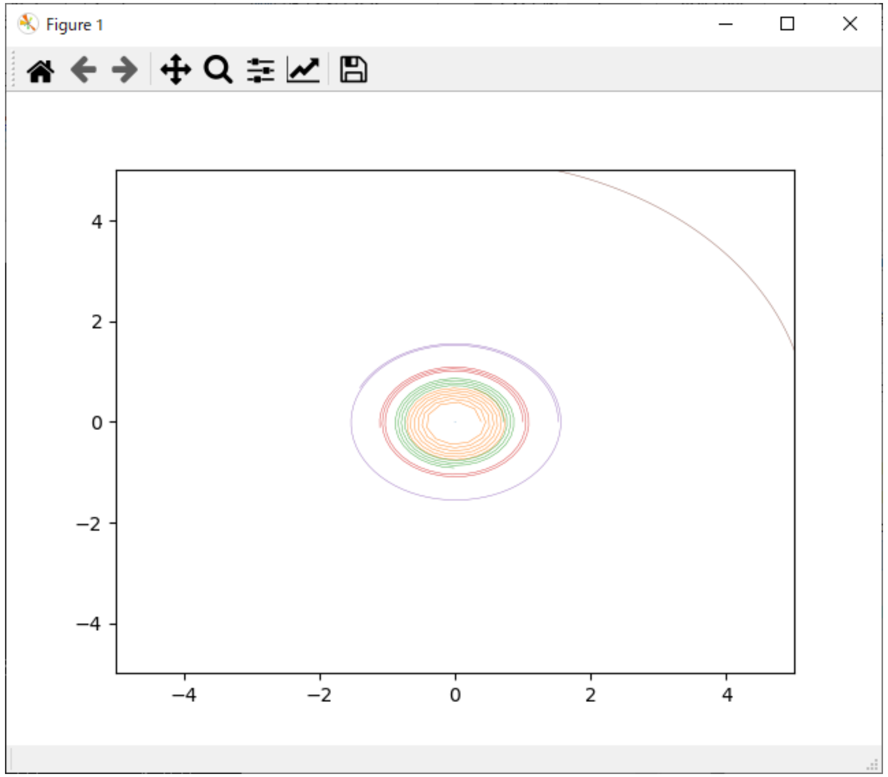
Usage: python diffeq2nd_planet.py solver dt nt

solver: ‘Euler’ or ‘Verlet’

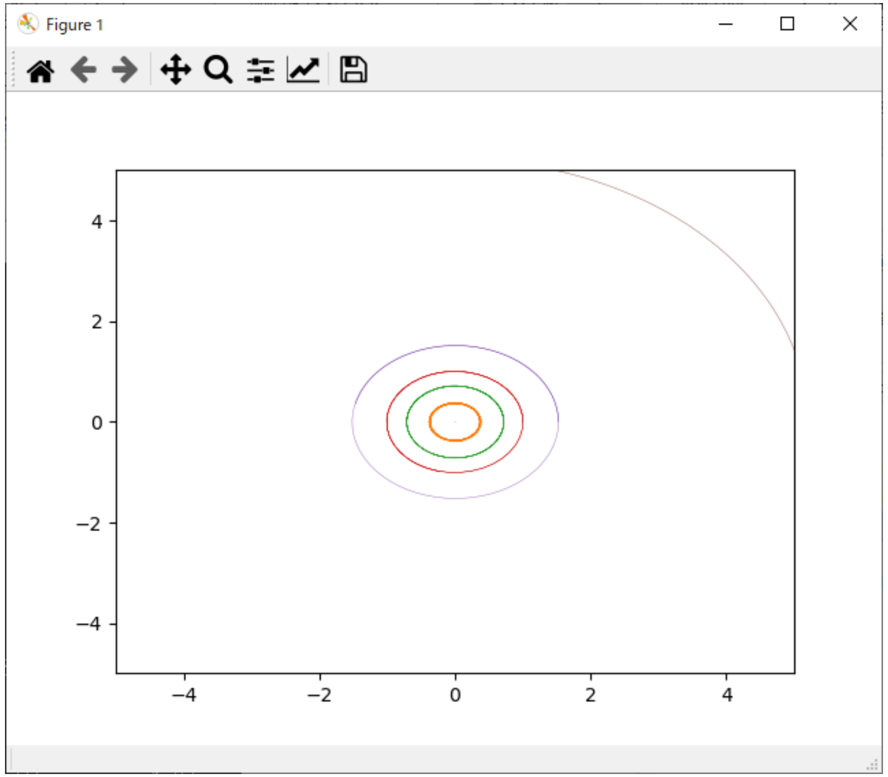
dt: time step in day (time is normalized by a day)

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