

# Computational Materials Science (計算材料学特論)

## Lecture materials updated

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

### 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 10, 11:46 Final version of lecture materials on June 10 have been updated ([20250610ComputerAndErrorSources.zip](#))

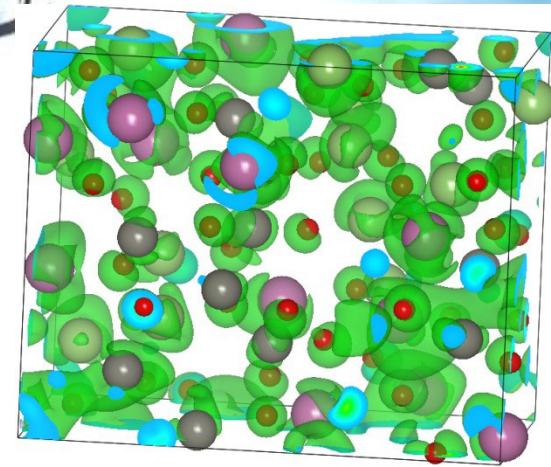
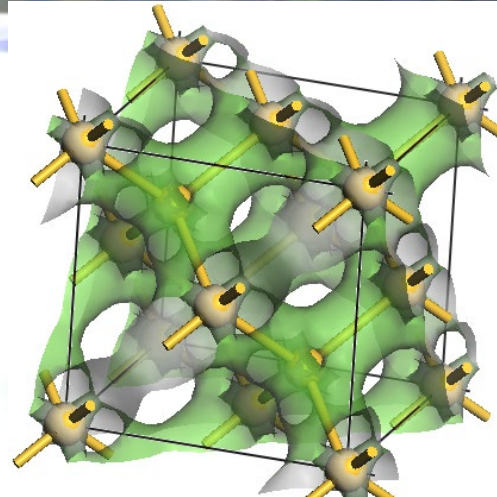
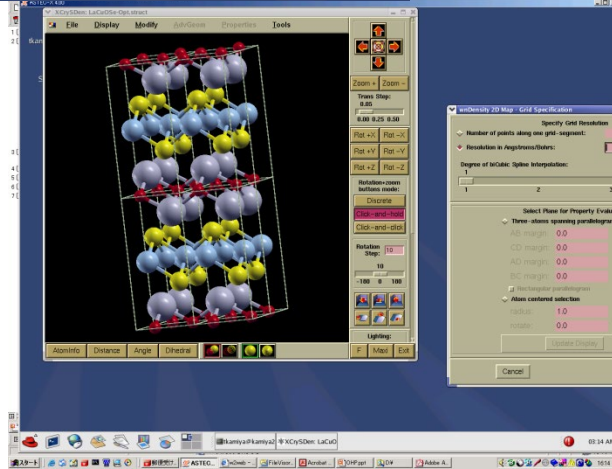
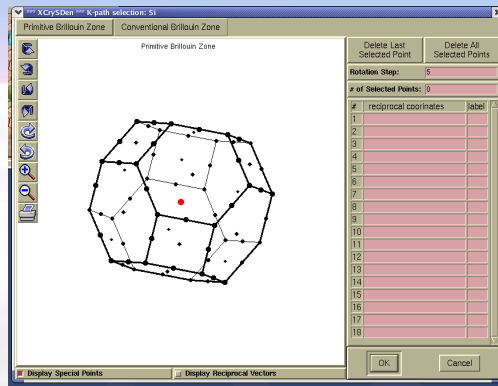
▶ 詳細履歴

**pythonプログラミングを始める前に (Getting Started with python)**

# Computational Materials Science

## 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](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),  
Differential equation (微分方程式))
- #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) Kamiya (Nonlinear optimization (非線形最適化),  
Fourier transformation (フーリエ変換))
- #06 June 27 (Fri) Kamiya, Matrix (行列)
- #07 July 1 (Tue) Kamiya, Review (復習)
- #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, June 01**

**課題解答の解説**

# PROBLEM, June 10

- **Submit electronic file(s) via LMS until the midnight of June 11**

(If LMS doesn't work, send the files to [kamiya.t.aa@m.titech.ac.jp](mailto:kamiya.t.aa@m.titech.ac.jp).

In this case, file name must include your STUDENT ID and FULL NAME)

**Choose one of the following PROBLEM 1 or PROBLEM 2**

## PROBLEM 1:

- (i) Convert  $101001_2$  to base 10
- (ii) Convert  $4251_{10}$  to base 16

## PROBLEM 2:

Choose one of the python programs given today (sum\_error.py, sum.py, base.py).

- Explain what each block of the source code does,

or

- list up the source code parts that you cannot understand what they do or why they are needed.

今日配布したプログラム (sum\_error-plt.py, sum.py, base.py) から1つを選び、

以下のいずれかを答えよ

- ソースコードのそれぞれの部分が何をしているかを説明する
- ソースコードの中で理解できない部分、あるいは なぜそれが必要かわからない部分を述べよ

# PROBLEM, June 10

Choose one of the following **PROBLEM 1** or **PROBLEM 2**

## PROBLEM 1:

(i) Convert  $110011_2$  to base 10

> python base.py 110011 2 10

Convert 110011 in base 2 to base 10

0-th digit = 1 <sub>2</sub> :	+ 1*2 <sup>0</sup> => +	1 <sub>10</sub> =>	1 <sub>10</sub>
1-th digit = 0 <sub>2</sub> :	+ 0*2 <sup>1</sup> => +	0 <sub>10</sub> =>	1 <sub>10</sub>
2-th digit = 0 <sub>2</sub> :	+ 0*2 <sup>2</sup> => +	0 <sub>10</sub> =>	1 <sub>10</sub>
3-th digit = 1 <sub>2</sub> :	+ 1*2 <sup>3</sup> => +	8 <sub>10</sub> =>	9 <sub>10</sub>
4-th digit = 0 <sub>2</sub> :	+ 0*2 <sup>4</sup> => +	0 <sub>10</sub> =>	9 <sub>10</sub>
5-th digit = 1 <sub>2</sub> :	+ 1*2 <sup>5</sup> => +	32 <sub>10</sub> =>	41 <sub>10</sub>



# PROBLEM, June 10

(ii) Convert  $4251_{10}$  to base 16

$$4251 = 265 * 16 + 11$$

$$265 = 16 * 16 + 9$$

$$16 = 1 * 16 + 0$$

$$1 = 0 * 16 + 1$$

$$\Rightarrow 109B_{16}$$

> python base.py 5323 10 16

Convert 4251 in base 10 to base 10

0-th digit = 1 <sub>10</sub> :	+ 1*10 <sup>0</sup> => +	1 <sub>10</sub> =>	1 <sub>10</sub>
1-th digit = 5 <sub>10</sub> :	+ 5*10 <sup>1</sup> => +	50 <sub>10</sub> =>	51 <sub>10</sub>
2-th digit = 2 <sub>10</sub> :	+ 2*10 <sup>2</sup> => +	200 <sub>10</sub> =>	251 <sub>10</sub>
3-th digit = 4 <sub>10</sub> :	+ 4*10 <sup>3</sup> => +	4000 <sub>10</sub> =>	4251 <sub>10</sub>

Convert 4251 in base 10 to base 16

$$4251 = 265 * 16 + 11: \text{base\_16} \Rightarrow B$$

$$265 = 16 * 16 + 9: \text{base\_16} \Rightarrow 9B$$

$$16 = 1 * 16 + 0: \text{base\_16} \Rightarrow 09B$$

$$1 = 0 * 16 + 1: \text{base\_16} \Rightarrow 109B$$

# PROBLEM, June 10

## Modern C++ code by ChatGPT4:

### Prompt:

make a C++ program to convert a base 10 value to **base 20** value.

- (i) receive the base 10 value from command line argument
- (ii) The **characters of the base 20 is given by a hash variable** and to convert an integer from 0 to 20 to **a character in "0123456789ABCDEFGHIJK"**
- (iii) convert the base 10 value to base 20

### Answer:

Here is a C++ program that meets your requirements. It receives a base 10 value from the command line, uses a hash variable to map integers to characters in the base 20 system ("0123456789ABCDEFGHIJK"), and converts the base 10 value to a base 20 value.cpp



# PROBLEM, June 10

## Code: base20.cpp

```
#include <iostream>
#include <unordered_map>
#include <string>
#include <cstdlib> // For std::atoi

std::string convertBase10ToBase20(int base10Value, const std::unordered_map<int, char>& base20Map) {
    std::string base20Value;

    while (base10Value > 0) {
        int remainder = base10Value % 20;
        base20Value = base20Map.at(remainder) + base20Value;
        base10Value /= 20;
    }
    return base20Value.empty() ? "0" : base20Value;
}
```

# PROBLEM, June 10

## Code: base20.cpp

```
int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <base10Value>" << std::endl;
        return 1;
    }

    int base10Value = std::atoi(argv[1]);

    std::unordered_map<int, char> base20Map = {
        {0, '0'}, {1, '1'}, {2, '2'}, {3, '3'}, {4, '4'},
        {5, '5'}, {6, '6'}, {7, '7'}, {8, '8'}, {9, '9'},
        {10, 'A'}, {11, 'B'}, {12, 'C'}, {13, 'D'}, {14, 'E'},
        {15, 'F'}, {16, 'G'}, {17, 'H'}, {18, 'I'}, {19, 'J'},
        {20, 'K'}
    };
    // hash (dictionally in python)

    std::string base20Value = convertBase10ToBase20(base10Value, base20Map);

    std::cout << "Base 10 value: " << base10Value << std::endl;
    std::cout << "Base 20 value: " << base20Value << std::endl;

    return 0;
}
```

# C++ compilers

**Windows, macOS, Linux: Intel oneAPI compiler**

**Linux, WSL2, macOS: gcc**

**Windows: Visual Studio**

**mingw**

**Cygwin/gcc**

**NOTE: C++ is a steadily growing (i.e., getting *fat*) language,  
seriously depends on C++ compilers.**

**Check which C++ version and specifications are supported by the C++  
compiler.**

# PROBLEM, June 13

- **Submit electronic file(s) via LSM until June 15**

(If LSM doesn't work, send the files to [kamiya.t.aa@m.titech.ac.jp](mailto: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  $\hbar$ .**

# Effective mass

## LCAO band

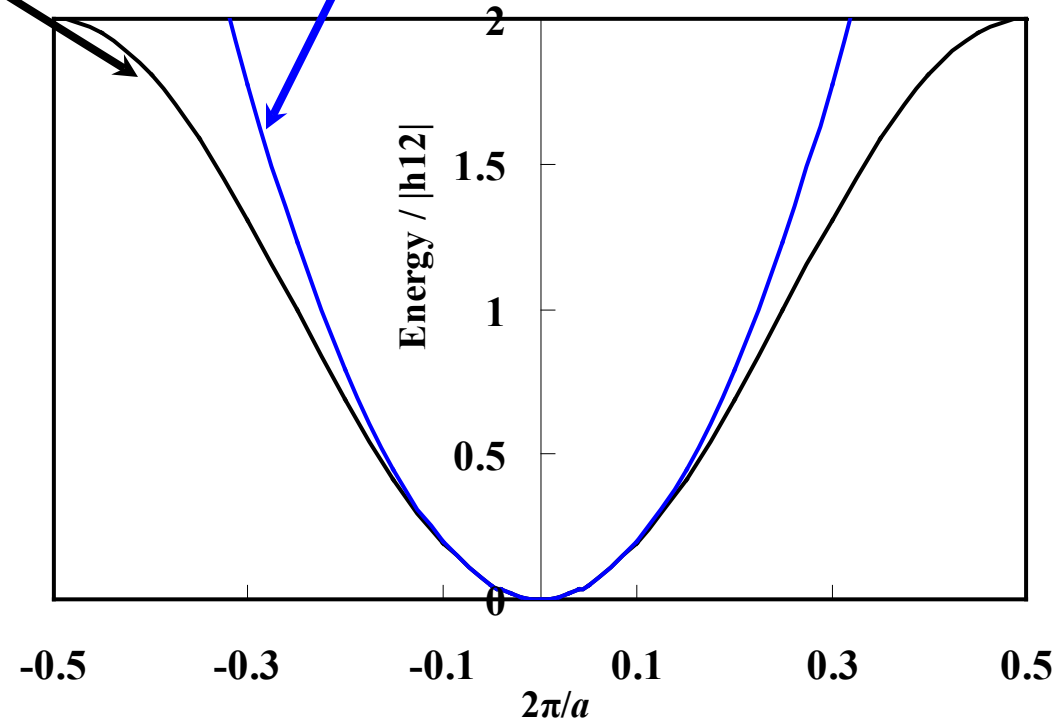
$$E(k) = \varepsilon_1 - 2|h_{12}|\cos(ka) \sim \varepsilon_1 - 2|h_{12}| + |h_{12}|a^2k^2 + O((ka)^4)$$

## Free electron model

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

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

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



# Effective mass

$k$  represents fractional coordinate in reciprocal unit cell:  
generally expressed in the range  $[-1/2 \ 1/2]$

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

Note  $E(k)$  is in eV

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

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

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

# Numerical differentiation

数值微分



# Fundamental of numerical analysis:

## Differential => Difference (差分法)

Differential (微分)  $\frac{dy}{dx}$  => approximated by difference (差分)  $\frac{\Delta y}{\Delta x}$

The following terms will often appear.

Difference (差分) :  $\Delta x = x_i - x_j, \Delta y = y_i - y_j$

Divided difference (差分商) :  $\frac{\Delta y}{\Delta x}$

Forward difference (前進差分):  $\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \quad (x_i < x_{i+1})$

Backward difference (後退差分):  $\frac{y_i - y_{i-1}}{x_i - x_{i-1}} \quad (x_{i-1} < x_i)$

Central difference (中心差分):  $\frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} \quad (x_{i+1} - x_i = x_i - x_{i-1} = h > 0)$

# Numerical differentiation (数値微分)

To calculate  $\frac{dy}{dx}$  by computer,

replace the differential ' $d$ ' with finite difference ' $\Delta$ ' (for small  $\Delta x$ )

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{(x+h) - x} \sim \frac{\Delta f(x)}{\Delta x} = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$$

Accuracy can be improved by decreasing  $h$

⇔ But limited by the error of cancellation of significant digits  
at least  $h > 0.01\nu$  ( $\nu$ : a representative value to be handled)

32bit floating point (~7 digits) :  $h > 10^{-5}\nu$  (should be much larger)

64bit floating point (~16 digits):  $h > 10^{-14}\nu$  (should be much larger)

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

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

**Error  $\propto h^1$**   
(Difference error,  
差分誤差)

# Numerical differentiation: Effect of $h$

$$f(x) = x^3 \quad df/dx = 3x^2$$

		h=	1	0.1	0.01	0.001	1.00E-06
x	f(x)	df(x)/dx	$\Delta f(x)/\Delta x$				
0	0	0	1	0.01	0.0001	0.000001	1E-12
0.1	0.001	0.03	1.33	0.07	0.0331	0.030301	0.0300003
0.2	0.008	0.12	1.72	0.19	0.1261	0.120601	0.1200006
0.3	0.027	0.27	2.17	0.37	0.2791	0.270901	0.2700009
0.4	0.064	0.48	2.68	0.61	0.4921	0.481201	0.4800012
0.5	0.125	0.75	3.25	0.91	0.7651	0.751501	0.7500015
0.6	0.216	1.08	3.88	1.27	1.0981	1.081801	1.0800018
0.7	0.343	1.47	4.57	1.69	1.4911	1.472101	1.4700021
0.8	0.512	1.92	5.32	2.17	1.9441	1.922401	1.9200024
0.9	0.729	2.43	6.13	2.71	2.4571	2.432701	2.4300027
1	1	3	7	3.31	3.0301	3.003001	3.000003
1.1	1.331	3.63	7.93	3.97	3.6631	3.633301	3.6300033
1.2	1.728	4.32	8.92	4.69	4.3561	4.323601	4.3200036
1.3	2.197	5.07	9.97	5.47	5.1091	5.073901	5.070003899
1.4	2.744	5.88	11.08	6.31	5.9221	5.884201	5.8800042
1.5	3.375	6.75	12.25	7.21	6.7951	6.754501	6.750004499
1.6	4.096	7.68	13.48	8.17	7.7281	7.684801	7.680004799
1.7	4.913	8.67	14.77	9.19	8.7211	8.675101	8.6700051
1.8	5.832	9.72	16.12	10.27	9.7741	9.725401	9.720005399
1.9	6.859	10.83	17.53	11.41	10.8871	10.8357	10.8300057
2	8	12	19	12.61	12.0601	12.006	12.000006

# How to improve accuracy?: Average

$$\frac{df(x)}{dx} \sim \frac{f(x+h) - f(x)}{h} \quad \text{Asymmetric equation with respect to 'x'}$$

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

1st order error ( $h$ に関して一次の誤差)

Average => Symmetric formula: Three-point formula (3点公式, 中点則)

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

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

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

$$\text{Error: } \frac{f(x+h) - f(x-h)}{2h} = \frac{df(x)}{dx} + \frac{1}{3!} \frac{d^3f(x)}{dx^3} \underline{h^2} + O(h^3) \quad \text{2nd order error} \\ \propto h^2 \quad (\text{二次の誤差})$$

# How to improve accuracy?: Take average

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

**Asymmetric equations with respect to 'x'**

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

**Symmetric equation, better**

$$f(x) = x^3 \quad df(x)/dx = 3x^2$$

		h=	1	1	0.01	0.01
x	f(x)	df(x)/dx	(f(x+h)-f(x))/h	(f(x+h)-f(x-h))/(2h)	(f(x+h)-f(x))/h	(f(x+h)-f(x-h))/(2h)
0	0	0	1	1	0.0001	0.0001
0.2	0.008	0.12	1.72	1.12	0.1261	0.1201
0.4	0.064	0.48	2.68	1.48	0.4921	0.4801
0.6	0.216	1.08	3.88	2.08	1.0981	1.0801
0.8	0.512	1.92	5.32	2.92	1.9441	1.9201
1	1	3	7	4	3.0301	3.0001
1.2	1.728	4.32	8.92	5.32	4.3561	4.3201
1.4	2.744	5.88	11.08	6.88	5.9221	5.8801
1.6	4.096	7.68	13.48	8.68	7.7281	7.6801
1.8	5.832	9.72	16.12	10.72	9.7741	9.7201
2	8	12	19	13	12.0601	12.0001

# Higher order formula

## Three-point formula (3点公式)

$$\begin{aligned} f'(a) &= \frac{1}{h} \left\{ \frac{1}{2} f(a+h) - \frac{1}{2} f(a-h) \right\} \\ &+ \frac{1}{6} f^{(3)}(\underline{a}) h^2 + \dots \end{aligned}$$

## Five-point formula (5点公式)

$$\begin{aligned} f'(a) &= \frac{1}{h} \left\{ -\frac{1}{12} f(a+2h) + \frac{2}{3} f(a+h) - \frac{2}{3} f(a-h) + \frac{1}{12} f(a-2h) \right\} \\ &+ \frac{1}{30} f^{(5)}(\underline{a}) h^4 + \dots \end{aligned}$$

## Seven-point formula (7点公式)

$$\begin{aligned} f'(a) &= \frac{1}{h} \left\{ \frac{1}{60} f(a+3h) - \frac{3}{20} f(a+2h) + \frac{3}{4} f(a+h) - \frac{3}{4} f(a-h) \right. \\ &+ \left. \frac{3}{20} f(a-2h) - \frac{1}{60} f(a-3h) \right\} \\ &+ \frac{1}{140} f^{(7)}(\underline{a}) h^6 + \dots \end{aligned}$$

# Numerical error

$$\left. \frac{d}{dx} \exp(x) \right|_{x=1}$$

**Analytic solution** (解析解):

$$\exp(1.0) = 2.71828182845905$$

$N_{\text{div}}$	$h$	2-point	3-point	5-point	7-point
1	0.5	8.09E-01	1.15E-01	-5.83E-03	3.18E-04
2	0.25	3.70E-01	2.84E-02	-3.57E-04	4.80E-06
3	0.125	1.77E-01	7.08E-03	-2.22E-05	7.43E-08
4	0.0625	8.67E-02	1.77E-03	-1.38E-06	1.16E-09
5	0.03125	4.29E-02	4.42E-04	-8.64E-08	1.81E-11
6	0.015625	2.13E-02	1.11E-04	-5.40E-09	2.64E-13
7	0.007813	1.06E-02	2.77E-05	-3.38E-10	4.44E-15
8	0.003906	5.32E-03	6.91E-06	-2.11E-11	-7.90E-14
9	0.001953	2.66E-03	1.73E-06	-1.37E-12	-3.51E-14
10	0.000977	1.33E-03	4.32E-07	-1.23E-13	-3.65E-13
11	0.000488	6.64E-04	1.08E-07	-8.42E-13	-5.70E-13
12	0.000244	3.32E-04	2.70E-08	-2.36E-13	7.04E-13
13	0.000122	1.66E-04	6.75E-09	1.28E-12	5.52E-13
14	6.1E-05	8.30E-05	1.69E-09	-2.36E-13	-1.93E-12
15	3.05E-05	4.15E-05	4.19E-10	-5.09E-12	-1.69E-12
16	1.53E-05	2.07E-05	1.06E-10	-7.51E-12	1.63E-11
17	7.63E-06	1.04E-05	1.92E-11	-1.48E-11	3.64E-12
18	3.81E-06	5.18E-06	-9.94E-12	-4.87E-11	-9.94E-12
19	1.91E-06	2.59E-06	-9.94E-12	-2.93E-11	-2.18E-12



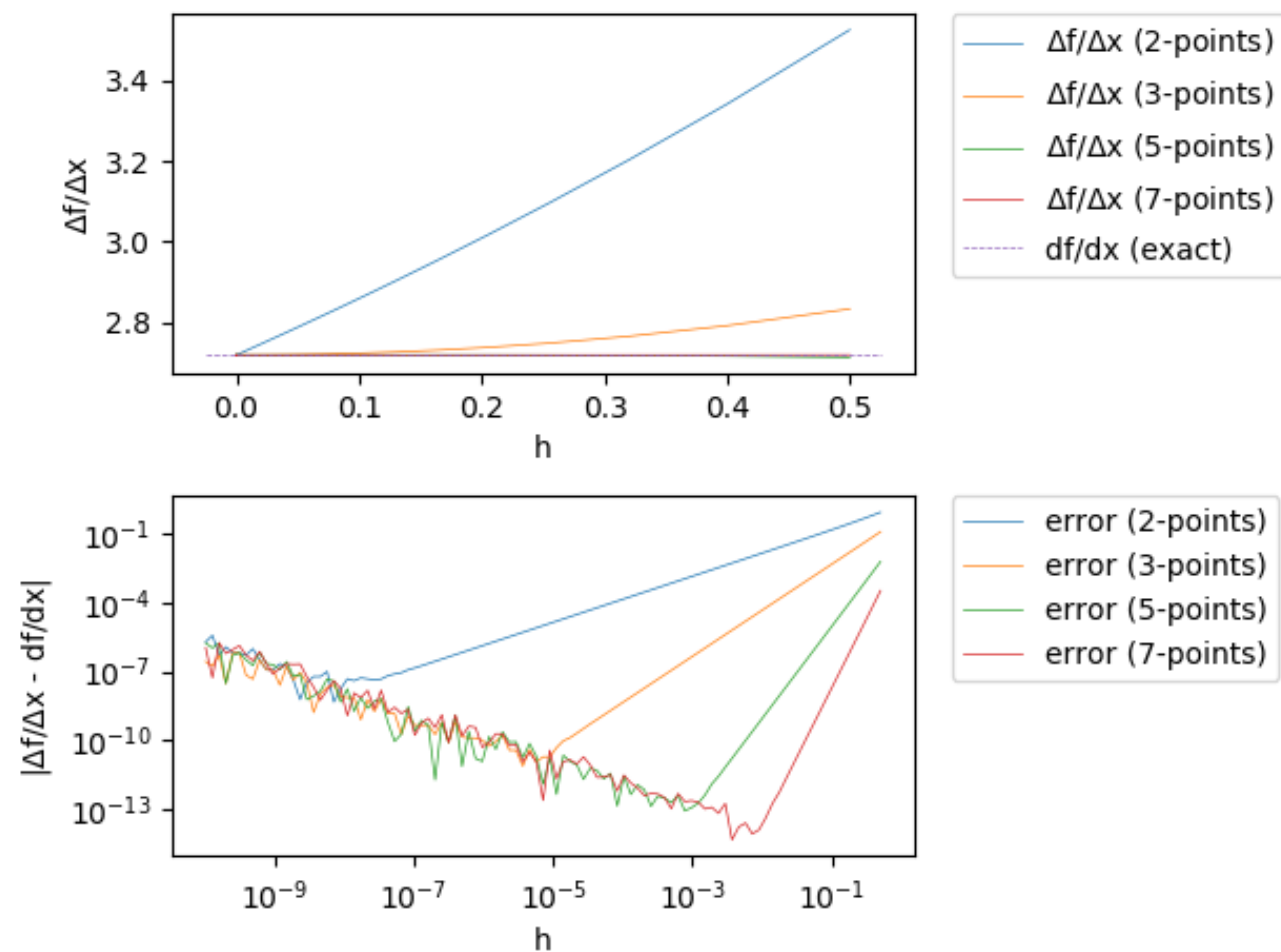
# Program: diff\_order.py

$$\left. \frac{d}{dx} \exp(x) \right|_{x=1}$$

**Analytic solution** (解析解):

$$\left. \frac{d}{dx} \exp(x) \right|_{x=1} = \exp(1.0) = 2.71828182845905$$

run: `python diff_order.py`



# Richardson extrapolation differentiation

(リチャードソン補外)

森正武, FORTRAN 77 数値計算プログラミング、岩波書店 (1987年増補版)

- Start from the three-point formula (中点則), and then iteratively repeat the following formula that updates the calculation precision until a required precision will be satisfied.

(中点則から出発し、高次の微分に相当する公式を自動的に適用し、要求精度を満たすまで繰り返す)

1. Calc by three-point formula  $D_0^{(0)} = (f(x+h) - f(x-h)) / (2h)$  at the  $x$  mesh  $h = h_0$ .
2. Reduce the mesh to a half  $h_k = (1/2)^k h$ , and then calculate  $D_0^{(k)}$  by the three-point formula.
3. Calculate next quantity

$$D_m^{(k)} = \frac{4^m D_{m-1}^{(k+1)} - D_{m-1}^{(k)}}{4^m - 1}$$

4. Iteration will be terminated if  $|D_m^{(0)} - D_{m-1}^{(0)}|$  becomes smaller than the required precision.

# Numerical error

$$\left. \frac{d}{dx} \exp(x) \right|_{x=1}$$

**Analytic solution** (解析解):

$$\exp(1) = 2.71828182845905$$

$N_{\text{div}}$	$h$	2-point	3-point	5-point	7-point	Richardson extrapolation		
1	0.5	8.09E-01	1.15E-01	-5.83E-03	3.18E-04			
2	0.25	3.70E-01	2.84E-02	-3.57E-04	4.80E-06	-3.57E-04		
3	0.125	1.77E-01	7.08E-03	-2.22E-05	7.43E-08			
4	0.0625	8.67E-02	1.77E-03	-1.38E-06	1.16E-09	2.06E-09		
5	0.03125	4.29E-02	4.42E-04	-8.64E-08	1.81E-11			
6	0.015625	2.13E-02	1.11E-04	-5.40E-09	2.64E-13			
7	0.007813	1.06E-02	2.77E-05	-3.38E-10	4.44E-15			
8	0.003906	5.32E-03	6.91E-06	-2.11E-11	-7.90E-14	-1.38E-14		
9	0.001953	2.66E-03	1.73E-06	-1.37E-12	-3.51E-14			
10	0.000977	1.33E-03	4.32E-07	-1.23E-13	-3.65E-13			
11	0.000488	6.64E-04	1.08E-07	-8.42E-13	-5.70E-13			
12	0.000244	3.32E-04	2.70E-08	-2.36E-13	7.04E-13			
13	0.000122	1.66E-04	6.75E-09	1.28E-12	5.52E-13			
14	6.1E-05	8.30E-05	1.69E-09	-2.36E-13	-1.93E-12			
15	3.05E-05	4.15E-05	4.19E-10	-5.09E-12	-1.69E-12			
16	1.53E-05	2.07E-05	1.06E-10	-7.51E-12	1.63E-11	-3.11E-15		
17	7.63E-06	1.04E-05	1.92E-11	-1.48E-11	3.64E-12			
18	3.81E-06	5.18E-06	-9.94E-12	-4.87E-11	-9.94E-12	4.52E-13		
19	1.91E-06	2.59E-06	-9.94E-12	-2.93E-11	-2.18E-12	1.69E-12		

**For non-constant  $h_i = x_{i+1} - x_i$**

$x$	$y$
$x_{-1}$	$y_{-1}$
$x_0$	$y_0$
$x_1$	$y_1$

**Rough method: Take average**

**(maybe good but not best)**

$$y'(x_0) = \frac{1}{2} \left[ \frac{y_1 - y_0}{x_1 - x_0} + \frac{y_0 - y_{-1}}{x_0 - x_{-1}} \right]$$

**Polynomial method: Lagrange polynomial** (ラングランジュ多項式)

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

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

$$y(x) = y_{-1} \frac{(x - x_0)(x - x_1)}{(x_{-1} - x_0)(x_{-1} - x_1)} + y_0 \frac{(x - x_{-1})(x - x_1)}{(x_0 - x_{-1})(x_0 - x_1)} + y_1 \frac{(x - x_{-1})(x - x_0)}{(x_1 - x_{-1})(x_1 - x_0)}$$
$$y'(x) = y_{-1} \frac{2x - (x_0 + x_1)}{(x_{-1} - x_0)(x_{-1} - x_1)} + y_0 \frac{2x - (x_{-1} + x_1)}{(x_0 - x_{-1})(x_0 - x_1)} + y_1 \frac{2x - (x_{-1} + x_0)}{(x_1 - x_{-1})(x_1 - x_0)}$$

# Second differential (二階微分)

If calculate 2<sup>nd</sup> differential using forward differences both for the 1<sup>st</sup> and the 2<sup>nd</sup> differentials ...

(一階微分を前進差分で計算してから二階微分を前進差分で計算すると・・・)

$$\begin{aligned}\frac{d^2 f(t)}{dx^2} &= \frac{\frac{df}{dx}(x + \Delta x) - \frac{df}{dx}(x)}{\Delta x} \\ &\sim \frac{\frac{f(x+2\Delta x) - f(x+\Delta x)}{\Delta x} - \frac{f(x+\Delta x) - f(x)}{\Delta x}}{\Delta x} = \frac{f(x+2\Delta x) - 2f(x+\Delta x) + f(x)}{\Delta x^2}\end{aligned}\quad (1)$$

If use backward differentials only for the 1<sup>st</sup> differentials  
(but logically inconsistent):

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

Symmetric formula w.r.t.  $x + \Delta x$  and  $x - \Delta x$  is obtained

( $x + \Delta x, x - \Delta x$  について対称になる式が取れ、精度が上がる)

Note:  $x$  value of eq. (1) is shifted by one  $\Delta x$  from eq. (2)

(eq.(1)では、横軸が $\Delta x$ ひとつ分ずれているために精度が落ちる)

# Second differential by central differences

If  $\frac{d^2 f(x)}{dx^2}$  is approximated by the central difference of  $\frac{df}{dx}$ :

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

$$\frac{d^2 f(t)}{dx^2} = \frac{f(x+\Delta x') - 2f(x) + f(x-\Delta x')}{\Delta x'^2}$$

**Symmetric formula w.r.t.  $x + \Delta x$  and  $x - \Delta x$  is obtained again**

(前進差分を使ったときと同様、

$x + \Delta x, x - \Delta x$  について対称になる式が取れ、精度が上がる)

# **Numeral integration (quadrature)**

**数值積分 (求積)**



# Numerical integration (数値積分)

How to calculate  $F(x) = \int_{x_0}^x g(x')dx'$  by computer

**Replace integral with summation of small mesh area**  
(積分を和で置き換える)

$$\int_{x_0}^x g(x')dx' = \sum_{i=0}^{x_i=x} g(x_i)h$$

**Derivation from difference approximation (差分式からの導出):**

$$\frac{df(x)}{dx} \sim \frac{f(x+h) - f(x)}{h} \quad \Rightarrow \quad g(x) \sim \frac{F(x+h) - F(x)}{h}$$

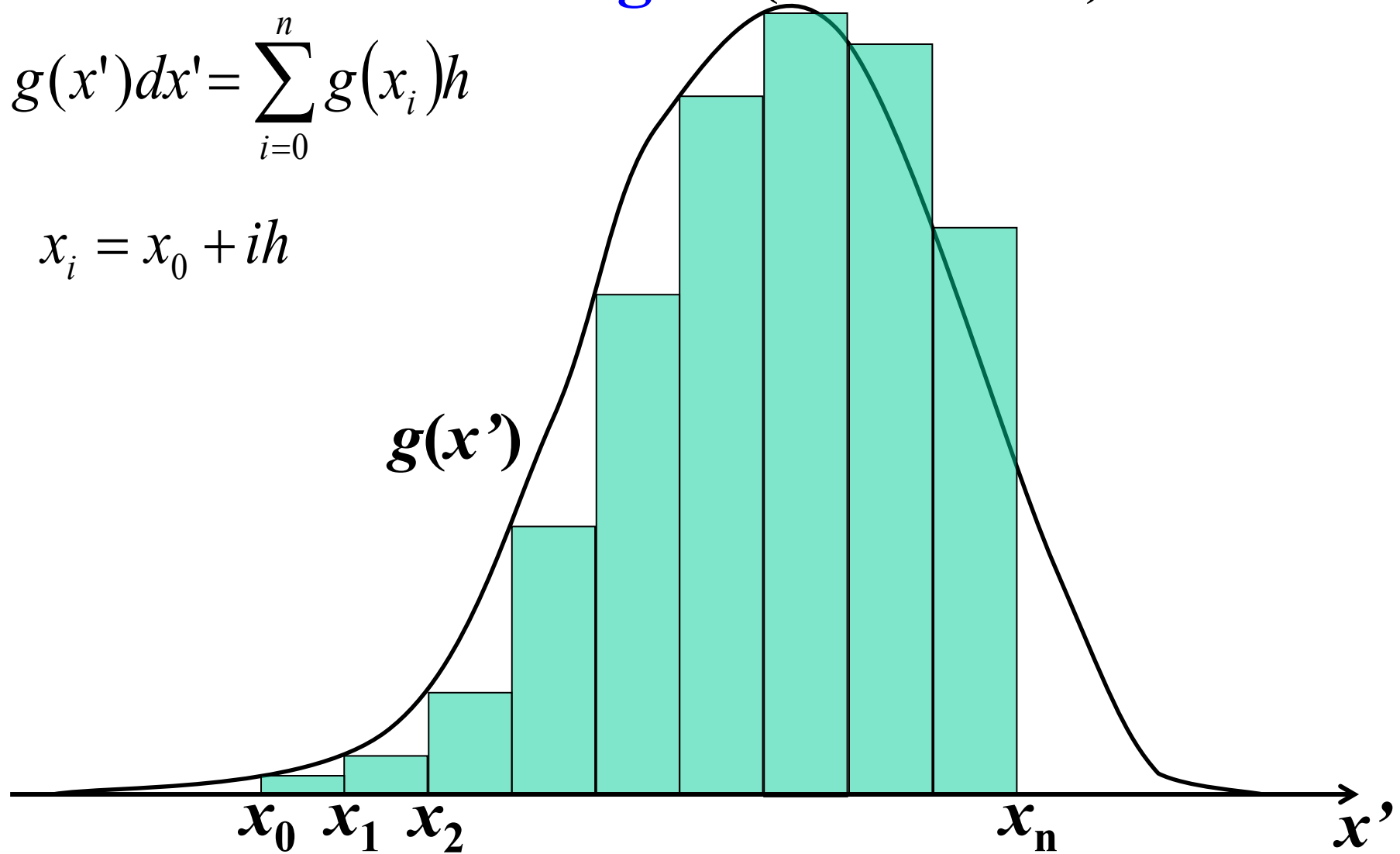
$$F(x+h) = F(x) + g(x)h = F(x-h) + [g(x) + g(x-h)]h$$

$$= \sum_{i=0}^{x_i=x} g(x_i)h$$

# Rieman integral (Rieman積分)

$$\int_{x_0}^x g(x') dx' = \sum_{i=0}^n g(x_i) h$$

$$x_i = x_0 + ih$$



**Asymmetric formula:**

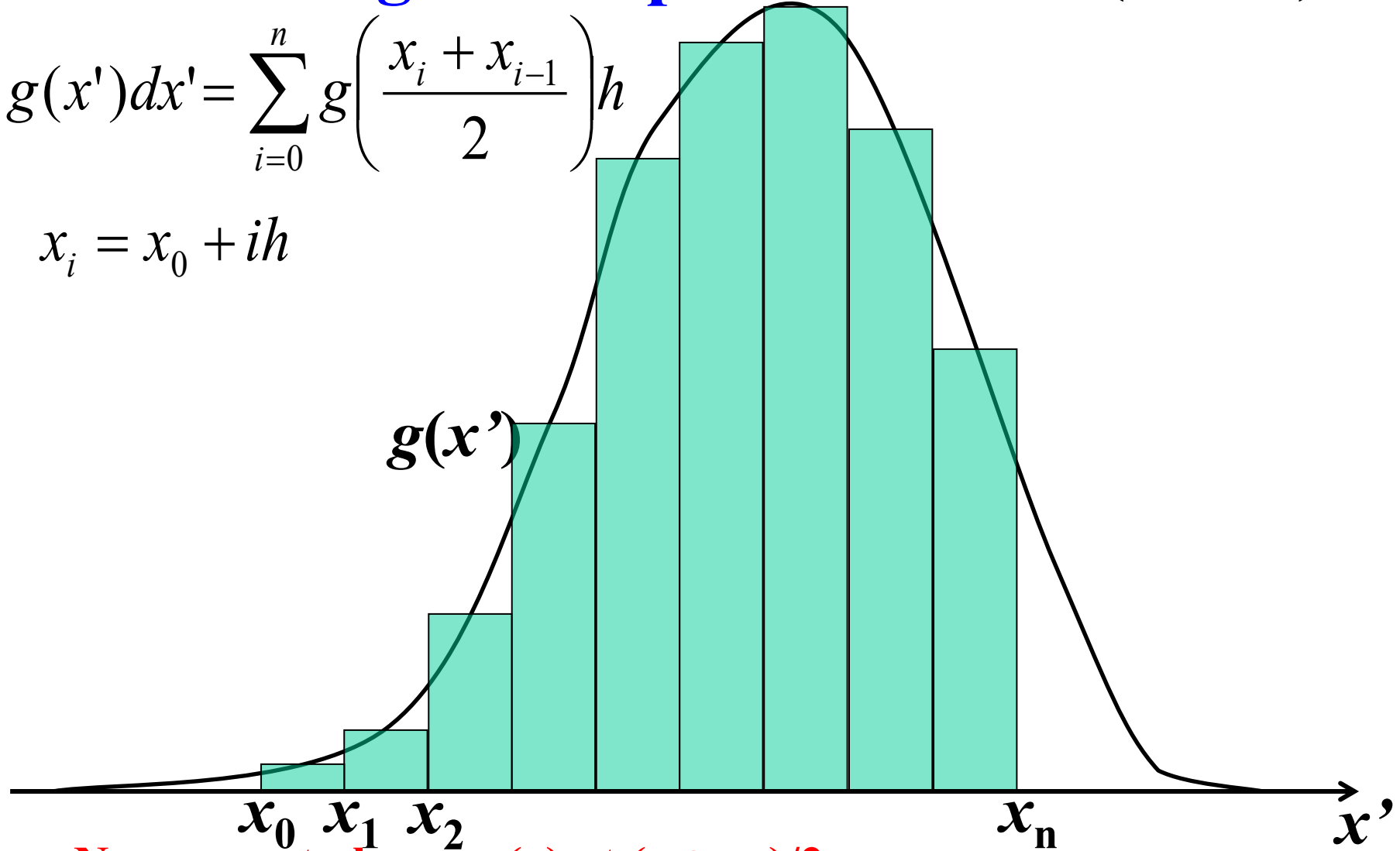
**monotone increasing  $g(x)$  => Underestimation (過小評価)**

**monotone decreasing  $g(x)$  => Overestimation (過大評価)**

# Take average: Mid-point formula (中点則)

$$\int_{x_0}^x g(x') dx' = \sum_{i=0}^n g\left(\frac{x_i + x_{i-1}}{2}\right) h$$

$$x_i = x_0 + ih$$



**Necessary to know  $g(x)$  at  $(x_i + x_{i-1})/2$ .**

**=> Unavailable for  $g(x)$  given only by numerical data**

( $g(x)$ が数値データで与えられている場合は使えない)

# Trapezoid formula (台形公式)

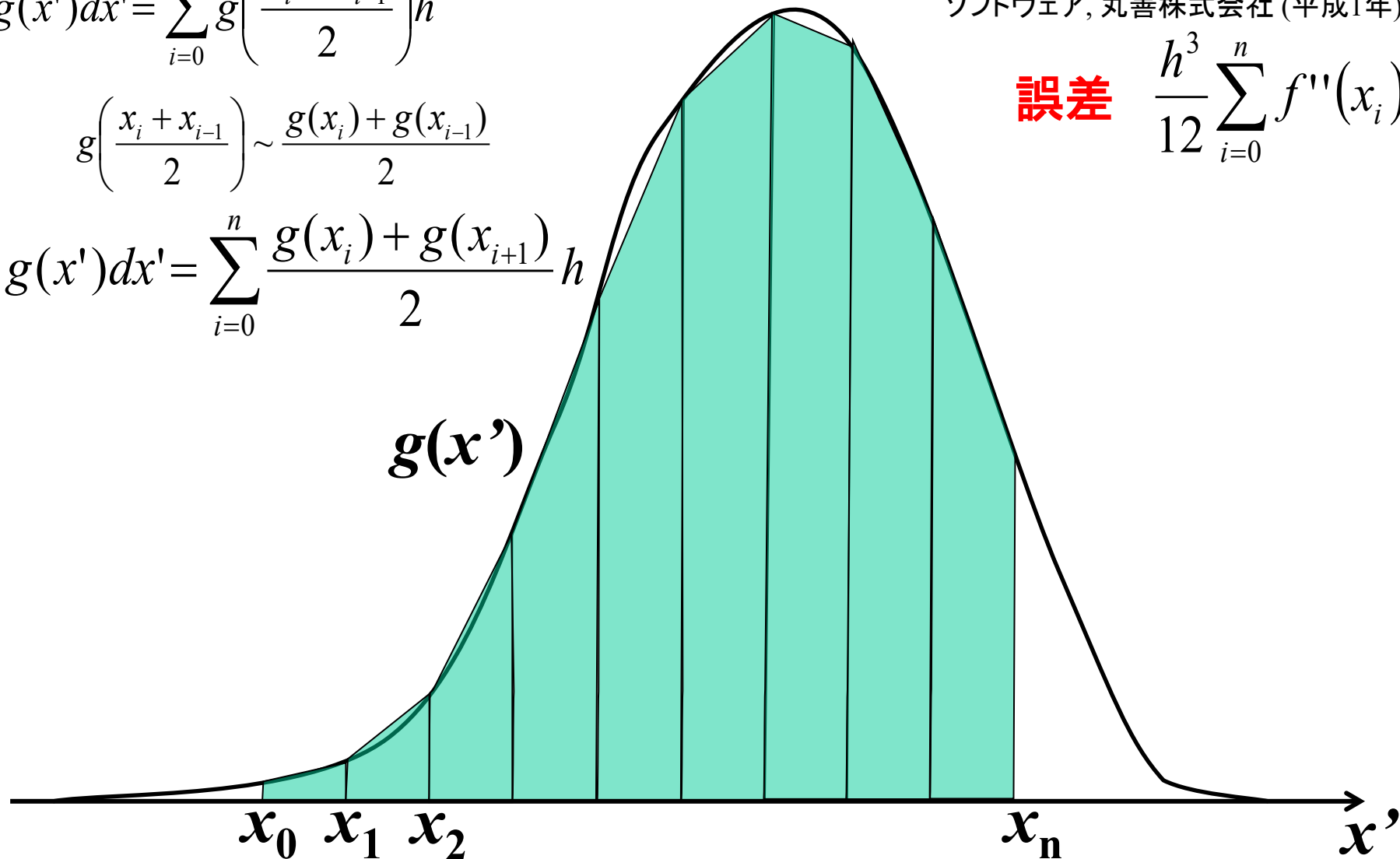
渡部力ら監修、Fortran77による数値計算  
ソフトウェア, 丸善株式会社 (平成1年)

$$\int_{x_0}^x g(x') dx' = \sum_{i=0}^n g\left(\frac{x_i + x_{i-1}}{2}\right) h$$

$$g\left(\frac{x_i + x_{i-1}}{2}\right) \sim \frac{g(x_i) + g(x_{i-1})}{2}$$

$$\int_{x_0}^x g(x') dx' = \sum_{i=0}^n \frac{g(x_i) + g(x_{i+1})}{2} h$$

**誤差**  $\frac{h^3}{12} \sum_{i=0}^n f''(x_i)$



# Simpson formula

**1. Approximate by**  $g(x_i) \sim g(x_1) + a_1(x_i - x_1) + a_2(x_i - x_1)^2$ ,  
**and determine  $a_i$  so as to reproduce  $f(x_0)$ ,  $f(x_1)$ , and  $f(x_2)$ .**  
 **$(x_i = x_1 - h, x_1, x_1 + h)$**

**2. Integrate the above approximation analytically  
for a range  $x = x_0 \sim x_0 + 2h$ :**

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

**3. For multiply divided range ( $x = x_0 \sim x_n = x_0 + nh$ ):**

$$\int_{x_0}^{x_n} g(x') dx' \sim \frac{h}{3} [g(x_0) + 4g(x_1) + 2g(x_2) + 4g(x_3) + 2g(x_4) + \cdots + g(x_n)]$$

# Derivation of the Simpson formula

**1. Approximate by**  $g(x_i) \sim g(x_1) + a_1(x_i - x_1) + a_2(x_i - x_1)^2$   
**and determine  $a_i$  so as to reproduce  $f(x_0)$ ,  $f(x_1)$ , and  $f(x_2)$ .**

**$(x_i = x_1 - h, x_1, x_1 + h)$**

$$\begin{array}{l} g(x_0) \sim g(x_1) - a_1 h + a_2 h^2 \\ g(x_2) \sim g(x_1) + a_1 h + a_2 h^2 \end{array} \quad \Rightarrow \quad a_1 = \frac{g(x_2) - g(x_0)}{2h} \quad a_2 = \frac{g(x_2) - 2g(x_1) + g(x_0)}{2h^2}$$

$$\begin{aligned} \int_{x_0}^{x_2} g(x') dx' &\sim g(x_1)x_2 + \frac{1}{2} \frac{g(x_2) - g(x_0)}{2h} (x_2 - x_1)^2 + \frac{1}{3} \left[ \frac{g(x_2) - 2g(x_1) + g(x_0)}{2h^2} \right] (x_2 - x_1)^3 \\ &\quad - \left\{ g(x_1)x_0 + \frac{1}{2} \frac{g(x_2) - g(x_0)}{2h} (x_0 - x_1)^2 + \frac{1}{3} \left[ \frac{g(x_2) - 2g(x_1) + g(x_0)}{2h^2} \right] (x_0 - x_1)^3 \right\} \\ &= 2g(x_1)h + 2 \left[ \frac{g(x_2) - 2g(x_1) + g(x_0)}{6} \right] h \end{aligned}$$

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

片岡勲 他、数値解析入門, コロナ社

$$\text{Error} \leq \frac{nh^5}{180} |f^{(4)}(x_i)|$$

# Comparison of numerical integration

$$g(x) = x^2$$

$$\int_0^x g(x') dx' = \frac{1}{3} x^3$$

<b>x</b>	<b>g(x)</b>	<b>Exact</b>	<b>Rieman</b>	<b>Trapezoid</b>	<b>Simpson</b>
0	0	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
0.2	0.04	<b>0.0027</b>	<b>0</b>	<b>0.004</b>	
0.4	0.16	<b>0.0213</b>	<b>0.008</b>	<b>0.024</b>	<b>0.021333</b>



# Series of Newton-Cotes formula

- **Trapezoid formula** (台形則)

$$\int_{x_1}^{x_2} f(x)dx = h \left[ \frac{1}{2} f_1 + \frac{1}{2} f_2 \right] + O(\underline{h^3} f'')$$

- **Simpson formula** (Simpson則)

$$\int_{x_1}^{x_3} f(x)dx = h \left[ \frac{1}{3} f_1 + \frac{4}{3} f_2 + \frac{1}{3} f_3 \right] + O(\underline{h^5} f^{(4)})$$

- **Simpson's 3/8 formula** (Simpsonの3/8則)

$$\int_{x_1}^{x_4} f(x)dx = h \left[ \frac{3}{8} f_1 + \frac{9}{8} f_2 + \frac{9}{8} f_3 + \frac{3}{8} f_4 \right] + O(\underline{h^5} f^{(4)})$$

- **Bode/Boole-Vilarceau formula** (Bode/Boole則)

$$\int_{x_1}^{x_5} f(x)dx = h \left[ \frac{14}{45} f_1 + \frac{64}{45} f_2 + \frac{24}{45} f_3 + \frac{64}{45} f_4 + \frac{14}{45} f_5 \right] + O(\underline{h^7} f^{(6)})$$

# Rieman/Trapezoid formula are better than Simpson formula for infinite-range integration

Simpson則より単純和/台形則の方が良い

$$\int_{-\infty}^{\infty} g(x') dx' \sim \frac{h}{3} [g(x_0) + 4g(x_1) + 2g(x_2) + 4g(x_3) + 2g(x_4) + \cdots + g(x_n)]$$

For infinite-range integration  $(-\infty \sim \infty)$ ,  $x_0$  and  $x_n$  are not essential.

$$\int_{-\infty}^{\infty} g(x') dx' \sim \frac{h}{3} [g(x_{-1}) + 4g(x_0) + 2g(x_1) + 4g(x_2) + 2g(x_3) + \cdots + g(x_{n-1})]$$

$$\int_{-\infty}^{\infty} g(x') dx' \sim \frac{h}{3} [g(x_0) + 4g(x_1) + 2g(x_2) + 4g(x_3) + \cdots + g(x_n)]$$

also provides the essentially the same result.

$$\int_{x_0}^{x_n} g(x') dx' \sim \frac{h}{3} [0.5g(x_{-1}) + 2.5g(x_0) + 3g(x_1) + 3g(x_2) + 3g(x_3) + 3g(x_4) + \cdots + 0.5g(x_n)]$$

Considering  $g(x_{-1})$  and  $g(x_n)$  are negligible for infinite integration leads to

$$\int_{x_0}^{x_n} g(x') dx' \sim h [g(x_1) + g(x_2) + g(x_3) + g(x_4) + \cdots + g(x_{n-2})]$$

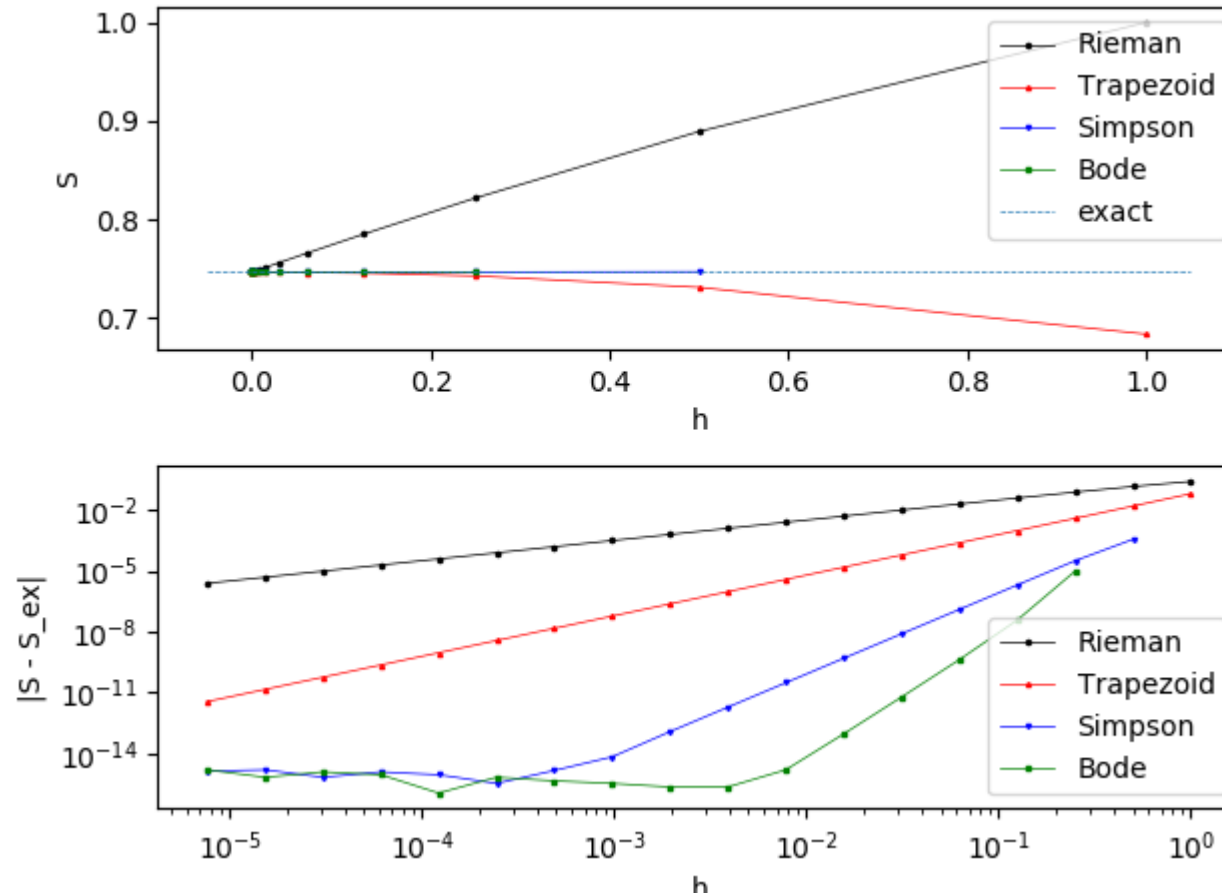
, which is the same as the Rieman sum and the trapezoid formula.

# Program: integ\_order\_h.py

$$g(x') = \exp(-x'^2), \int_{x_0}^{x_1} g(x') dx' = \text{erf}(x_1) - \text{erf}(x_0)$$

$$[x_0, x_1] = [0, 1.0], \text{exact} = 0.746824132812427$$

Run: **python integ\_order\_h.py 0 1 18 gauss**



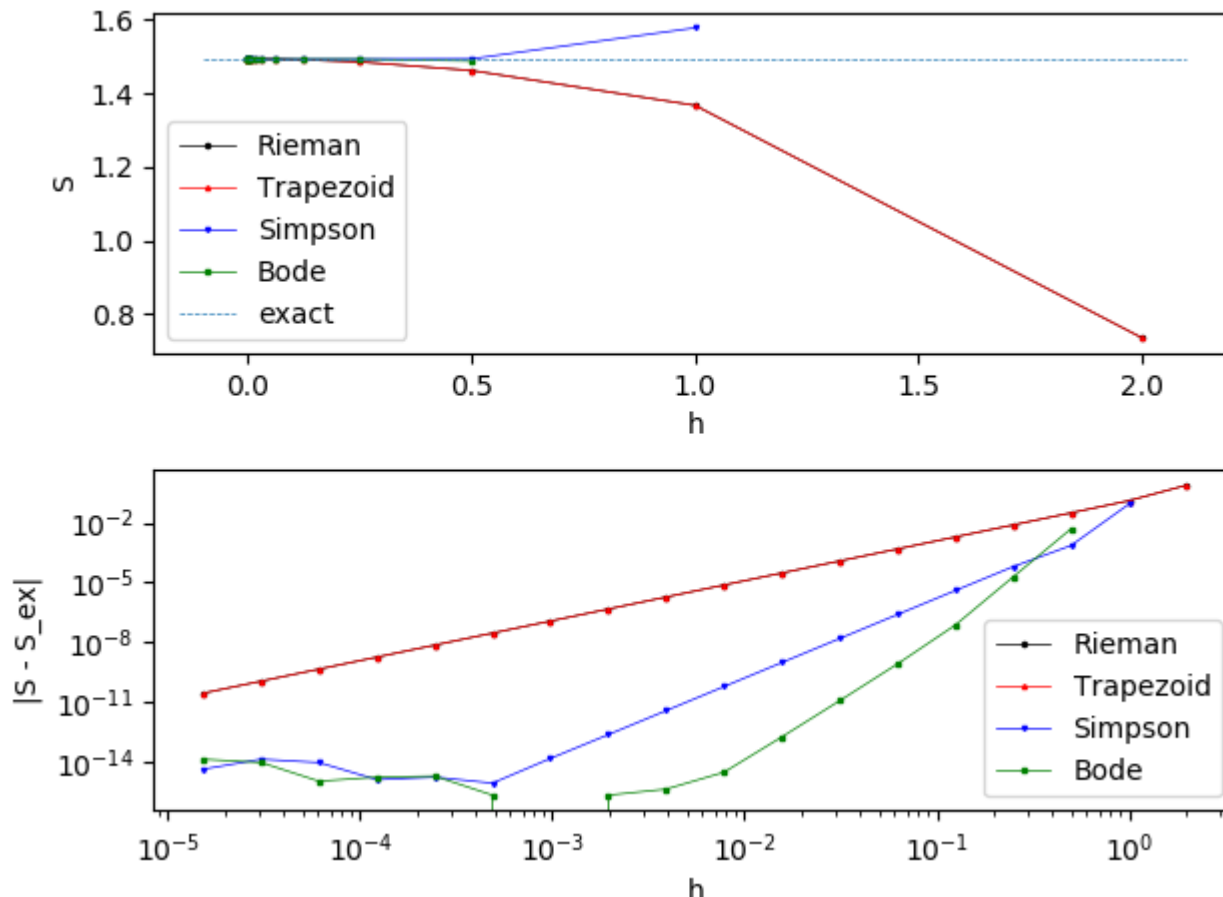
Trapezoid approx. is better than Riemann sum for asymmetric function over **finite range**

# Program: integ\_order\_h.py

$$g(x') = \exp(-x'^2), \int_{x_0}^{x_1} g(x') dx' = \text{erf}(x_1) - \text{erf}(x_0)$$

$$[x_0, x_1] = [-1.0, 1.0], \text{exact} = 1.493648265624854$$

Run: **python integ\_order\_h.py -1 1 18 gauss**



Trapezoid approx. is better than Rieman sum also for symmetric integration over **finite range**

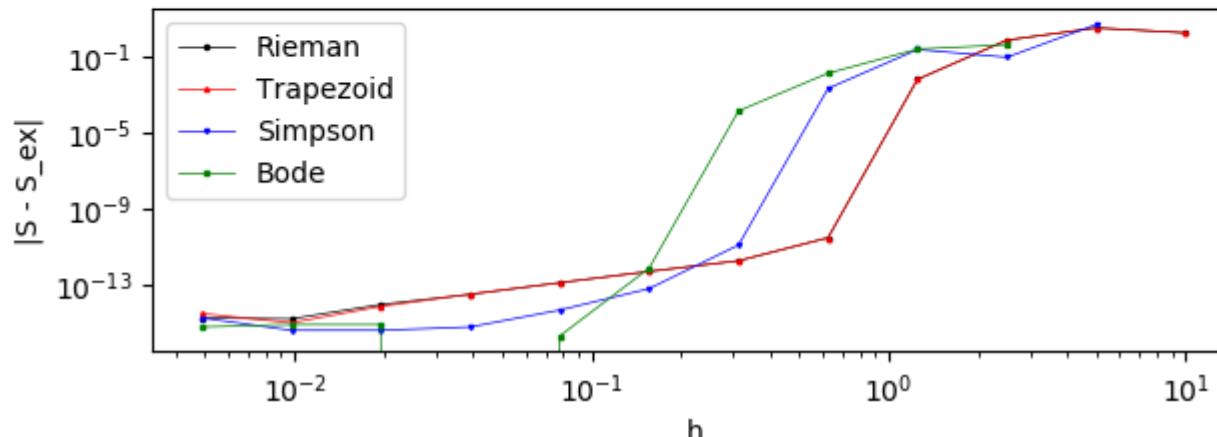
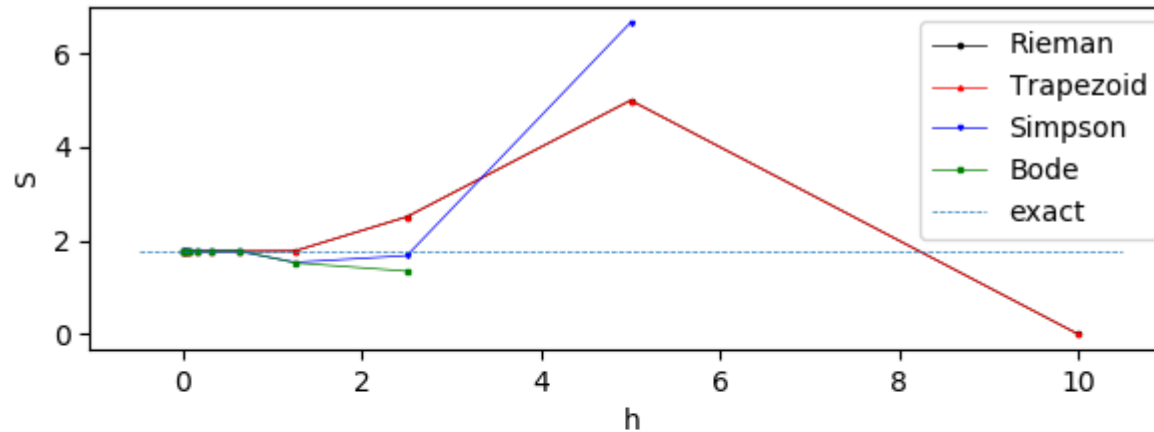
# Program: integ\_order\_h.py

$$g(x') = \exp(-x'^2), \int_{x_0}^{x_1} g(x') dx' = \text{erf}(x_1) - \text{erf}(x_0)$$

$$[x_0, x_1] = [-5, 5], \text{exact} = 1.772453850902791 (\sim\sqrt{\pi})$$

**Note: The range  $[-5, 5]$  is virtually equivalent to infinite integration range as  $\exp(-25)$  can be negligible**

Run: **python integ\_order\_h.py -5 5 12 gauss**



Simpson method loses accuracy for integration over **infinite range**

# Features of other numerical integrations

**Newton-Cotes formula: Analytically integrate approximated polynomial that exactly takes  $g(x)$  with uniform integration points.**

(積分範囲を等分割し、各積分点を通る多項式で近似して解析的に積分する)

- Trapezoid formula (first order) (台形則, 一次式)
- Simpson formula (second/third order) (Simpson則, 二次式、三次式)
- Bode/Boole formula (fourth order) (Bode/Bool則, 四次式)

**Maximize precision by optimize both weights and integration points**

(計算点位置も含めて精度が最大になるようにする)

**(High precision, Non-uniform points)** (精度は高い、積分点が等間隔でない)

- Gauss-Legendre formula
- Gauss-Chebyshev formula

**Interpolation type** (補間型) **(Better precision?)**

- Spline integration (スプライン積分)

**Extrapolation type** (補外型) **(Controlled precision)**

- Romberg integration (ロンバーグ積分)

**Variable conversion type** (変数変換型) **(better for infinite integration, anomaly points)** 無限積分や特異点を含む積分に有利)

# Gauss-Legendre method

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)

- **Choose  $n$  integration points  $x_i$  and weights  $w_i$  so as to minimize the integration error by  $(2n-1)$  order polynomial.**

積分区間に  $n$  個の積分点を選ぶ際、積分点と重みの  $2n$  個のパラメータを  $f(x)$  が  $(2n-1)$  次の多項式に一致するように 決める。

- **Can integrate a function with anomaly points.**

端点を含まないなので、積分区間端に特異点があっても計算できる

- **Best accuracy for good functions in finite integration range.**

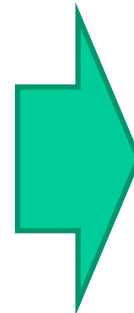
有限区間で解析的な関数の積分では最も精度が高い

- **Integration points  $x_i$  are given as the zero points of Legendre polynomial.**

分点はLegendre多項式の零点

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n = 0$$

$$w_i = \frac{2(1-x^2)}{(n+1)^2 [P_{n+1}(x_i)]^2}$$



$$S = \sum_{i=1}^n f(x_i) w_i$$

# Gauss-Legendre method: fractional coordinates and weights (分点と重み)

Fractional coordinates (分点)	Weight (重み係数)
<b>Four points formula (4 点公式)</b>	
−0.861136311594052575223946488892	0.347854845137453857373063949221
−0.339981043584856264802665759103	0.652145154862546142626936050778
+0.339981043584856264802665759103	0.652145154862546142626936050778
+0.861136311594052575223946488892	0.347854845137453857373063949221
<b>Five points (5点公式)</b>	
−0.906179845938663992797626878299	0.236926885056189087514264040719
−0.538469310105683091036314420700	0.478628670499366468041291514835
0	0.568888888888888888888888888888
+0.538469310105683091036314420700	0.478628670499366468041291514835
+0.906179845938663992797626878299	0.236926885056189087514264040719
<b>Six points (6 点公式)</b>	
−0.932469514203152027812301554493	0.171324492379170345040296142172
−0.661209386466264513661399595019	0.360761573048438607569833513837
−0.238619186093196908630501721680	0.467913934572691047389870343989
+0.238619186093196908630501721680	0.467913934572691047389870343989
+0.661209386466264513661399595019	0.360761573048438607569833513837
+0.932469514203152027812301554493	0.171324492379170345040296142172
<b>Seven points (7 点公式)</b>	
−0.949107912342758524526189684047	0.129484966168869693270611432679
−0.741531185599394439863864773280	0.279705391489276667901467771423
−0.405845151377397166906606412076	0.381830050505118944950369775488
0	0.417959183673469387755102040816
+0.405845151377397166906606412076	0.381830050505118944950369775488
+0.741531185599394439863864773280	0.279705391489276667901467771423
+0.949107912342758524526189684047	0.129484966168869693270611432679



# Extrapolation method: Romberg integration

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 integration without anomaly points**

▪ Start from the Trapezoid formula, and sequentially apply higher order Newton-Cotes precision formula.

(台形則から出発し、高次のニュートン・コーツ型に相当する公式を自動的に適用し、要求精度を満たすまで続ける)

1. Integrate by the Trapezoid formula in  $[a, b]$  with the mesh  $h_0$

$$\Rightarrow S_{0,0}$$

2. Decrease mesh to  $h_1 = (1/2)h_0$  and integrate all the range

$$\Rightarrow S_{1,0}$$

3. Decrease mesh to  $h_k = (1/2)h_{k-1}$  and integrate all the range

$\Rightarrow S_{k,0}$ , and calculate  $S_{k,d}$  ( $d = 1, 2, \dots, k$ ) by

$$S_{k,d} = \frac{4^d S_{k,d-1} - S_{k-1,d-1}}{4^d - 1}$$

4.  $S_{k,k}$  will be the approximated integration values.

Stop if  $|S_{k,k} - S_{k-1,k-1}|$  becomes smaller than the required accuracy.

# Error of numerical integration:

## Monotone increasing function

$$S = \int_{-1}^1 \exp(x) dx \quad \text{Exact: } \exp(1) - \exp(-1) = 2.3504023872876$$

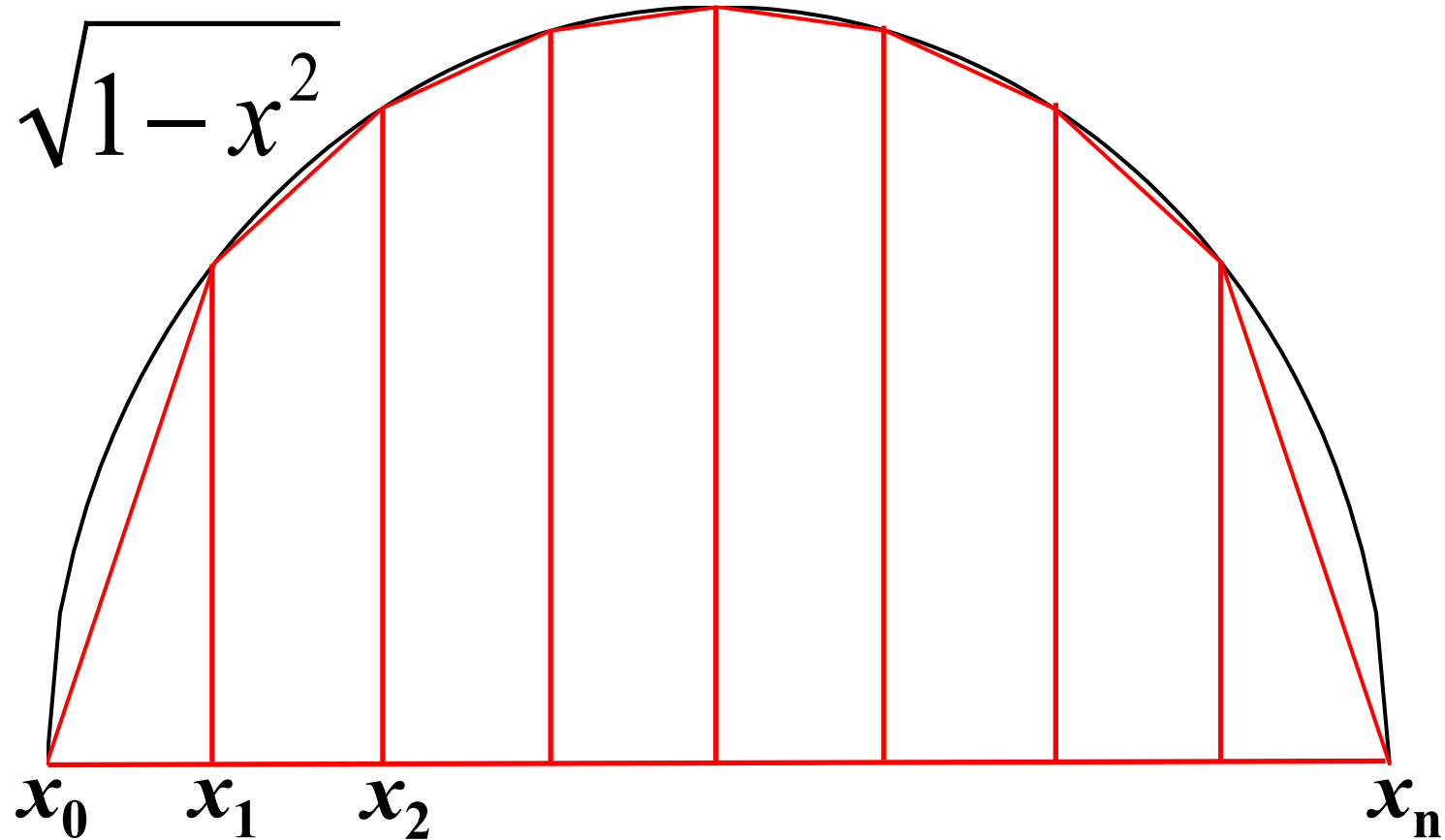
nDivide	Rieman	Trapezoid	Simpson	Simpson 3/8	Bode	Romberg	Cubic Spline	Order 3 Gauss- Legendre
1	1.61E+00	-7.36E-01				-7.36E-01		
2	9.83E-01	-1.93E-01	-1.17E-02			-1.17E-02		6.55E-05
3	6.97E-01	-8.64E-02		-5.25E-03				
4	5.39E-01	-4.88E-02	-7.92E-04		-6.85E-05	-6.85E-05	7.19E-03	1.13E-06
5	4.39E-01	-3.13E-02					3.75E-03	
6	3.70E-01	-2.17E-02	-1.59E-04	-3.53E-04			2.35E-03	1.01E-07
7	3.20E-01	-1.60E-02					1.54E-03	
8	2.82E-01	-1.22E-02	-5.06E-05		-1.18E-06	-1.07E-07	1.07E-03	1.81E-08
9	2.51E-01	-9.66E-03		-7.08E-05			7.73E-04	
10	2.27E-01	-7.83E-03	-2.08E-05				5.77E-04	4.75E-09
11	2.07E-01	-6.47E-03					4.41E-04	
12	1.90E-01	-5.44E-03	-1.00E-05	-2.25E-05	-1.05E-07		3.45E-04	1.59E-09
13	1.76E-01	-4.63E-03					2.75E-04	
14	1.64E-01	-4.00E-03	-5.43E-06				2.23E-04	6.32E-10
15	1.53E-01	-3.48E-03		-9.25E-06			1.83E-04	
16	1.44E-01	-3.06E-03	-3.18E-06		-1.88E-08	-4.21E-11	1.52E-04	2.84E-10
17	1.36E-01	-2.71E-03					1.28E-04	
18	1.28E-01	-2.42E-03	-1.99E-06	-4.46E-06			1.08E-04	1.40E-10
19	1.22E-01	-2.17E-03					9.27E-05	
20	1.16E-01	-1.96E-03	-1.30E-06		-4.95E-09		7.99E-05	7.45E-11
32						-3.55E-15		

# 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  $(-\infty, \infty)$**

**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  $x \Rightarrow u$ :  $x = \varphi(u)$**

**Calculate by the Trapezoid formula**

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

# 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)**

$$x = \phi(u) = \frac{1}{Q} \int_0^u \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt \quad \phi'(u) = \frac{1}{Q} \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right)$$
$$Q = \int_0^1 \exp\left(-\frac{1}{t} - \frac{1}{1-t}\right) dt = 0.00702985841$$

**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

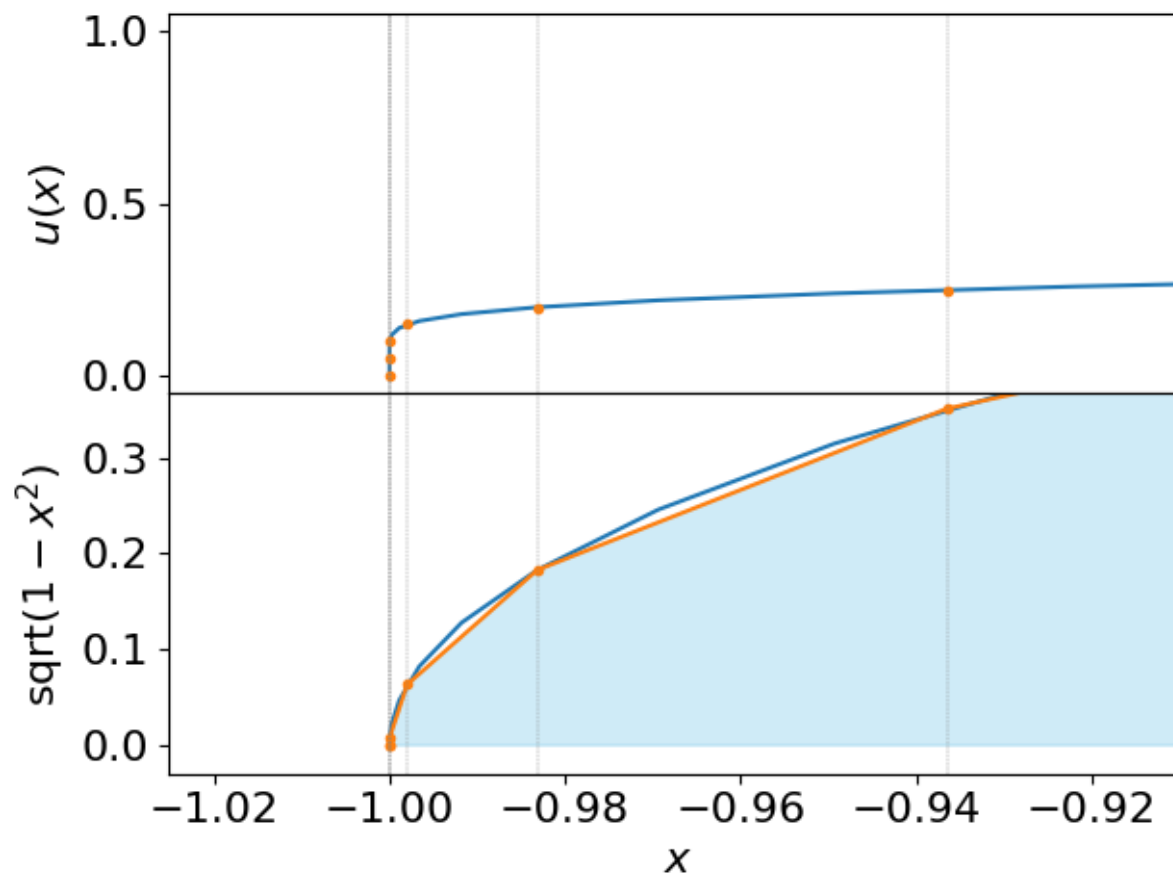
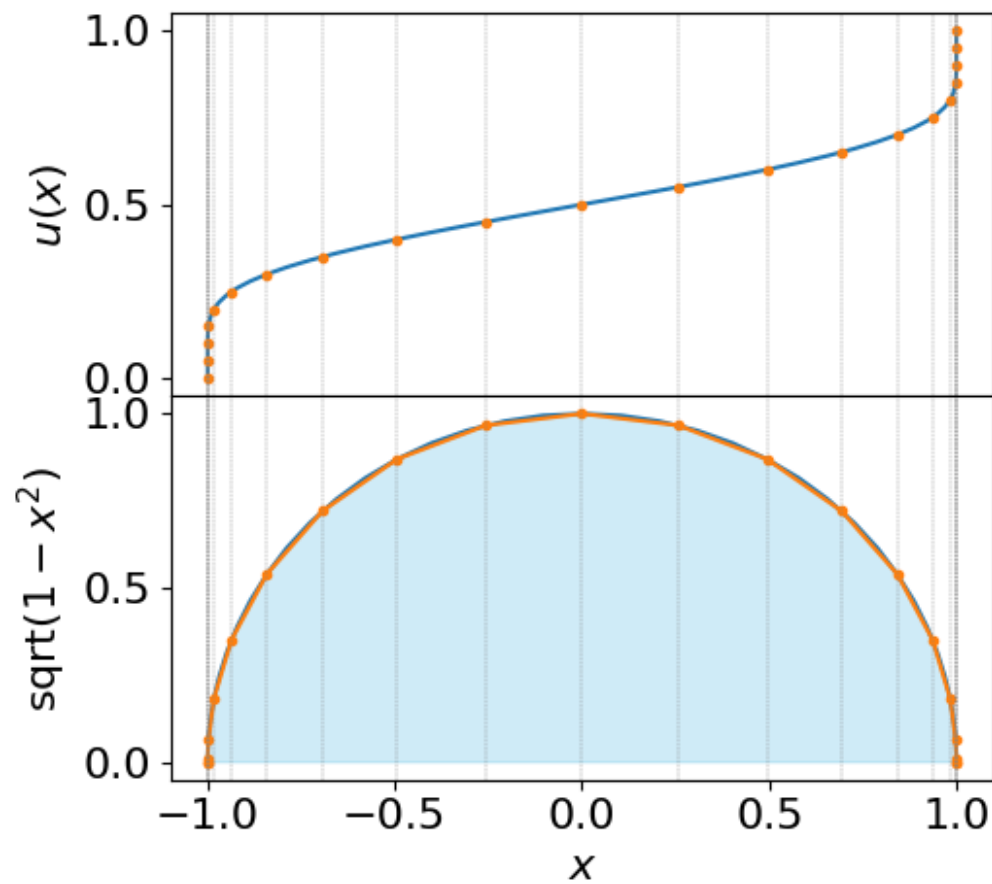
## 伊理・森口・高沢(IMT)の公式

> `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**  $\int_{-1}^1 f(x) dx$

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

**For**  $\int_0^\infty f(x) dx$

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

**For**  $\int_0^\infty f(x) dx$  where  $f(x)$  includes  $\exp(-x)$  type factor

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

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

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

# 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)

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

$x$  range:  $[-1, 1]$

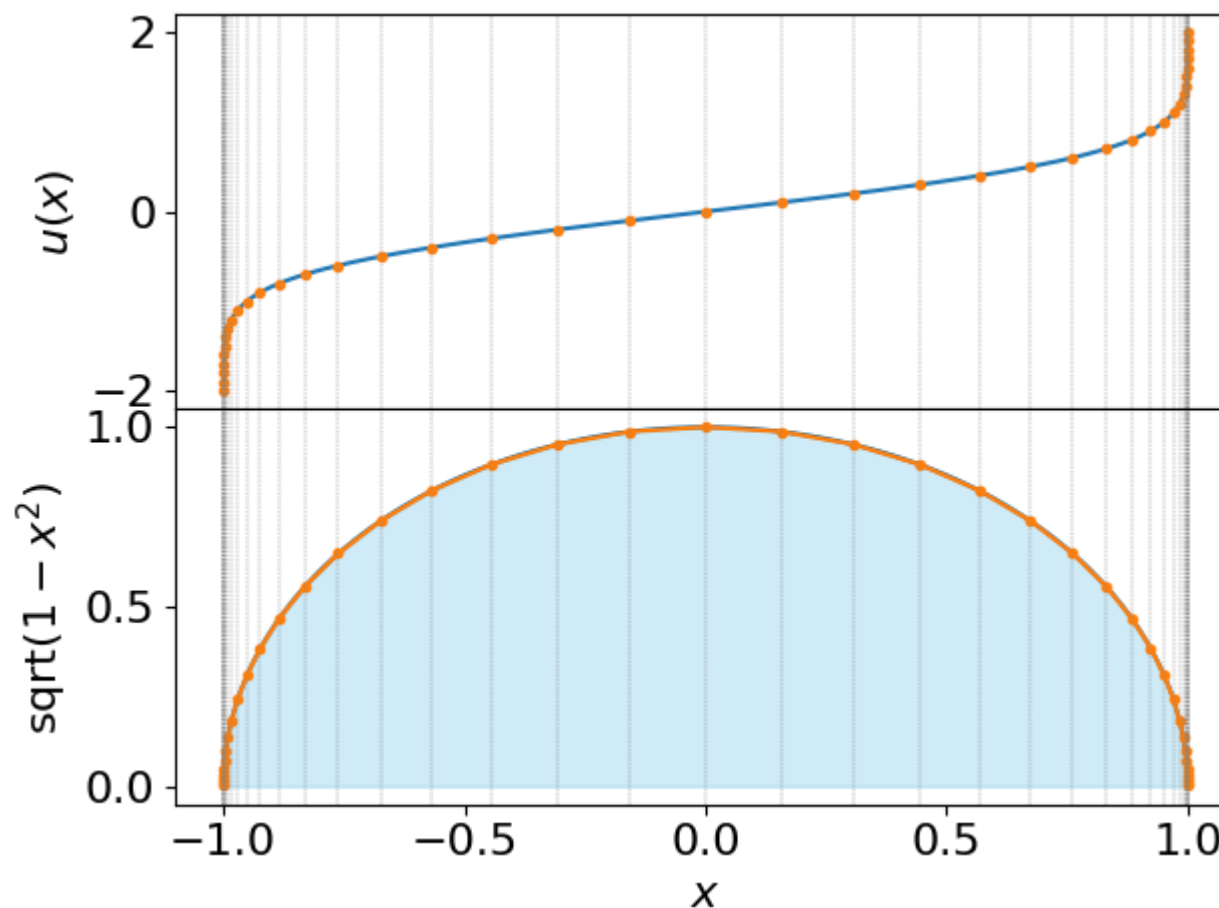
$u$  range:  $[-2, 2] \Rightarrow x$  range  $\sim [-1, 1]$

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

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

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

> python integ\_double\_exp\_-1\_1.py





# 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)

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

x range:  $[0, \infty]$

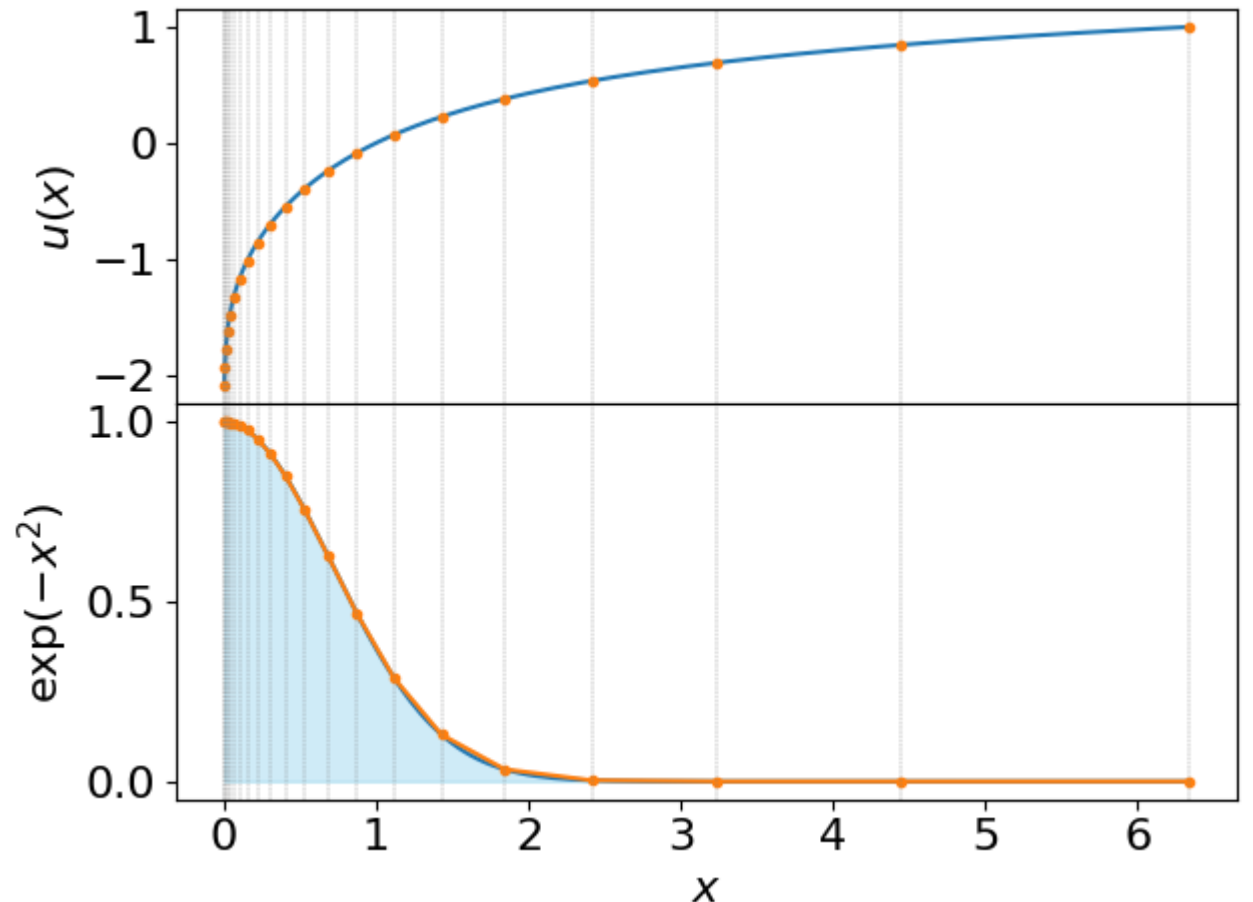
u range:  $[-2, 1] \Rightarrow$  x range  $\sim [0, 6]$

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

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

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

> python integ\_double\_exp\_0\_inf.py



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

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

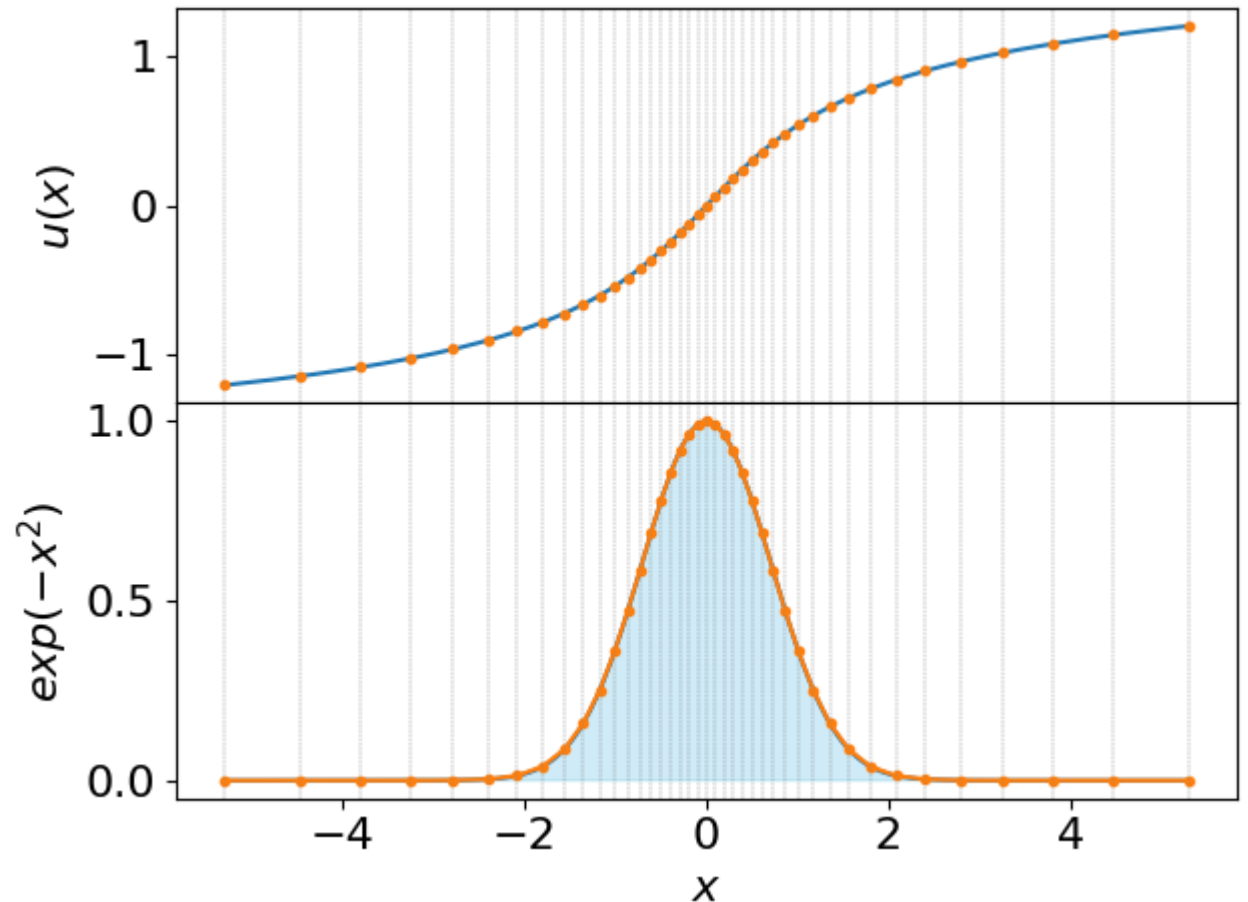
u range:  $[-1.2, 1.2] \Rightarrow$  x range  $\sim [-5, 5]$

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

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

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

> python integ\_double\_exp\_inf\_inf.py



# Error for integration with anomaly points

$$S = \int_{-1}^1 \sqrt{1-x^2} dx \quad \text{Exact: } \pi/2 = 1.5707963$$

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)

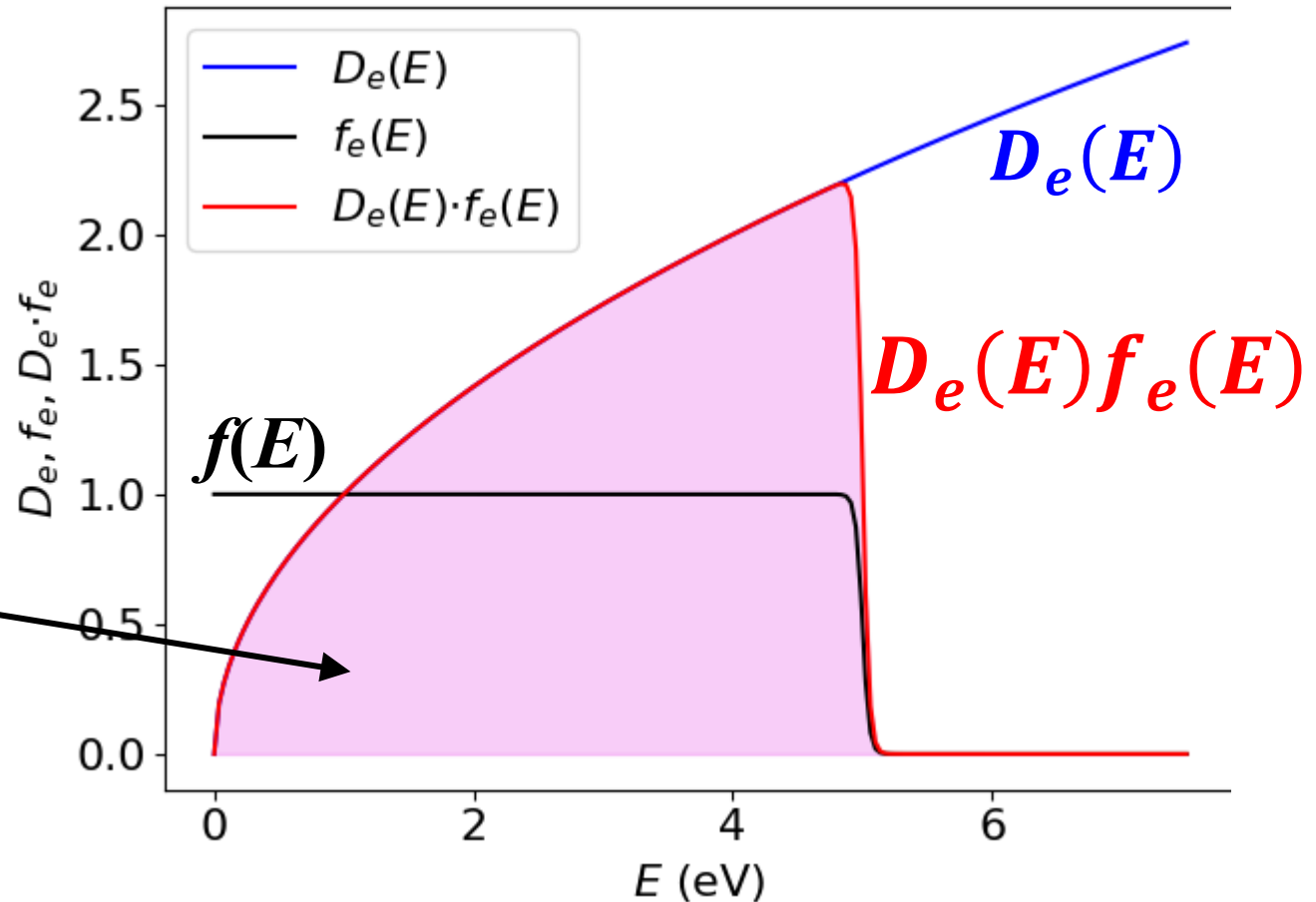
$$D_e(E) = D_{e0} \sqrt{E}$$

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

## Number of electrons in CB

$$N = \int_0^\infty D_e(E) f(E) dE$$

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



# 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  $\sim \exp(-\alpha)$ )
- 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  $\Delta E$  should be fine enough  
(e.g.,  $\Delta E < \alpha k_B T / 100$ , 1 meV)

=> We should not the same  $\Delta E$  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

Measure time by repeating for 300 times

**Precision 8 digits (epsrel = 1e-8),  $\alpha = 6$ :**

**Integ. range Time for 300 repetition**

(1)  $0 \sim E_F + \alpha k_B T$  0.109 秒

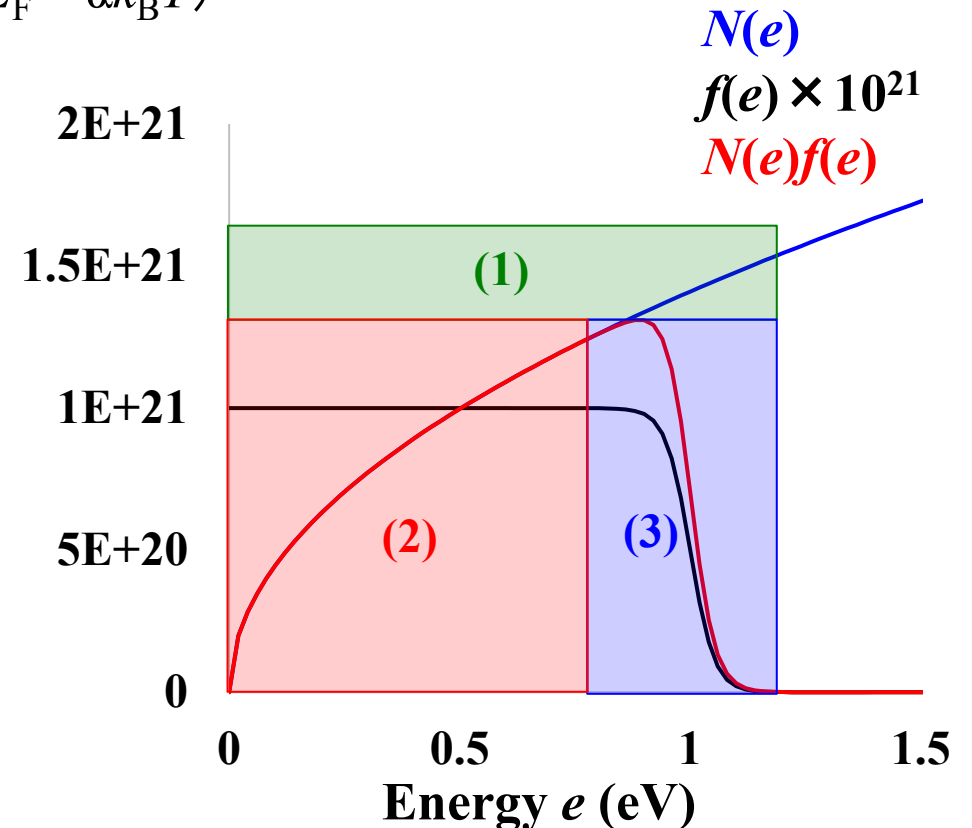
(2)  $0 \sim E_F - \alpha k_B T$  0.063 秒

(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

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

Use quad() function (adaptive polynomial integration) in the scipy library of python

cf: <https://org-technology.com/posts/integrate-function.html>

**python debye\_function.py 300 0 500 10**

Debye temperature 300 K

Temperate range 0 – 500 K, 10 K step

