Numerical integration (quadrature)

数値積分 (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):

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

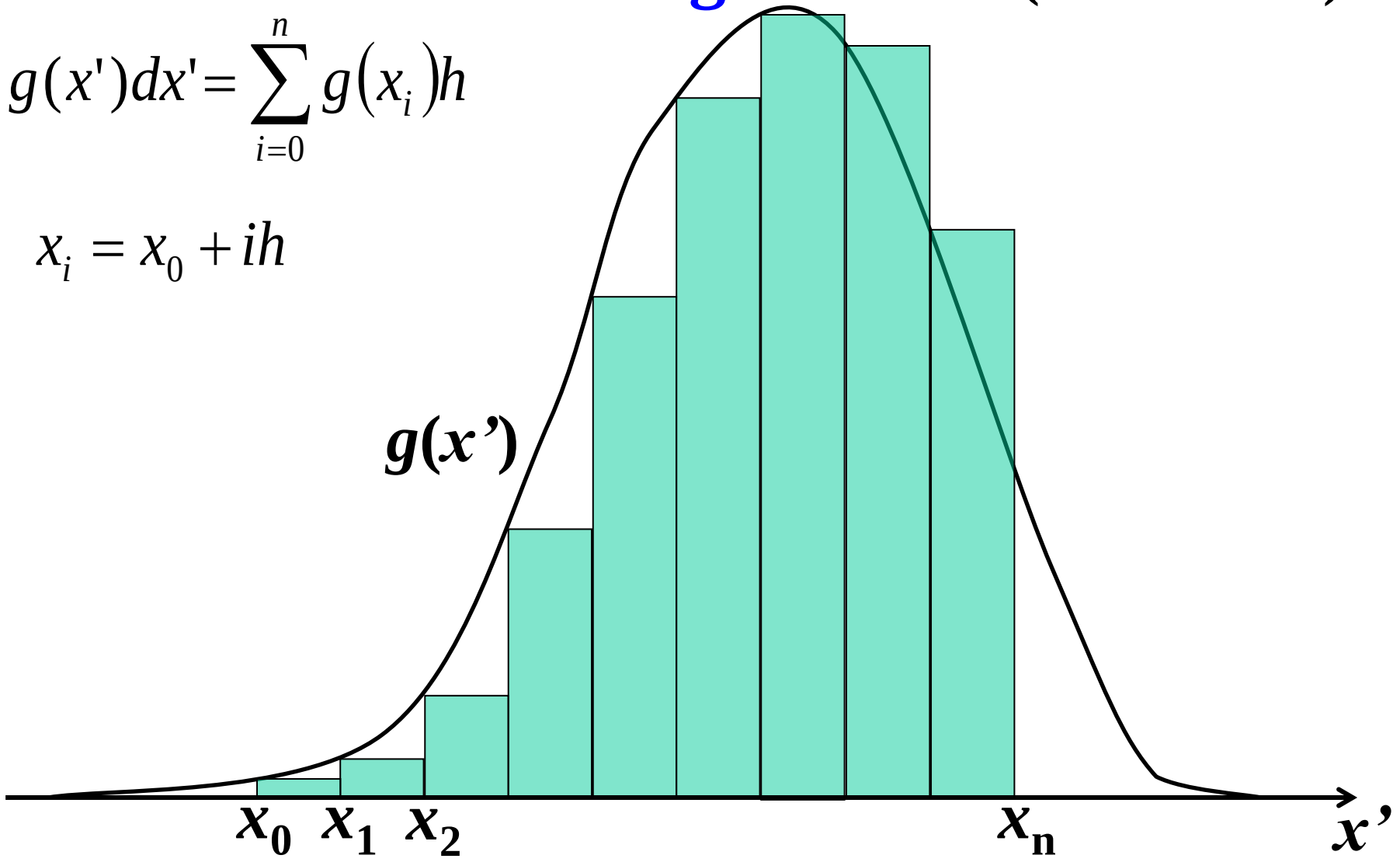
$$F(x+h)=F(x)+g(x)h=F(x-h)+\left[g(x)+g(x-h)\right]h$$

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

Riemann sum / rectangular rule (Riemann和)

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

$$x_i = x_0 + ih$$



Asymmetric formula:

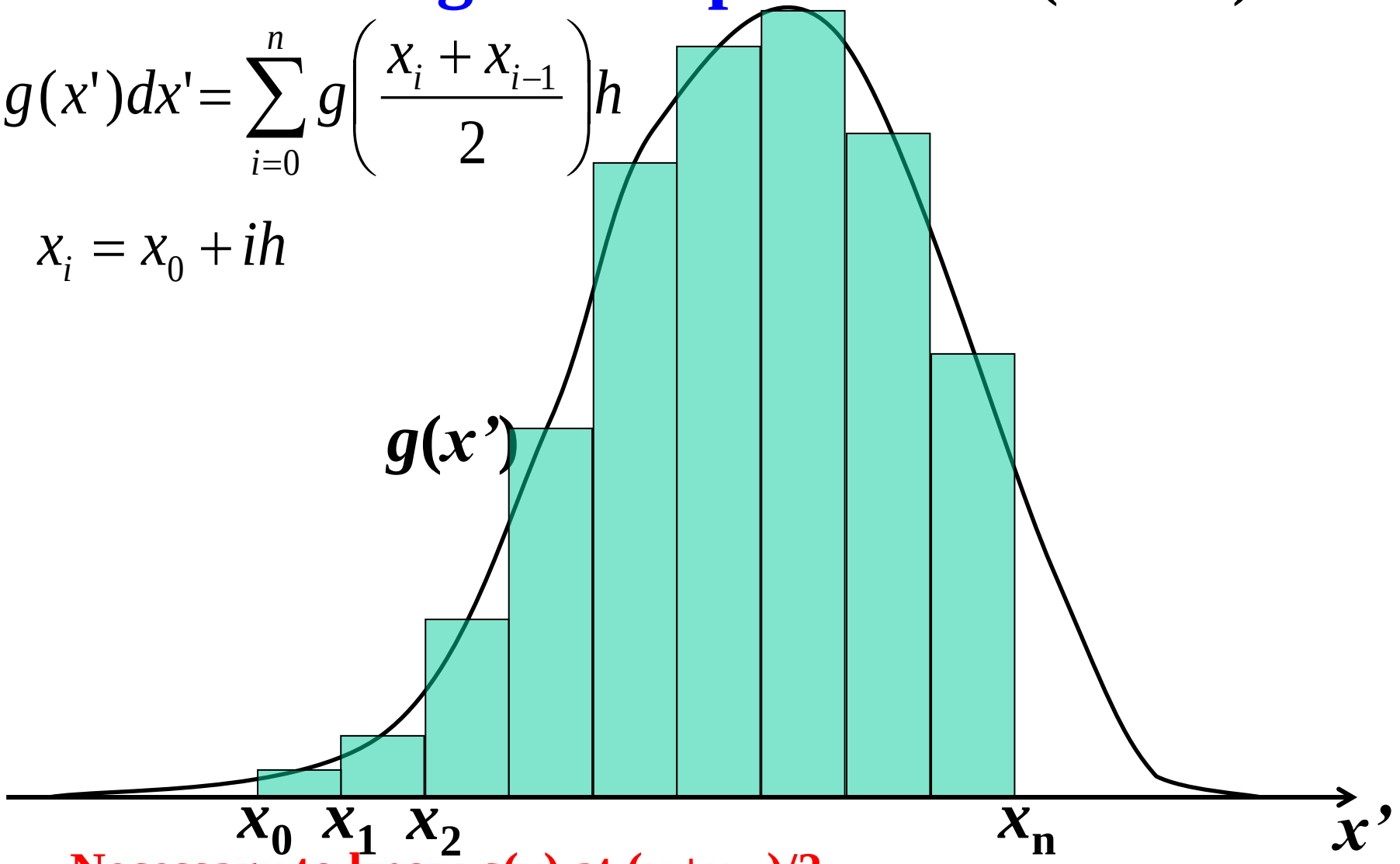
monotone increasing $g(x)$ => Underestimation (過小評価)

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

Take average: Mid-point rule (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 rule (台形則)

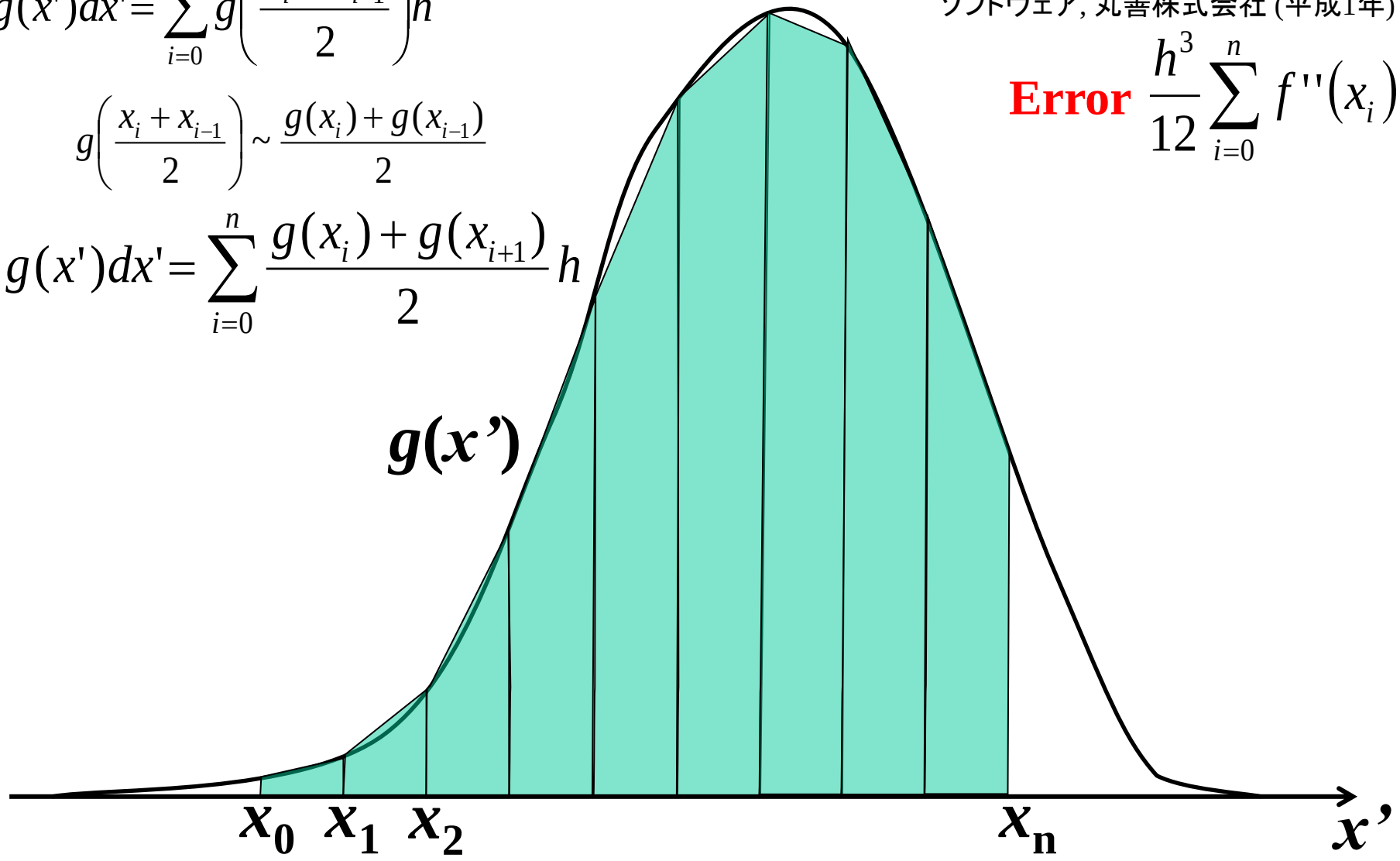
$$\int_{x_0}^{x} g(x')dx' = \sum_{i=0}^{n} g\left(\frac{x_i + x_{i-1}}{2}\right)h$$

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

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

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)

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



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 $x = x_0 \sim x_0 + 2h$:

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

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

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

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 $f(x_2)$.
($x_i = x_1 - h, x_1, x_1 + h$)

$$g(x_0) \sim g(x_1) - a_1 h + a_2 h^2$$
$$g(x_2) \sim g(x_1) + a_1 h + a_2 h^2$$

$$\Rightarrow \quad a_1 = \frac{g(x_2) - g(x_0)}{2h} \qquad a_2 = \frac{g(x_2) - 2g(x_1) + g(x_0)}{2h^2}$$

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

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

片岡勲 他、数値解析入門, コロナ社

$$\textbf{Error} \le \frac{nh^5}{180}\left|f^{(4)}(x_i)\right|$$

Comparison of numerical integration

$g(x) = x^2$

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

x	g(x)	Exact	Rieman	Trapezoid	Simpson
0	0	**0**	**0**	**0**	**0**
0.2	0.04	**0.0027**	**0**	**0.004**	
0.4	0.16	**0.0213**	**0.008**	**0.024**	**0.021333**

Series of Newton-Cotes formula

- **Trapezoid formula** (台形則)

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

- **Simpson formula** (Simpson則)

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

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

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

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

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

Riemann sum/Trapezoid rule 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 (-∞ ~ ∞), x_0 and x_n are not essential.

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

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

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

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

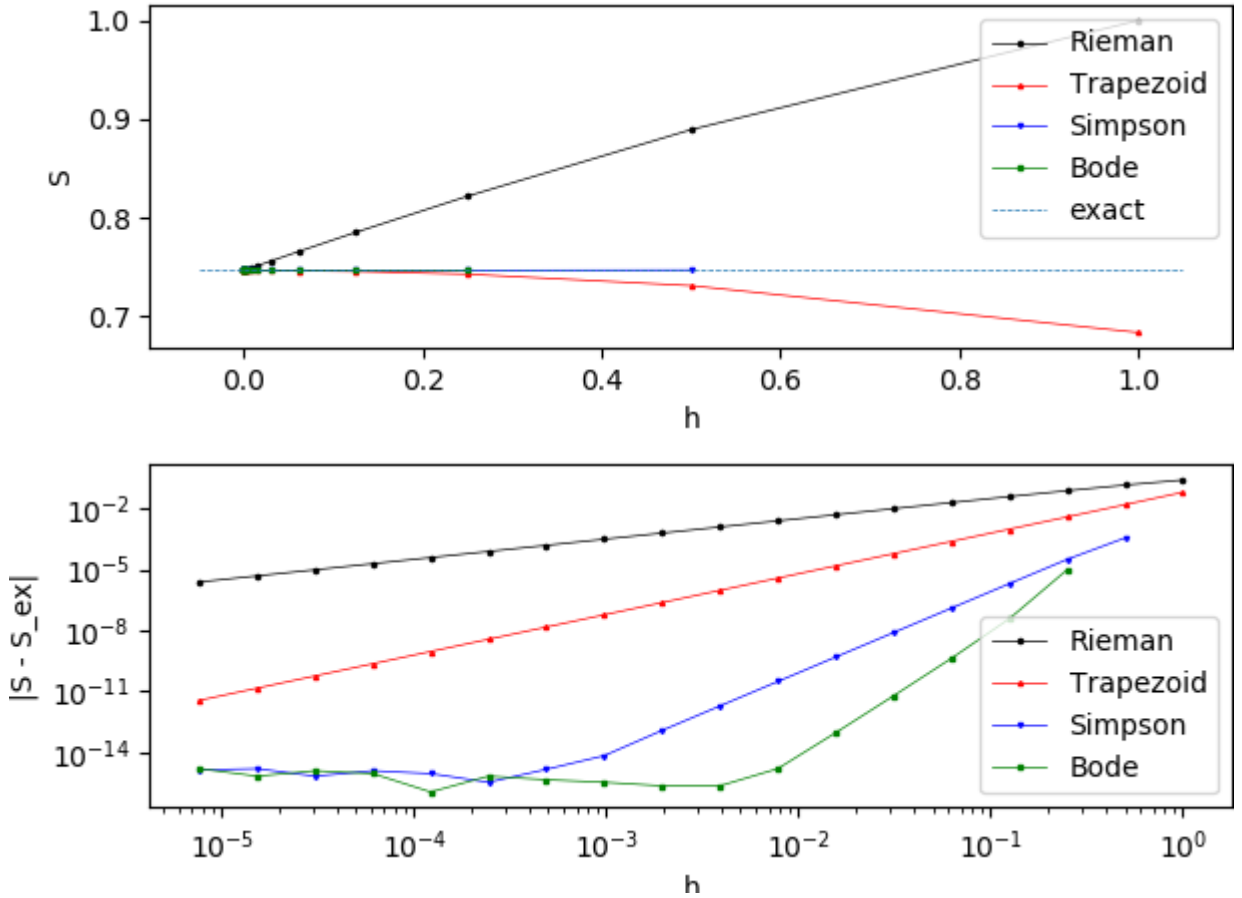
, which is **the same as the Riemann sum and the trapezoid formula**.

Program: integ_order_h.py

$g(x') = \exp(-x^2), \int_{x0}^{x1} g(x')dx' = \text{erf}(x_1) - \text{erf}(x_0)$

$[x_0, x_1] = [0, 1.0]$, exact = 0.746824132812427

Run: **python integ_order_h.py 0 1 18 gauss**



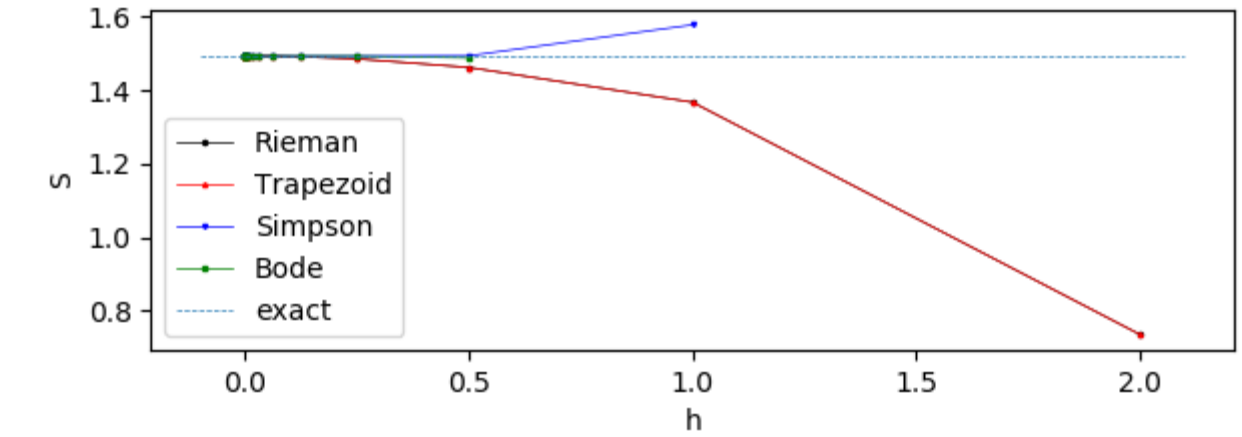
Trapezoid approx. is better than Riemann sum for asymmetric function over **finite range**

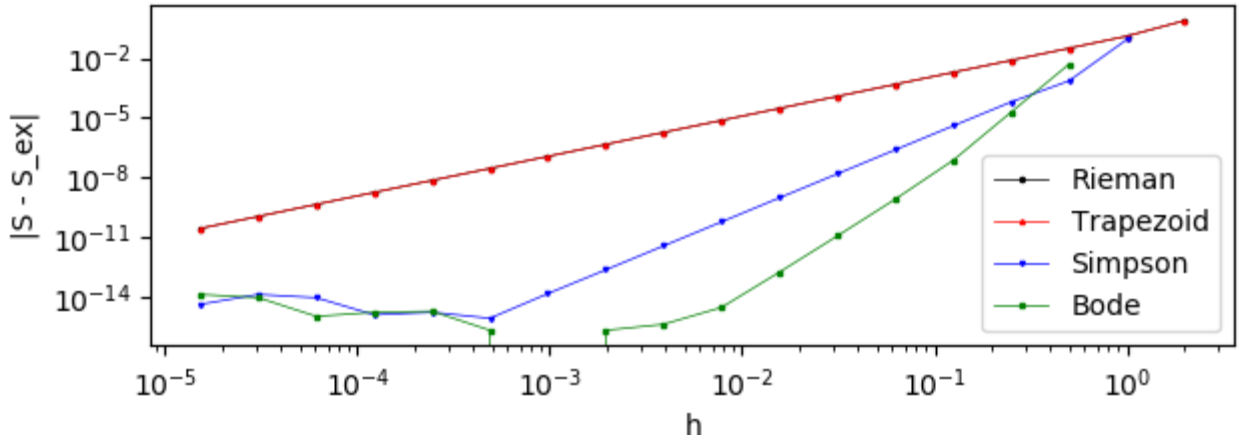
Program: integ_order_h.py

$g(x') = \exp(-x^2), \int_{x0}^{x1} g(x')dx' = \mathrm{erf}(x_1) - \mathrm{erf}(x_0)$

$[x_0, x_1]$ = [-1.0, 1.0], exact = 1.493648265624854

Run: **python integ_order_h.py -1 1 18 gauss**





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

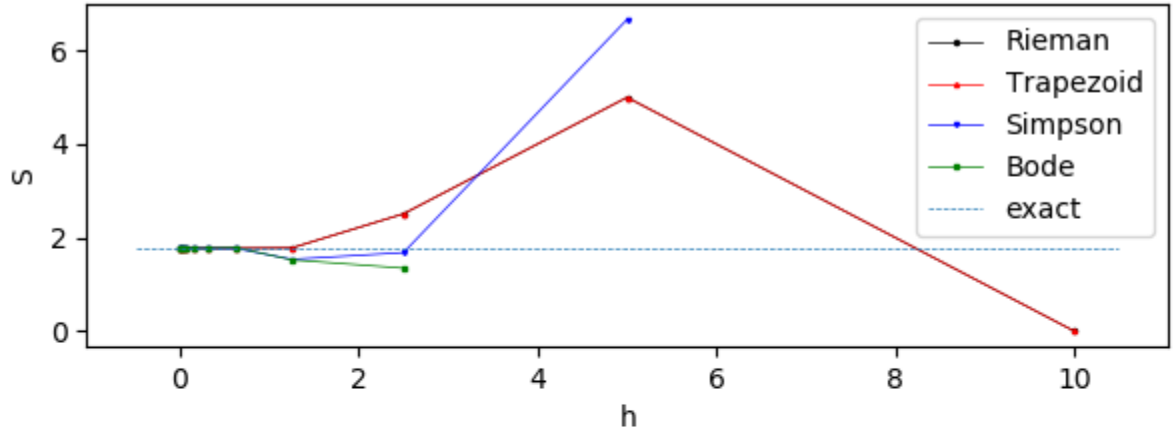
Program: integ_order_h.py

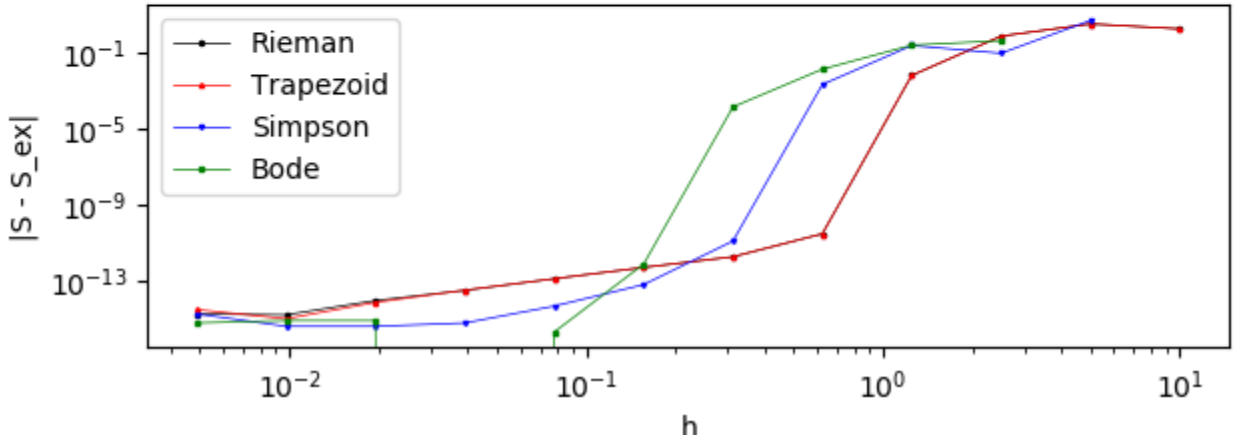
$g(x') = \exp(-x^2), \int_{x0}^{x1} g(x')dx' = \mathrm{erf}(x_1) - \mathrm{erf}(x_0)$

$[x_0, x_1]$ = [-5, 5], 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 rule (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, singular 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* nodes and weights so that all polynomials up to degree $2n-1$ are integrated exactly.**

  n個の分点と重みを選び、$2n-1$次以下の多項式を厳密に積分できるようにする。

- **Because the endpoints are not used as integration nodes, it can handle some integrable endpoint singularities.**

  端点を含まないので、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

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

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

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

Gauss-Legendre method: nodes and weights (分点と重み)

Fractional coordinates (分点)	Weight (重み係数)
Four points formula (4 点公式)	
−0.86113631159405257522394648892	0.34785484513745385737306394922
−0.33998104358485626480266575910	0.65214515486254614262693605078
+0.33998104358485626480266575910	0.65214515486254614262693605078
+0.86113631159405257522394648892	0.34785484513745385737306394922
Five points (5点公式)	
−0.90617984593866399279762687829	0.23692688505618908751426404071
−0.53846931010568309103631442070	0.47862867049936646804129151483
0	0.56888888888888888888888888888
+0.53846931010568309103631442070	0.47862867049936646804129151483
+0.90617984593866399279762687829	0.23692688505618908751426404071
Six points (6 点公式)	
−0.93246951420315202781230155449	0.17132449237917034504029614217
−0.66120938646626451366139959502	0.36076157304843860756983351383
−0.23861918609319690863050172168	0.46791393457269104738987034398
+0.23861918609319690863050172168	0.46791393457269104738987034398
+0.66120938646626451366139959502	0.36076157304843860756983351383
+0.93246951420315202781230155449	0.17132449237917034504029614217
Seven points (7 点公式)	
−0.94910791234275852452618968405	0.12948496616886969327061143268
−0.74153118559939443986386477328	0.27970539148927666790146777142
−0.40584515137739716690660641208	0.38183005050511894495036977549
0	0.41795918367346938775510204082
+0.40584515137739716690660641208	0.38183005050511894495036977549
+0.74153118559939443986386477328	0.27970539148927666790146777142
+0.94910791234275852452618968405	0.12948496616886969327061143268

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 singular points

- **Start from the Trapezoid formula,**
  and sequentially apply higher order Newton-Cotes formula
  until a given precision is satisfied.
  (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 => $S_{0,0}$**
2. **Decrease mesh to $h_1 = (1/2)h_0$ and integrate all the range**
   => $S_{1,0}$
3. **Decrease mesh to $h_k = (1/2)h_{k-1}$ and integrate all the range**
   => $S_{k,0}$, and calculate $S_{k,d}$ (d = 1, 2, ･･･, 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

$$S = \int_{-1}^{1} \exp(x)dx$$

Exact: exp(1) - exp(-1) = 2.3504023872876

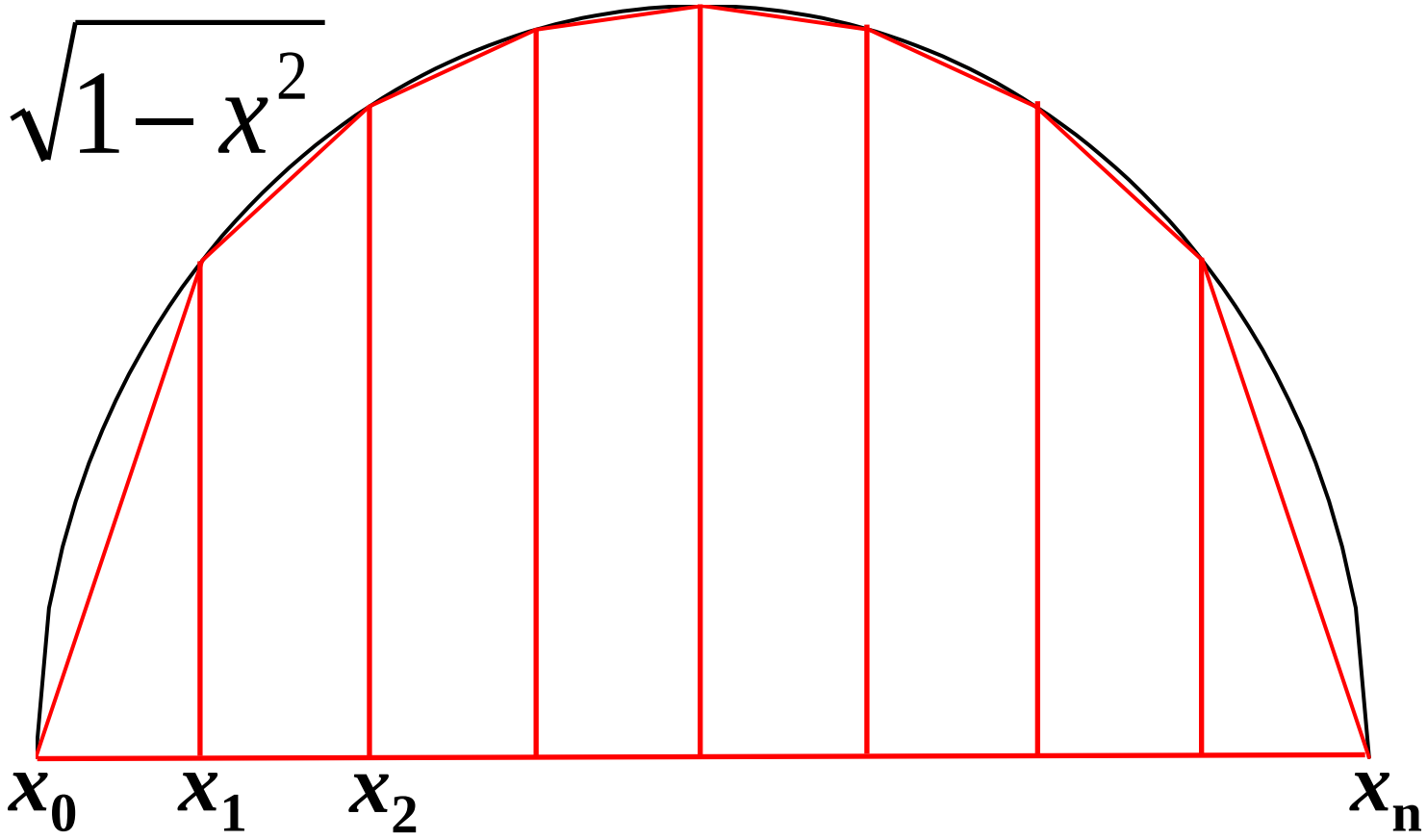
nDivide	Rieman	Trapezoid	Simpson	Simpson 3/8	Bode	Romberg	Cubic Spline	Order 3 Gauss-Legendre
1	1.61E+00	-7.36E-01				-7.36E-01		
2	9.83E-01	-1.93E-01	-1.17E-02			-1.17E-02		6.55E-05
3	6.97E-01	-8.64E-02		-5.25E-03				
4	5.39E-01	-4.88E-02	-7.92E-04		-6.85E-05	-6.85E-05	7.19E-03	1.13E-06
5	4.39E-01	-3.13E-02					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 singular 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 (変数変換型公式)

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 singular 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

1. Variable conversion x => u: $x = \varphi(u)$

2. Calculate by the Trapezoid rule

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

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 singular points at the ends and for infinite range

By variable conversion (変数変換)

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

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

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 = ϕ(k/n) and weights w_k = ϕ'(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

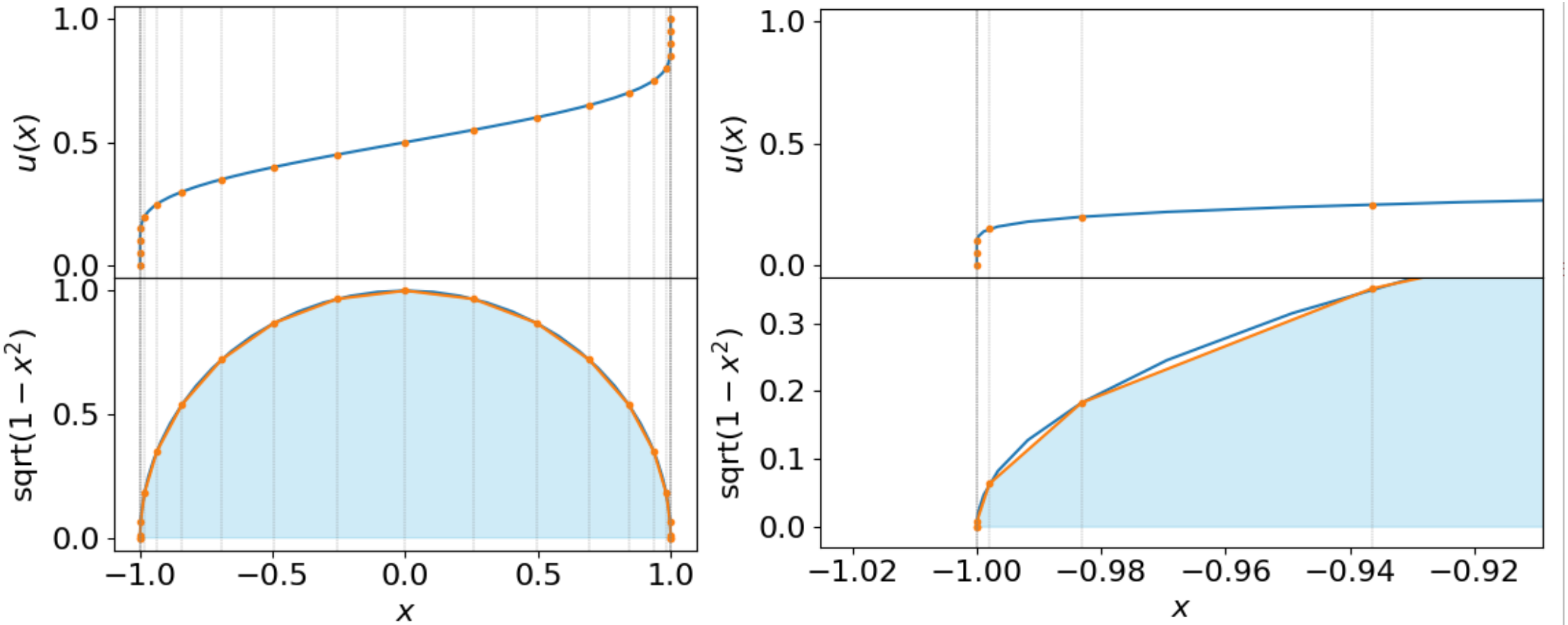
伊理･森口･高沢(IMT)の公式

> python integ_imt.py

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

Q = 0.00702985841

戸田英雄, 小野令美, 入門 数値計算, オーム社 (昭和58年)



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: $[-1,1]$

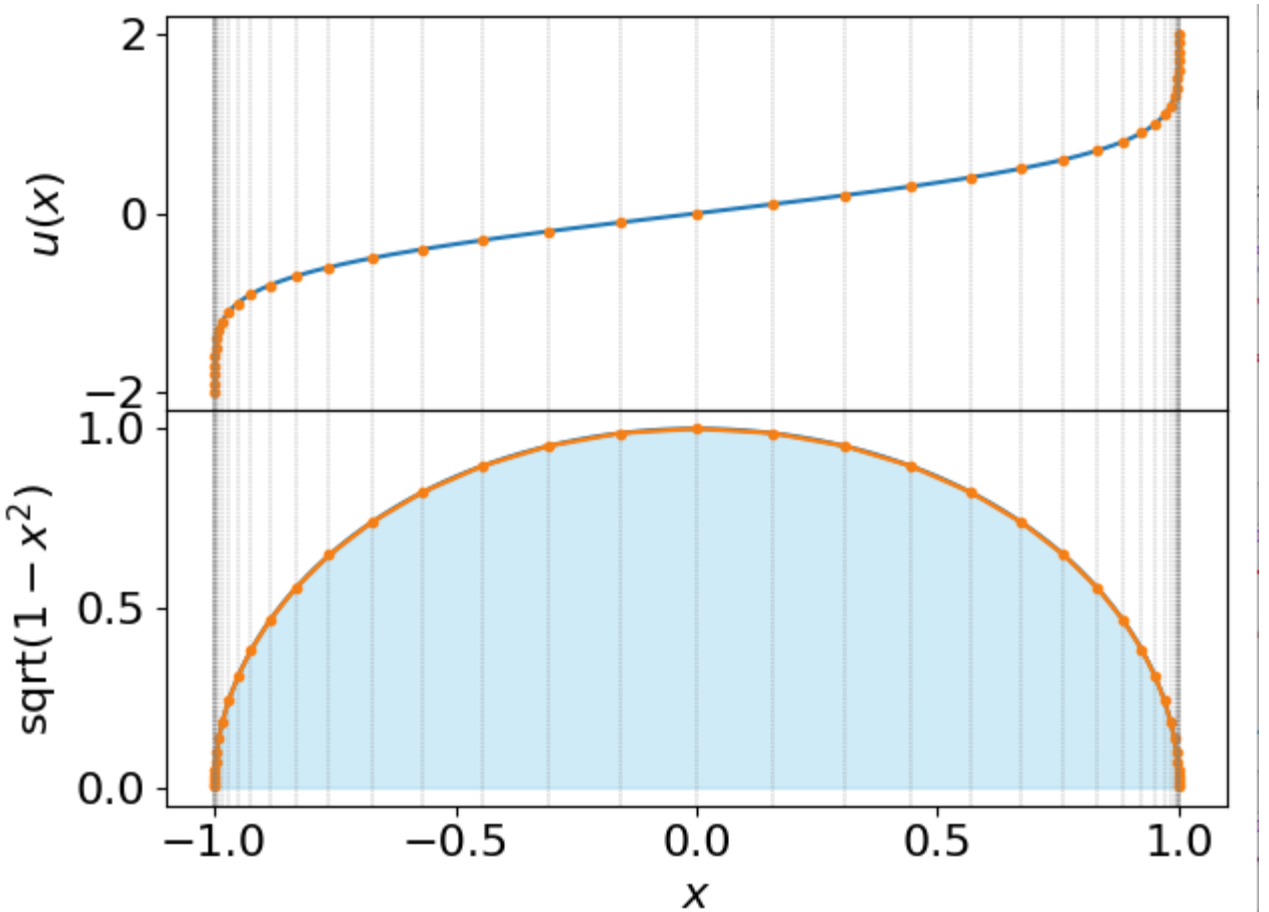
u range: $[-2,2]$ => x range ~ $[-1,1]$

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

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

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

> 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)

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: $[0, \infty]$

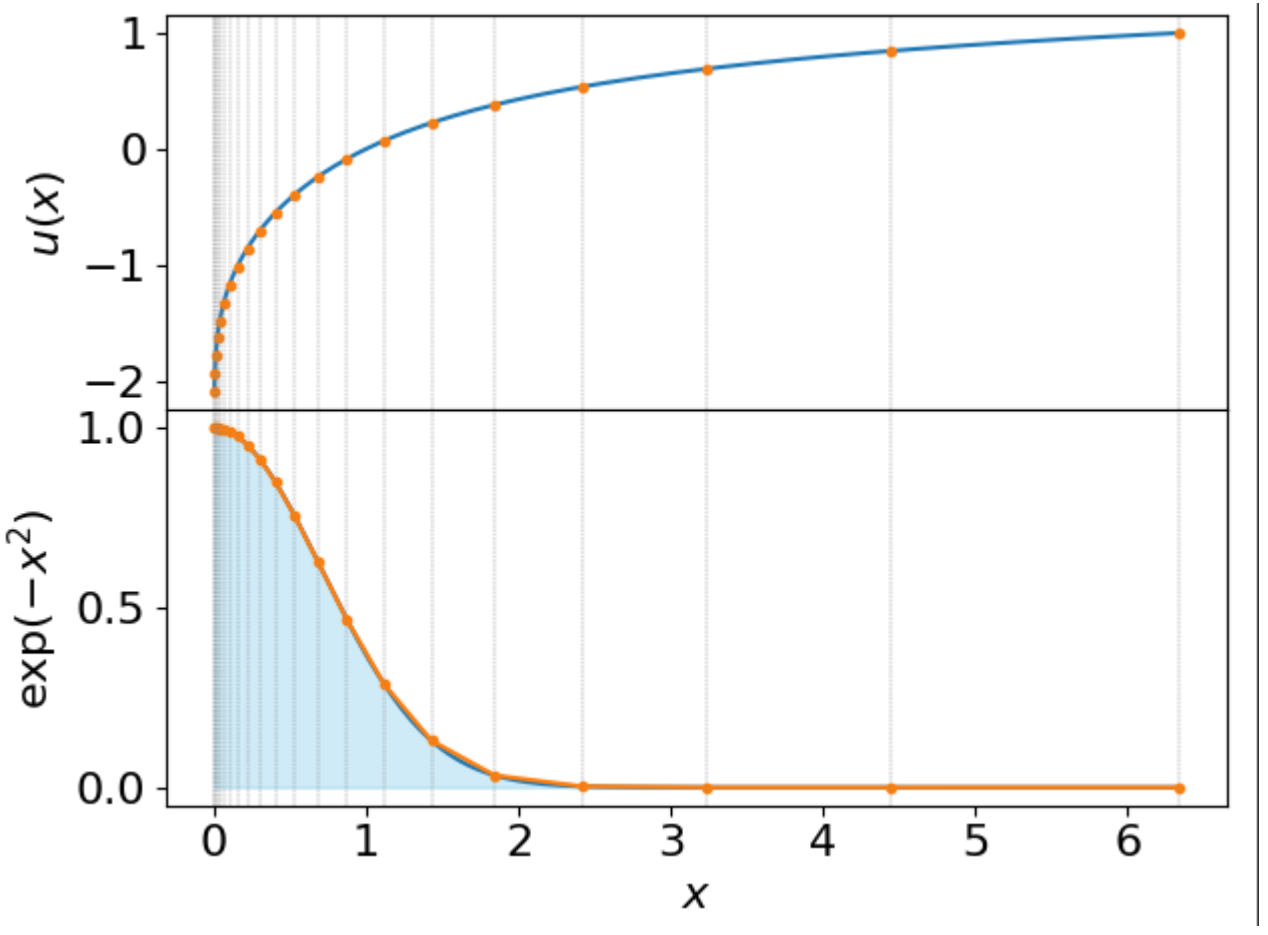
u range: $[-2,1]$ => x range ~ $[0,6]$

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

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

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

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

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

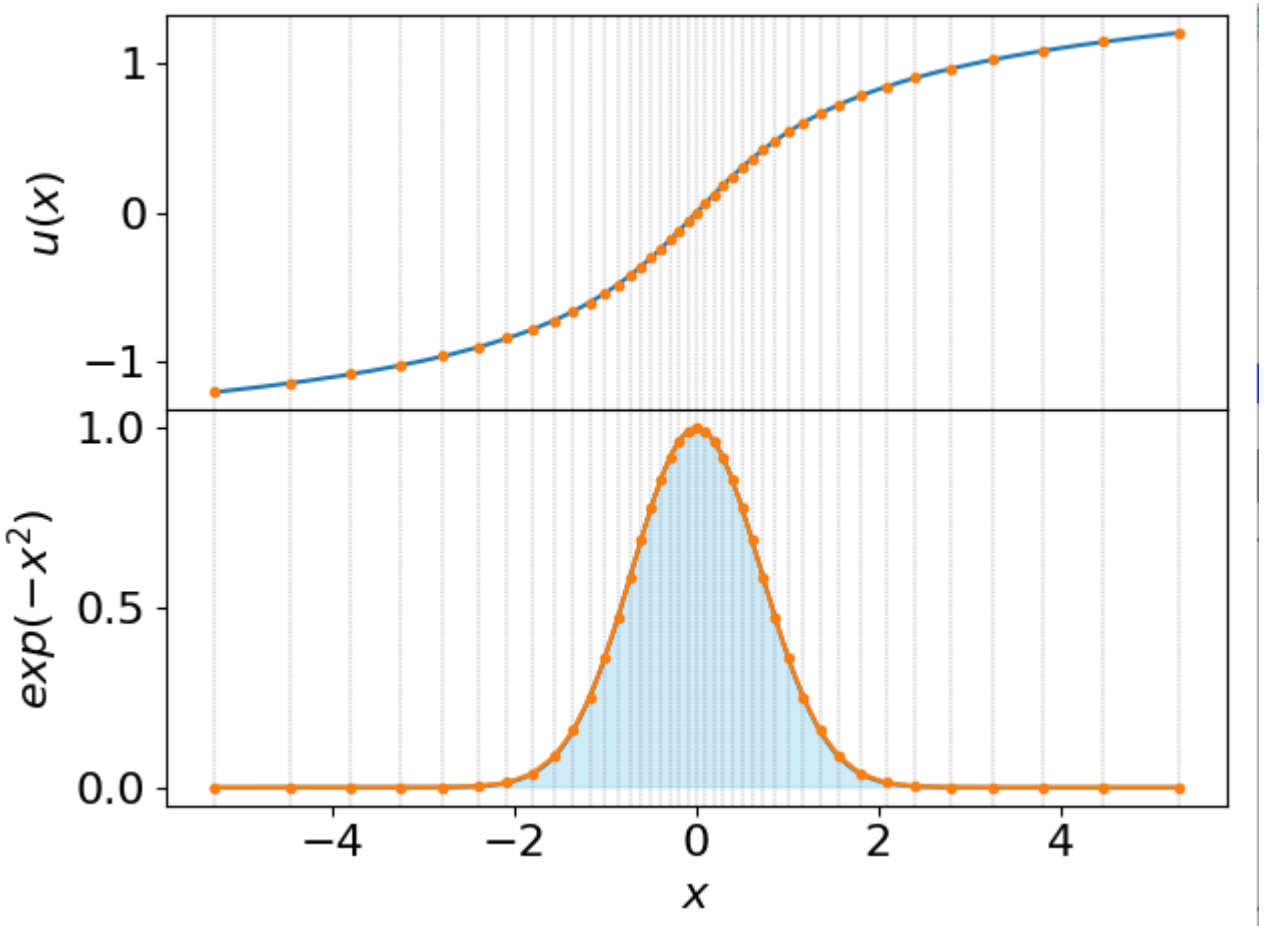
u range: $[-1.2,1.2]$ => x range ~ $[-5,5]$

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

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

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

> python integ_double_exp_inf_inf.py



Error for integration with singular 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)

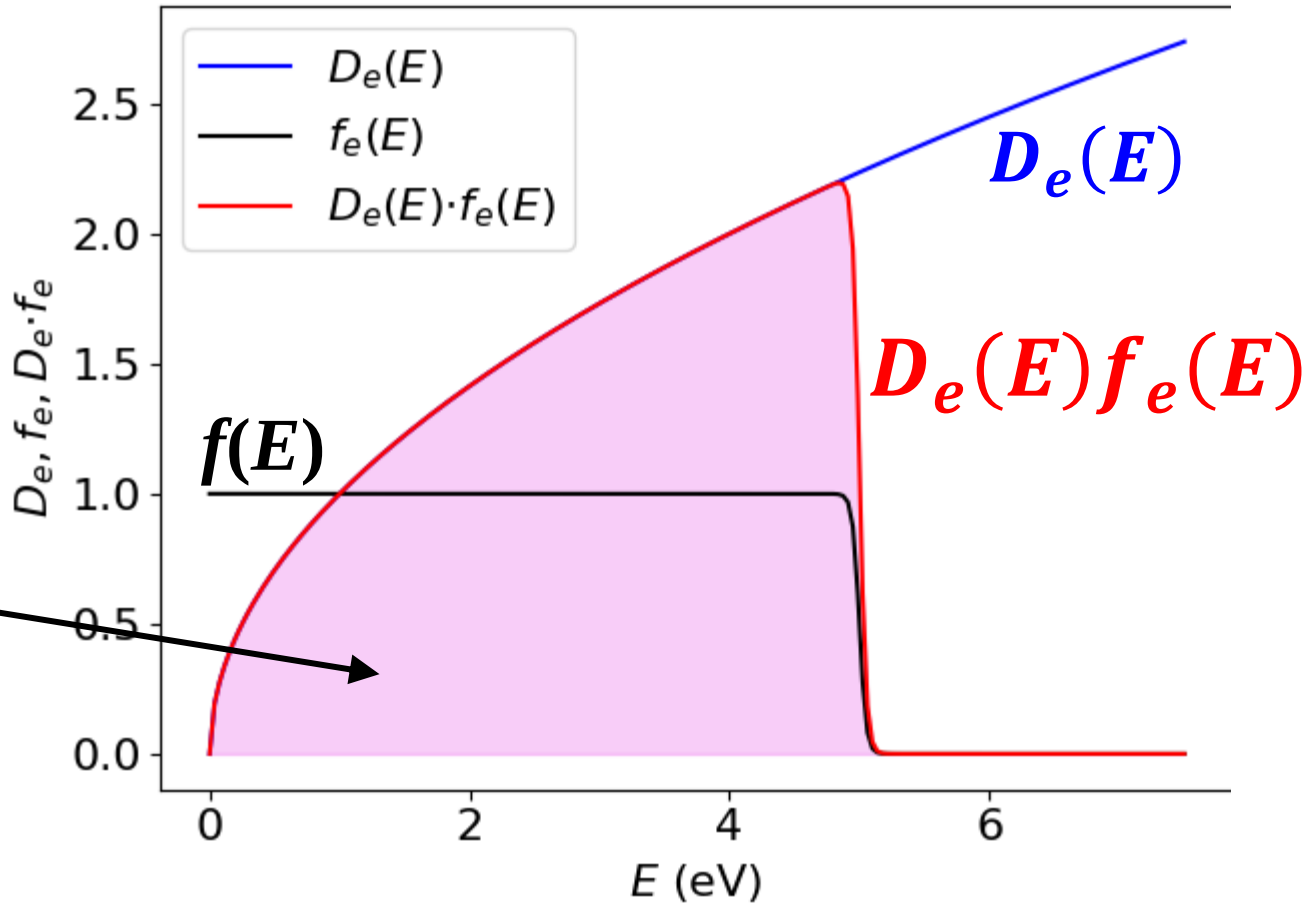
$$D_e(E) = D_{e0}\sqrt{E}$$

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

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 $D_e(E)\,f(E)$ 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 300 5.0

Temperature at 300K, $E_F = 5.0$ eVで

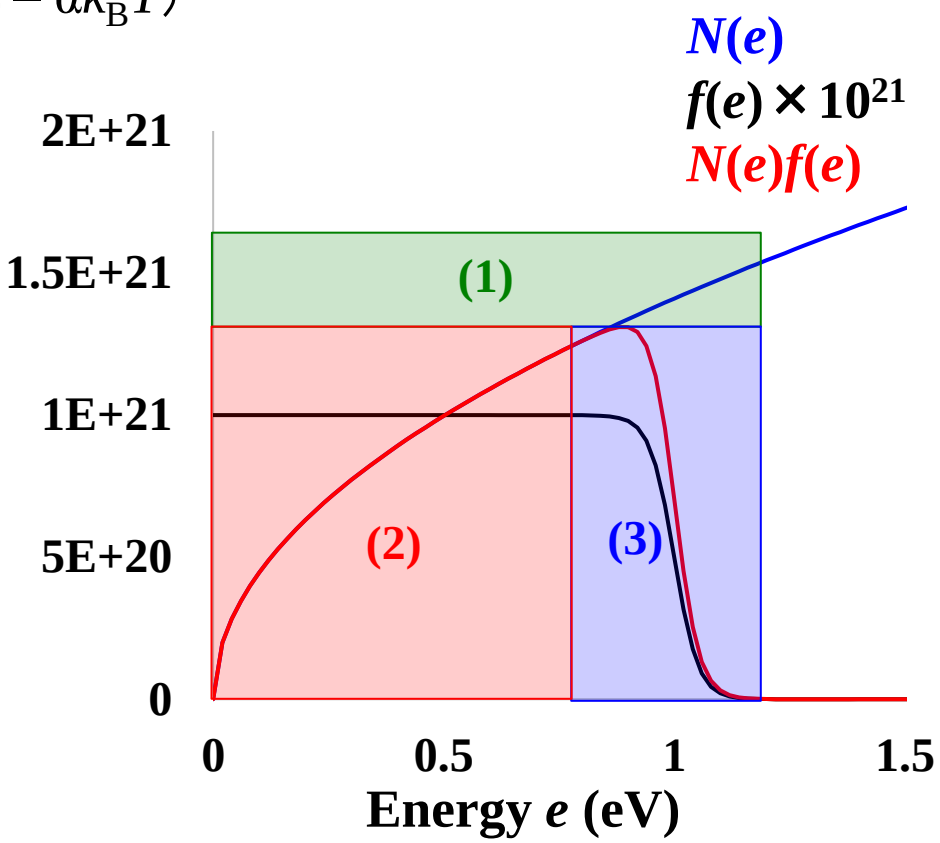
Measure time by repeating for 300 times

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 s
(2)	$0 \sim E_F - \alpha k_B T$	0.063 s
(3)	$E_F - \alpha k_B T \sim E_F + \alpha k_B T$	0.016 s

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**

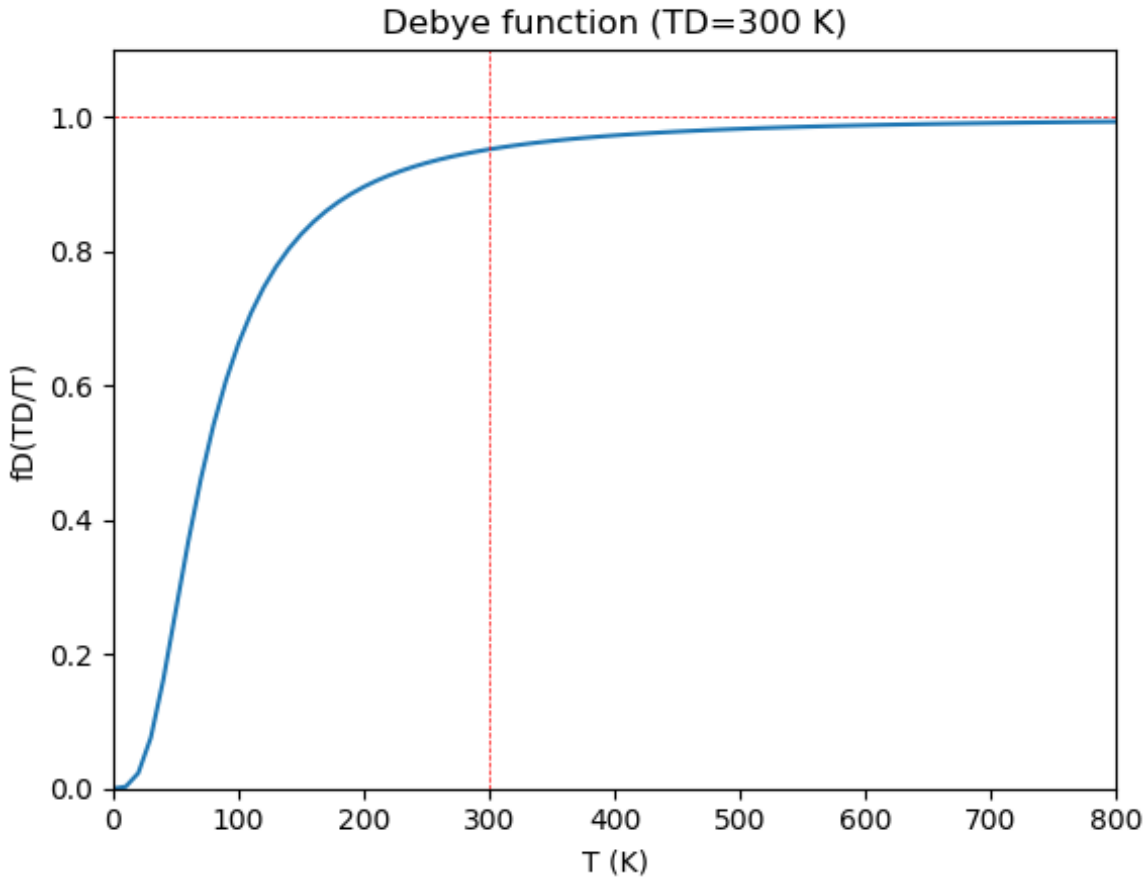
Uses scipy.integrate.quad, an adaptive quadrature routine based on QUADPACK.

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

python debye_function.py 300 0 800 10

Debye temperature 300 K

Temperate range 0 – 800 K, 10 K step



When actually writing a Python program...

- Recommend using scipy.integrate.quad() first.
  It strives to satisfy the requested accuracy and also provides an estimated error.
- If the integrand includes singularities, or if computational speed is important, compare it with other methods and decide the algorithm and computational conditions.

situation	Recommended algorithms
uniformly spaced experimental data	Trapezoid, Simpson, Spline
function can be evaluated at arbitral x	scipy.integrate.quad, Gauss–Legendre、Romberg
smooth integral over a finite interval	scipy.integrate.quad, Gauss–Legendre、Romberg
has endpoint singularities	Variable conversion、IMT、Double exponential
integral over an infinite interval	Variable conversion、Double exponential, Adaptive integration
exhibits a sharp transition like the Fermi function	Split integration interval, use fine mesh

Actual problems may be split to several integration intervals and choose an optimal algorithm for each subinterval.
For example, for integrals such as the product of the Fermi-Dirac distribution function and the free-electron-approximation density of states, use a fast Fermi-integral library. Refer to the tutorial material "Vibeコーディング (難易度★★★★): 半導体のキャリア密度の温度依存性のフィッティング (非線形最小二乗法)" at http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/?page=tutorial&id=modify.

実際にpythonプログラムを作る場合は・・・

まずは、scipy.integrate.quad() を使うことを推奨

- 与えられた精度を満足するように努力する・推定精度も得られる
- 特異点を含む場合、計算速度が重要な場合は、他の方法と比較してアルゴリズム・計算条件を決定する

状況	使いやすい方法
等間隔の実験データ	台形則、Simpson則、スプライン積分
関数を自由に評価できる	scipy.integrate.quad, Gauss–Legendre、Romberg
滑らかな有限区間積分	scipy.integrate.quad, Gauss–Legendre、Romberg
端点特異点がある	変数変換、IMT、二重指数関数型公式
無限区間積分	変数変換、二重指数関数型公式、適応積分
Fermi関数のように急峻な変化がある	積分範囲を分割、細かいメッシュを必要部分だけ使う

実際の問題では、積分区間分割、各区分での最適アルゴリズムの選択、などを行う必要がある。
例えば、Fermi-Dirac分布関数 × 自由電子近似状態密度の積分 Fermi積分の高速ライブラリなど。
チュートリアル資料「Vibeコーディング (難易度★★★★): 半導体のキャリア密度の温度依存性のフィッティング (非線形最小二乗法)」 http://d2mate.mdxes.iir.isct.ac.jp/D2MatE/?page=tutorial&id=modify を参考

Reliable Numerical Integration Function: integrate.quad()

Usage:

Integral value, estimated error = quad(f(x), lower limit, upper limit,
epsrel=relative error tolerance)

Features

- Error is estimated.
- quad() **tries to satisfy the required accuracy specified by epsrel**.
- If the integration range is given as np.inf, **an algorithm suitable for infinite-range integration is used**.
- If the integration range or error tolerance is not chosen carefully, the calculation may take longer than expected.

Algorithm

- The integral is first calculated using the n-point Gauss-Legendre method.
  In the Gauss-Legendre method, the integration points and weights are chosen so as to maximize the accuracy.
- Then, n+1 additional points are added and the integral is calculated again using the Kronrod extension.
  - The original n Gauss-Legendre integration points are kept unchanged.
  - The additional n+1 integration points and weights are chosen so as to maximize the accuracy.
- The error is estimated from the difference between the Gauss-Legendre result and the Kronrod-extended result.
  If the estimated error becomes smaller than epsrel, the integral value and the estimated error are returned.

精度で安心できる数値積分関数: integrate.quad()

Usage:

積分値、推定誤差 = quad(f(x), 積分下限, 積分上限, epsrel=相対誤差の許容値)

f(x):被積分関数 (x以外の引数は args引数で渡すか、
lambda関数でxだけの引数に変換する)

特徴

- 誤差推定される
- 要求精度 epsrel を守るように努力する
- 積分範囲を np.inf で与えると、無限区間積分に適したアルゴリズムを使う
- 積分範囲や誤差の許容値などに注意しないと、かえって計算時間がかかることがある

アルゴリズム

- n点区分Gauss-Legendre法で積分値を計算
  GL法: 精度が最大になるように、積分点と重みを決定
- n+1点を追加して積分値を計算 (Kronrod拡張)
  ・GL法のn点の積分点はそのまま維持
  ・精度が最大になるように、追加したn+1点の積分点と重みを決定
- GL法とKronrod拡張の積分値の差から誤差を推定し、epsrel以下になったら、積分値と誤差を返す