Lecture 2-1
Numerical Differentiation and Integration

Min Sup Hur

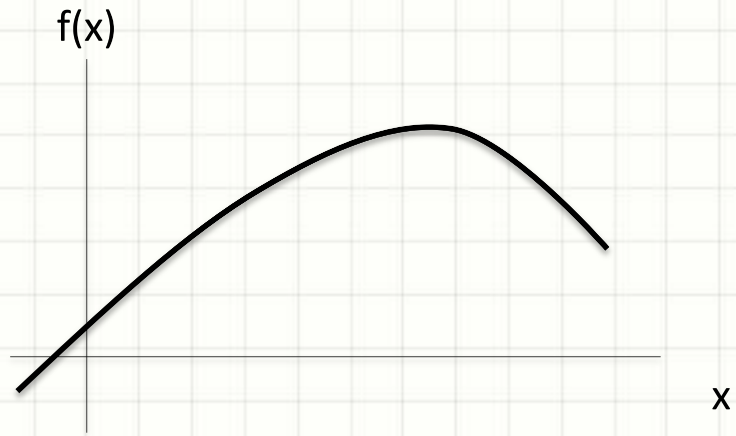
Mar. 7 2023

Outline

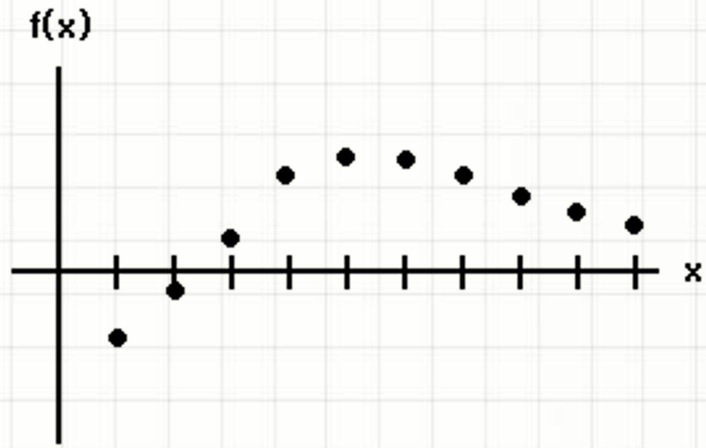
- Differentiation of Functions
- Differentiation of Data
- Integration of Functions
- Integration of Data

Function and Data

- Let's roughly distinguish the *function* and *data* by somethings that are *continuous* and *discretized*.



function



data

How to Find the Derivative Numerically

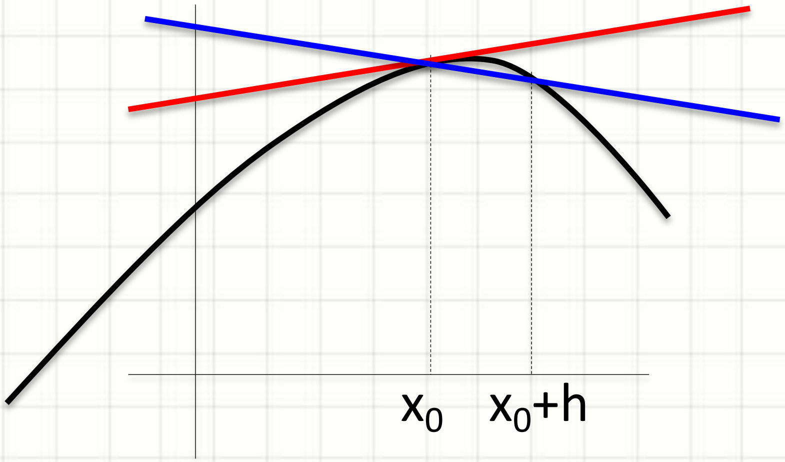
- $\lim_{h\to\infty} \frac{f(x_0+h)-f(x_0)}{h} = f'(x_0) \quad \rightarrow \qquad \frac{f(x_0+h)-f(x_0)}{h} \simeq f'(x_0)$
- Small h approximates the '$h \to 0$'
- Taylor expansion is used to estimate the error.

1st order method

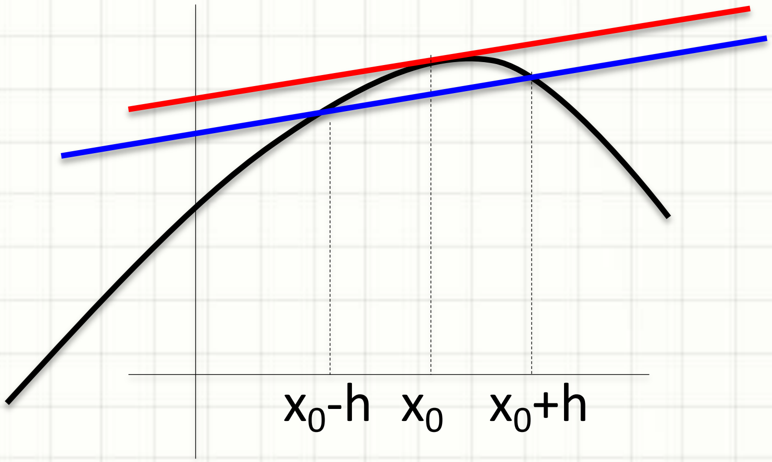
$$\frac{f(x_0+h)-f(x_0)}{h} = f'(x_0) + \frac{1}{2} f''(x_0)h + \cdots = f'(x_0) + O(h)$$

2nd order method

$$\frac{f(x_0+h)-f(x_0-h)}{2h} = f'(x_0) + \frac{1}{6} f'''(x_0)h^2 + \cdots = f'(x_0) + O(h^2)$$



2-point method



3-point method

Differentiation of a Function

- The differential coefficient of a function can be found using derivative() of scipy.misc package.

Example 1. Find the derivative of sin(x) at x=π/3

Answer: >>> import scipy.misc as sm; from numpy import *

>>> sm.derivative(sin,pi/3.0,dx=0.1)

function   x   spacing

Example 2. Find the second derivative of $x^2e^{-x^2}$ at $x = 1.0$. Do this with dx=0.2. Compare the results of 3 and 5 points-calculations and the exact value, -1.47152

Answer:

```
>>> import scipy.misc as sm; from numpy import *
>>> def f(x):
 . . .        return x**2*exp(-x**2)
>>> sm.derivative(f, 1.0, dx=0.2, n=2, order=3)
>>> sm.derivative(f, 1.0, dx=0.2, n=2, order=5)
```

Derivative order   Accuracy order (num. points used)

Derivative on a Data Set

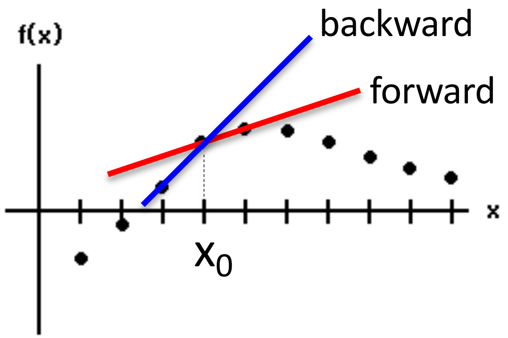
- The same as the function case, but h is fixed by the data spacing.

2-point forward method

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} + O(h)$$

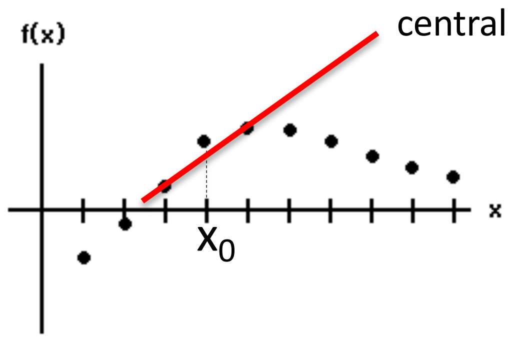
2-point backward method

$$f'(x_0) = \frac{f(x_0) - f(x_0-h)}{h} + O(h)$$



3-point, central-valued method

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(h^2)$$



Higher Accuracy Derivatives

- As more number of points are used in the calculation, you get more accurate results of the derivative.
- Taylor expansion can be used to determine the coefficient of each point.

- Let's write

$$f'(x_0) = \overbrace{\underbrace{af(x_0) + bf(x_0+h) + cf(x_0-h)}_{\text{3-point central valued method}} + df(x_0+2h) + ef(x_0-2h)}^{\text{5-point central valued method}} + \cdots$$

- Expand each term

$$f(x_0) = f(x_0)$$

$$f(x_0 \pm h) = f(x_0) \pm hf'(x_0) + \frac{h^2}{2}f''(x_0) \pm \frac{h^3}{6}f'''(x_0) + \cdots$$

$$f(x_0 \pm 2h) = f(x_0) \pm 2hf'(x_0) + 2h^2f''(x_0) \pm \frac{4h^3}{3}f'''(x_0) + \cdots$$

$$f(x_0 \pm 3h) = f(x_0) \pm 3hf'(x_0) + \frac{9}{2}h^2f''(x_0) \pm \frac{9h^3}{2}f'''(x_0) + \cdots$$

$$\vdots$$

Higher Accuracy Derivatives

- To eliminate $f(x_0), a = 0, b = -c, d = -e, \cdots$ and so on. From this

$$f'(x_0) = (2b + 4d + \cdots)hf'(x_0) + \left(\frac{b}{3} + \frac{8d}{3} + \cdots\right)h^3 f'''(x_0) + O(h^5)$$

- If we set $b = \frac{1}{2h}$ and $d = 0$ and all the zero higher's, we get the 3-point method.

$$f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h} + O(h^2)$$

- Set $2b + 4d = \frac{1}{h}, \frac{b}{3} + \frac{8d}{3} = 0$, and all the zero higher's, then we get the 4th order error,

$$f'(x_0) = \frac{2\big(f(x_0+h) - f(x_0-h)\big)}{3h} - \frac{\big(f(x_0+2h) - f(x_0-2h)\big)}{12h} + O(h^4)$$

which is the 5-point method.

- You can extend the same technique up to any order of the error.

Importance of the Error Order

- 1^{st} order with h=0.01 yields the same accuracy as the 2^{nd} order with h=0.1.
- Then why is the higher order of the error important, while we have only to reduce h to get the same accuracy from the lower order method?
- From the rationale described below, we find that it is important to use at least the second order or higher.

- To solve a differential equation with step *h* to the time t, you calculate the derivative as many times as

$$N = \frac{t}{h}$$

- When N-times iterated, the accumulated error of the method of order n is

$$Error \sim Nh^n = th^{n-1}$$

- You need to have at least $n \geq 2$ to reduce the accumulated error by decreasing *h*.

The Second Derivative

- To get the 2nd derivative, repeat twice the 1st derivative

- 1st derivatives

$$f'\left(x_0+\frac{h}{2}\right)=\frac{[f(x_0+h)-f(x_0)]}{h}+O(h^2)$$

$$f'\left(x_0-\frac{h}{2}\right)=\frac{[f(x_0)-f(x_0-h)]}{h}+O(h^2)$$

- 1st derivative of the 1st derivative

$$f''(x_0)=\frac{[f'(x_0+h/2)-f'(x_0-h/2)]}{h}+O(h^2)$$

- Finally, the central-valued 2nd derivative with the error of the 2nd order is

$$f''(x_0)=\frac{f(x_0+h)-2f(x_0)+f(x_0-h)}{h^2}+O(h^2)$$

Differentiation of Data

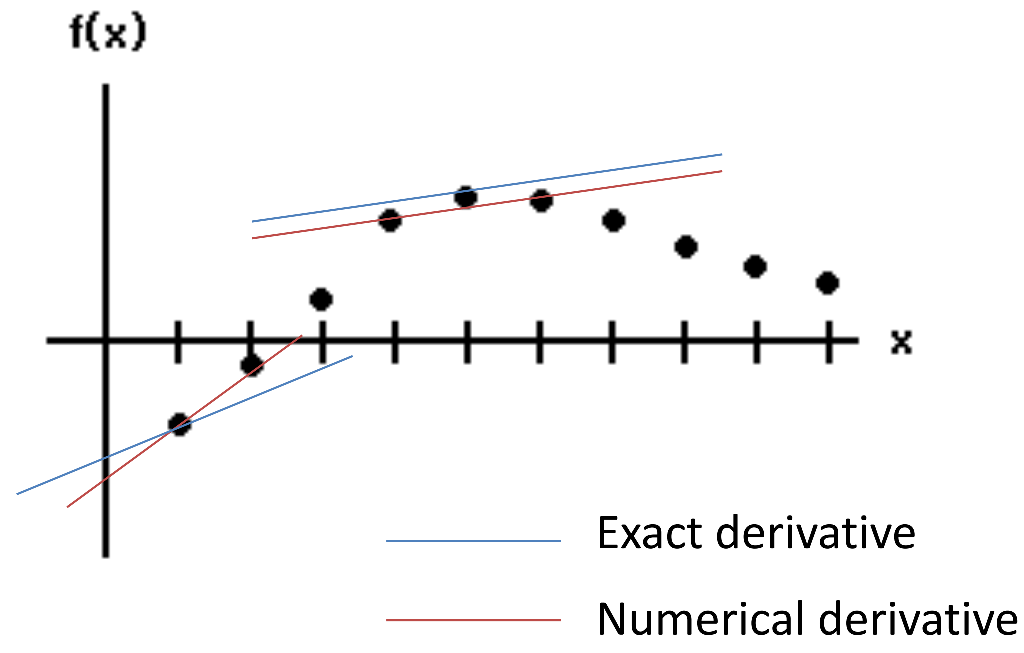
- The data can be differentiated by gradient() of numpy package.
- It returns the list of derivatives at all the data points.

Example 3. Find derivatives of a list y=[1,2,4, 8,16,32,64,128,256,512], with the spacing between the data points 0.1. Plot y and its derivative.

Answer:
```
>>> from numpy import *
>>> y=[1,2,4,8,16,32,64,128,256,512]
>>> z=gradient(y, 0.1);  print(z)
```

Derivatives at the Edge of Data

- At the edges of data, the high-order central-valued scheme is not available.
- Forward (left edge) or backward (right edge) method can be used, but the accuracy is low.



Derivatives at the Edge of Data

Example 4. Repeat three time successively the differentiation of discrete data $[\exp(x_n)]$, where x_n=0.1n with n going from 0 through 20. Plot the results and see how fast the error of the edge-derivative grows.

Answer:
```
>>> from numpy import *; import matplotlib.pyplot as plt
>>> x=arange(0,2, 0.1); y=exp(x)
>>> z=gradient(y, 0.1);  z2=gradient(z, 0.1);  z3=gradient(z2, 0.1)
>>> plt.plot(x,y,x,z,x,z2,x,z3);  plt.show()
```

Example 5. Do the Example 4 but this time, with a smaller spacing, i.e. 0.02.

Answer:
```
>>> import numpy as np; import matplotlib.pyplot as plt
>>> x=np.arange(0,2, 0.02); y=np.exp(x)
>>> z=np.gradient(y, 0.02);  z2=np.gradient(z, 0.02)
>>> z3=np.gradient(z2, 0.02)
>>> plt.plot(x,y,x,z,x,z2,x,z3);  plt.show()
```

The 2nd Order Derivatives at the Edge

- Extrapolate an outward point, using the quadratic polynomial determined by three inward contiguous points.
- Apply the central-valued scheme using the extra point.

The quadratic polynomial connecting f_0, f_1, and f_2 is

$$f(x) = \frac{f_0 - 2f_1 + f_2}{2h^2}x^2 + \frac{-3f_0 + 4f_1 - f_2}{2h}x + f_0$$

f_{-1} at x=-h is

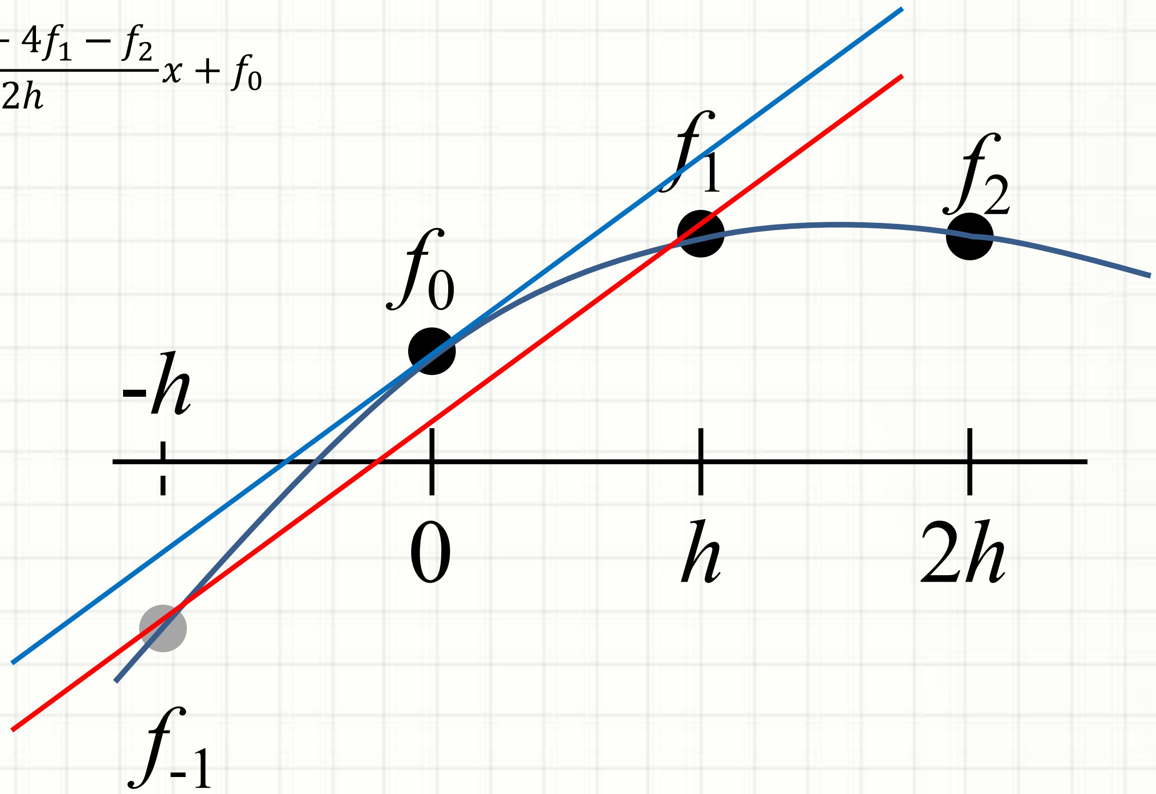
$$f_{-1} = 3f_0 - 3f_1 + f_2$$

The derivative at x=0 (edge) is,

$$f_0' = \frac{f_1 - f_{-1}}{2h} = \frac{1}{2h}(-3f_0 + 4f_1 - f_2)$$

On the other edge (N-1),

$$f_{N-1}' = \frac{1}{2h}(f_{N-3} - 4f_{N-2} + 3f_{N-1})$$



The 2nd Order Derivatives at the Edge

- The order of the error of the previous equation can be checked by Taylor expansion.
- Is it stable?

$$\frac{1}{2h}(-3f_0 + 4f_1 - f_2) = \frac{1}{2h}\left[-3f_0 + 4\left(f_0 + hf_0' + \frac{h^2}{2}f_0'' + \cdots\right) - (f_0 + 2hf_0' + 2h^2 f_0'' + \cdots)\right]$$

$$= f_0' + O(h^2) \quad \leftarrow \text{2nd order}$$

Example 6. Do the same job as in Example 4, but with the edge order two this time. You put ‘edge_order=2’ as a third argument of gradient().

Answer:
```
>>> from numpy import *; import matplotlib.pyplot as plt
>>> x=arange(0,2, 0.1); y=exp(x)
>>> z=gradient(y, 0.1,edge_order=2)
>>> z2=gradient(z,0.1,edge_order=2)
>>> z3=gradient(z2,0.1,edge_order=2)
>>> plt.plot(x, y, x, z, x, z2, x, z3);  plt.show()
```

Numerical Integration by Quadrature

- ‘quadrature’ means the process of constructing a square with an area equal to that of a figure bounded by a curve.

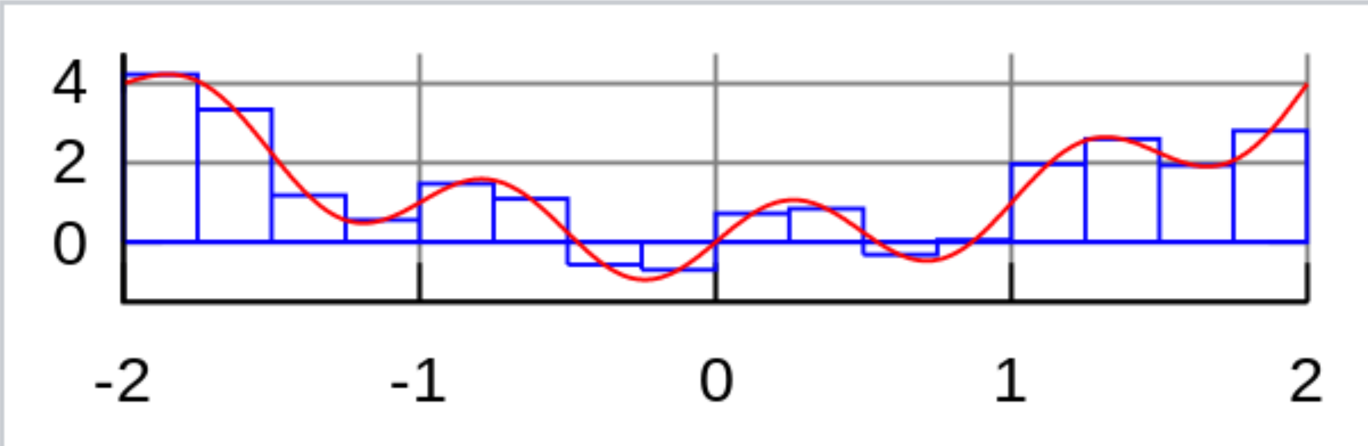


Illustration of the rectangle rule.

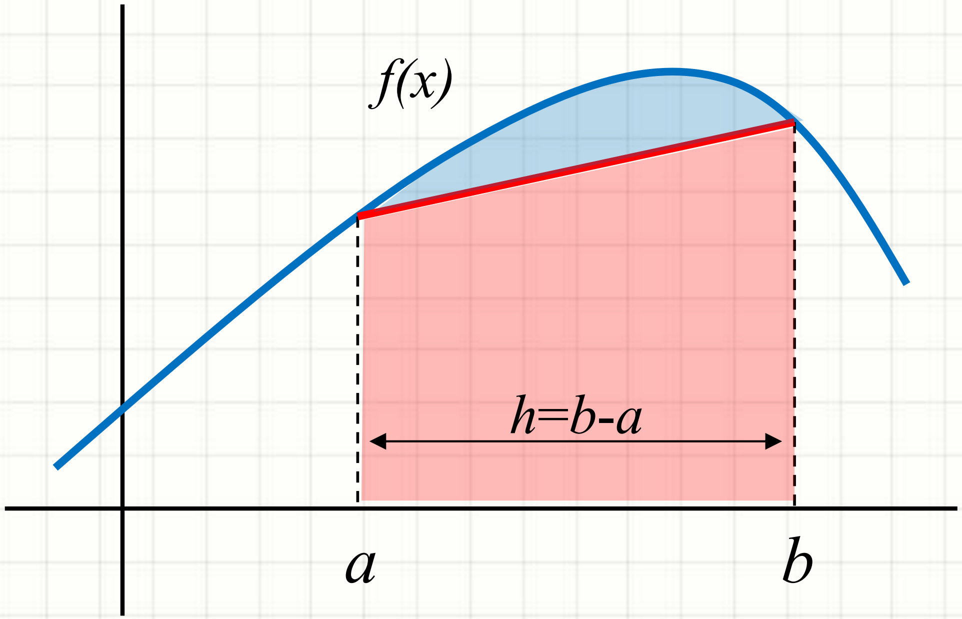
Rectangular Quadrature (Figure is from Wikipedia)

Evenly and Non-evenly Spaced Quadrature Rules

- Trapezoid or Simpson's rules are the examples of the 'Newton-Cotes' quadrature rules, which are a group of formulae based on evaluating the integrand at equally spaced nodes. There are 'open' and 'closed' Newton-Cotes rules, depending on whether the end points are included or not.
- There are advanced schemes of using non-evenly spaced evaluation points of the integrand, for higher order accuracy. We briefly outline the Gaussian quadrature and the Clenshaw-Curtis Quadrature. Not suitable for data integration, where the spacings are fixed.

Newton-Cotes Quadrature Rules and corresponding Python Functions	Non-evenly Spaced Quadrature Rules and corresponding Python Functions
Mid-point rule Trapezoid rule → trapz() for data. Simpson's rule → simps() for data. Romberg Integral → romb() for data or romberg() for functions. etc.	Gaussian quadrature → quadrature() Clenshaw-Curtis quadrature → quad() etc. * Not suitable for data integration

Trapezoid Quadrature Rule



$$\int_a^b f(x)dx = \frac{h}{2}[f(a) + f(b)] + O(f''h^3)$$

Error of the Trapezoid Rule

- Define a function $g(t)$ as follows. $g(t) = \int_a^{a+t} f(x)dx - \frac{t}{2}[f(a) + f(a+t)]$

  This is the difference between the exact integral and the trapezoidal rule, i.e. the error of the quadrature.

- Get the 2nd derivative of $g(t)$. $g'(t) = \frac{1}{2}f(a+t) - \frac{1}{2}f(a) - \frac{t}{2}f'(a+t)$

$$g''(t) = -\frac{t}{2}f''(a+t)$$

- For a certain ξ in $a \le \xi \le a+t$, $|f''(a+t)| \le |f''(\xi)|$, which leads to

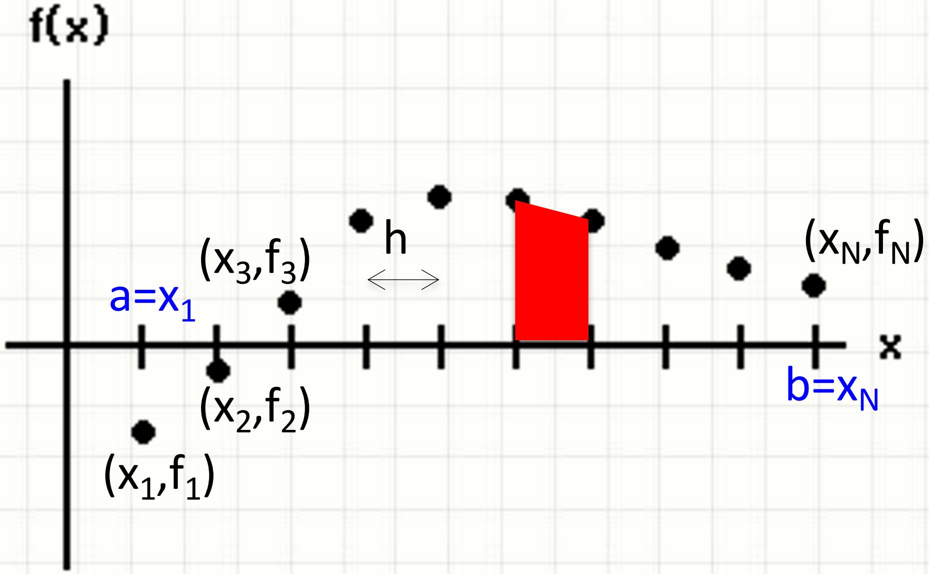
$$-\frac{t}{2}|f''(\xi)| \le g''(t) \le \frac{t}{2}|f''(\xi)|$$

- Integrate it twice over $a \le t \le a+t$. Using $g'(0) = g(0) = 0$,

$$-\frac{t^3}{12}|f''(\xi)| \le g(t) \le \frac{t^3}{12}|f''(\xi)|$$

- With substitution $t = h$, the error $g(h)$ is proportional to $f''h^3$

Composite Trapezoid Rule

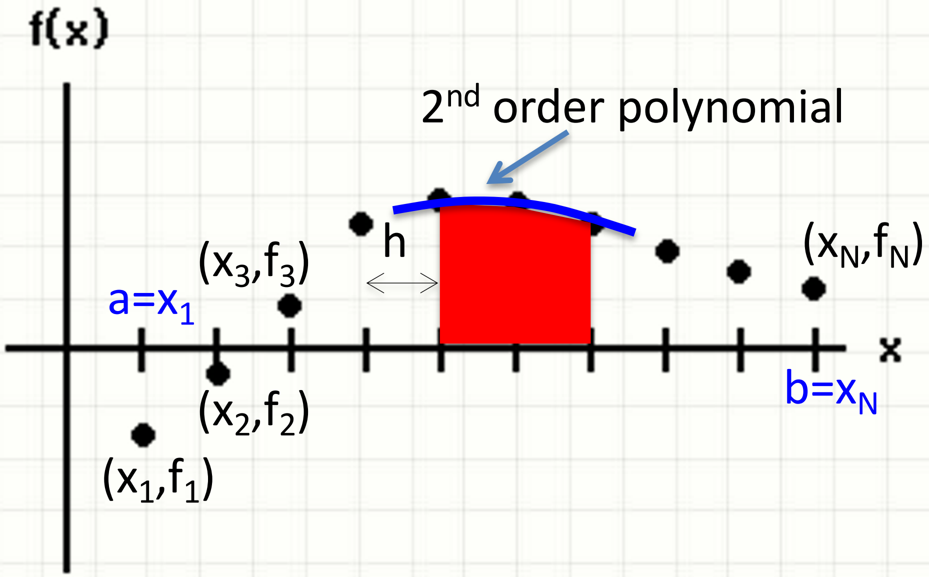


- Composite Trapezoid

$$\int_a^b f(x)dx = \frac{h_1}{2}[f_1 + f_2] + \frac{h_2}{2}[f_2 + f_3] + \cdots + \frac{h_{N-1}}{2}[f_{N-1} + f_N] + O\left(\sum_1^{N-1} h_i^3 f_i''\right)$$

$$\int_a^b f(x)dx = h\left[\frac{1}{2}f_1 + f_2 + \cdots + f_{N-1} + \frac{1}{2}f_N\right] + O\left(\frac{(b-a)^3}{N^2}f''\right)$$

Simpson's Quadrature Rule



Simpson's Rule

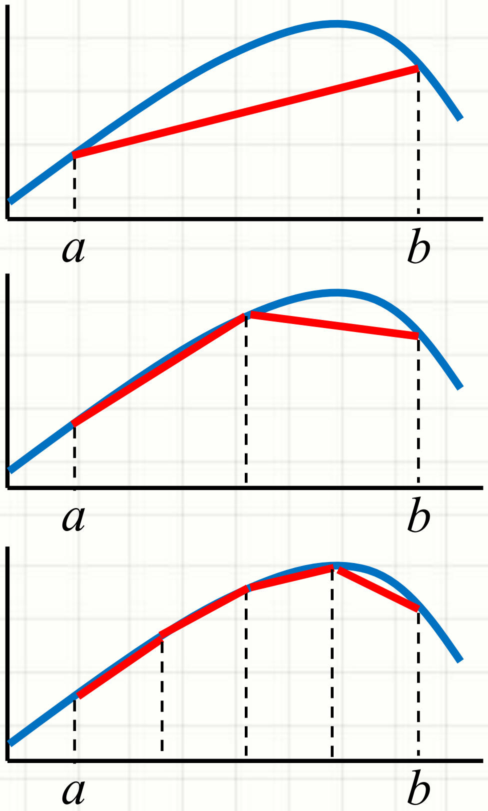
$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx = h\left[\frac{f_{i-1}}{3} + \frac{4f_i}{3} + \frac{f_{i+1}}{3}\right] + O\left(h^5 f^{(4)}\right)$$

Composite Simpson's Rule

$$\int_{x_1}^{x_N} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \cdots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right] + O\left(\frac{1}{N^4}\right)$$

Romberg Integral

- Examine the convergence of the quadrature as doubling the number of points where the integrand is evaluated.
- Accelerate the convergence using the Richardson extrapolation technique.



$$R(1,1) = \frac{b-a}{2^0}\left(\frac{f_a}{2}+\frac{f_b}{2}\right) + O(f''h_1^2)$$

$$R(2,1) = \frac{b-a}{2^1}\left(\frac{f_a}{2}+\cdots+\frac{f_b}{2}\right) + O(f''h_2^2)$$

$$R(3,1) = \frac{b-a}{2^2}\left(\frac{f_a}{2}+\cdots+\frac{f_b}{2}\right) + O(f''h_3^2)$$

$$R(n,1) = \frac{b-a}{2^{n-1}}\left(\frac{f_a}{2}+\cdots+\frac{f_b}{2}\right) + O(f''h_n^2)$$

Richardson Extrapolation

$$R(2,2) = \frac{2^2R(2,1)-R(1,1)}{2^2-1} + O(h_1^4)$$

$$R(3,2) = \frac{2^2R(3,1)-R(2,1)}{2^2-1} + O(h_2^4)$$

$$R(n,2) = \frac{2^2R(n,1)-R(n-1,1)}{2^2-1} + O(h_n^4)$$

Gaussian Quadrature

- The n-nodes and n-weights are determined from the roots of Legendre polynomials and its derivative.
- Those numbers are already well tabulated.

- Any definite integral over a finite range can be rescaled to the integral from -1 to 1. Represent the integral by the sum of values-weight products.

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i f(x_i)$$

- $x_1, x_2, \cdots, x_n$ is determined from the roots of the Legendre Polynomial $P_n(x)$

- The weight is calculated from

$$w_i = \frac{2}{(1 - x_i^2)[P_n'(x_i)]^2}$$

- Usually converges faster than the equally-spaced case.

Clenshaw-Curtis Quadrature

- The integrand is expanded by the Chebyshev polynomials.
- The coefficients of the polynomials are easily obtained from discrete cosine transform.
- The integral of each polynomial is tabulated. Once the coefficients are obtained, the integral is straightforward.

- As in the Gaussian Quadrature, the integral range is rescaled to [-1,1].
- Chebyshev polynomial: $T_k(\cos\theta) = \cos(k\theta)$
- Expansion of the integrand.

$$f(x) = \frac{a_0}{2} T_0(x) + \sum_{k=1}^{\infty} a_k T_k(x) \quad \rightarrow \quad f(\cos\theta) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos(k\theta)$$

- Integration

$$\int_{-1}^{1} f(x)dx = \int_{0}^{\pi} f(\cos\theta)\sin\theta d\theta = a_0 + \sum_{k=1}^{\infty} \frac{2a_{2k}}{1-(2k)^2}$$

Integration of Functions by quad()

Example 7. Calculate $\int_0^4 x^2 dx$

Answer: >>> from scipy import integrate

>>> x2=lambda x: x**2   # define an in-line function

>>> integrate.quad(x2,0,4)

>>> (21.333333333333333, 2.3684757857001e-13)

integration result,   upper bound of the error

Example 8. Calculate $\int_0^\infty x^2 e^{-\alpha x^2} dx$ for $\alpha = 2$

extra arguments to pass to function
optional tuple
Ex) $\alpha = 2$ is extra args -> args = (2,)

Answer: >>> import scipy.integrate as intg

>>> import numpy as np

>>> f=lambda x, a: np.exp(-a*x**2)*x**2

>>> intg.quad(f, 0, np.inf, args=(2,))

Clenshaw-Curtis quadrature → quad(function, lower limit, upper limit, args)

exp(-a*x**2)*x**2

Double Integration of Functions

- For double integration of a function, use

dblquad(f, a, b, g, h, args=(), ...)

- The y-limits, i.e. g and h, are functions of x.
- For triple integration, use tplquad(f, a, b, g, h, p, q, args=(), ...)

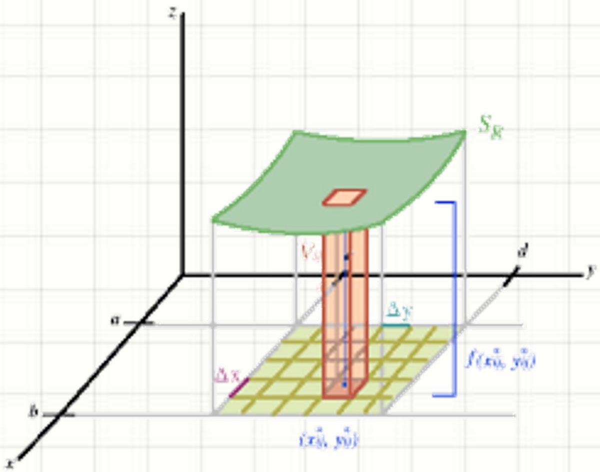


Figure is from Mathonline.

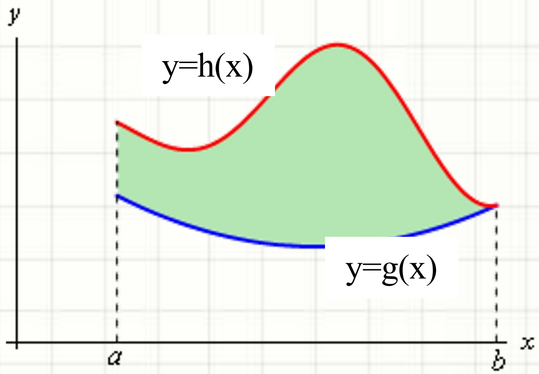
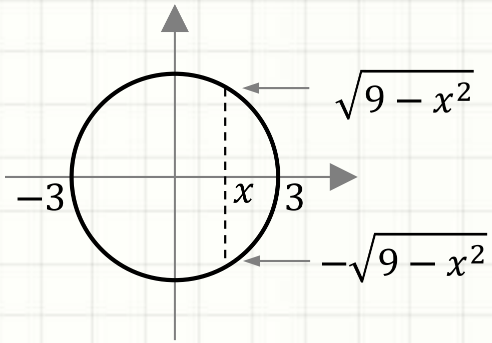


Figure is from tutorial.math.lamar.edu

Double Integration of Functions

Example 9. Integrate $e^{-x^2-y^2}$ over a circle centered at the origin and with radius 3.

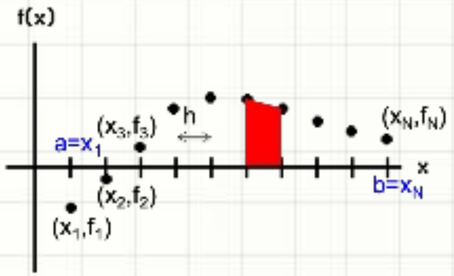
Answer:
```
>>> import numpy as np
>>> from scipy.integrate import dblquad
>>> f=lambda x,y :  np.exp(-x**2-y**2)
>>> g=lambda x: -np.sqrt(9-x**2)
>>> h=lambda x: np.sqrt(9-x**2)
>>> dblquad(f, -3, 3, g, h)
```



Integration of Data

- Data can be integrated by trapezoid rule, using trapz() of *numpy* package.

trapz(data, x, dx, axis)



Optional arguments

- *x* : an array of x (x_1,x_2… in the Fig.) which the data corresponds to.
- *dx* : spacing between data (*h* in the Fig.). Default is 1.

  * Use either *x* or *dx*; if one is known, the other is known.
- *axis* : direction of the integration. When the data is two-dimensional, 1 for row by row and 0 for column by column. Default is row-by-row.

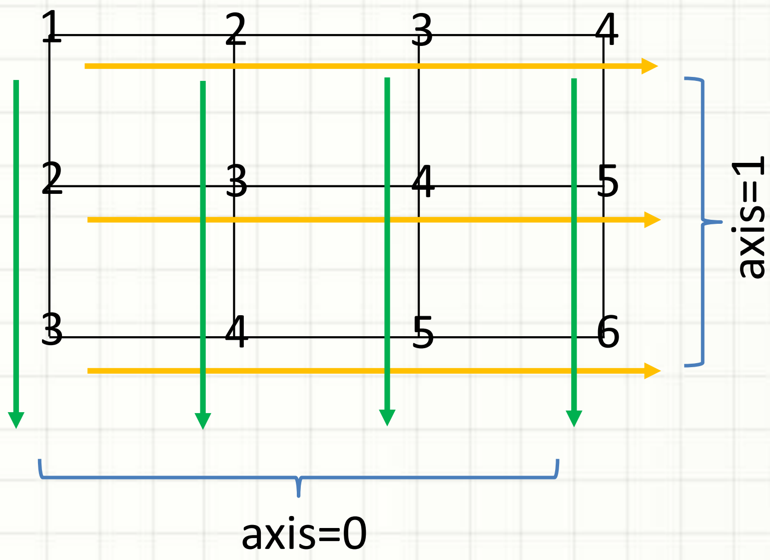
Example 10. Run the following routine.

Answer:
```
>>> import numpy as np
>>> data=[1,2,3]
>>> np.trapz(data)
>>> x=[-0.1,0,0.1]; np.trapz(data,x)
>>> np.trapz(data,dx=0.1)
```

Integration of Data

Example 11. Run the following routine.

Answer:
```
>>> import numpy as np
>>> data=[[1,2,3,4],[2,3,4,5],[3,4,5,6]]
>>> np.trapz(data,axis=0)
>>> np.trapz(data,axis=1)
```



Integration of Data

- Other methods are

  simps(data, x, dx, axis)
  romb(data, dx, axis)

* To use romb(), the number of data points should be 2^n+1.

Example 12. Generate a data set, where the elements are the values of function x2+x+1 for x from 0 through 3.2 with spacing 0.2. Integrate the data with trapz(), simps(), and romb(). Compare the result with the more accurate value obtained from function integration by quad().

Answer:

```
>>> import numpy as np
>>> from scipy.integrate import trapz, simps, romb, quad
>>> f=lambda x: x**2+x+1
>>> data = [f(x) for x in np.arange(0,3.4,0.2)]
>>> trapz(data,dx=0.2);  simps(data,dx=0.2);  romb(data,dx=0.2)
>>> quad(f,0,3.2)
```

End of Lecture 2-1

APPENDIX

- Growth of Edge Derivate Error
- Richardson extrapolation
- Romberg integral
- Chebyshev polynomial

Growth of Edge Derivative Error

1st

$$f_0^{(1)} = \frac{\bar{f}_1 - \bar{f}_0}{h} = \bar{f}_0^{(1)} + \frac{\bar{f}_0^{(2)}}{2} h + O(h^2) \qquad f_1^{(1)} = \frac{\bar{f}_2 - \bar{f}_0}{2h} = \bar{f}_1^{(1)} + \frac{\bar{f}_1^{(3)}}{6} h^2 + O(h^3)$$

2nd

$$f_0^{(2)} = \frac{f_1^{(1)} - f_0^{(1)}}{h} = \bar{f}_0^{(2)} - \frac{\bar{f}_0^{(2)}}{2} + \left(\frac{\bar{f}_0^{(3)}}{2} + \frac{\bar{f}_1^{(3)}}{6}\right) h + O(h^2)$$

$$f_1^{(2)} = \frac{f_2^{(1)} - f_0^{(1)}}{2h} = \bar{f}_1^{(2)} - \frac{\bar{f}_0^{(2)}}{4} + \frac{\bar{f}_2^{(3)}}{12} h + \frac{\bar{f}_1^{(4)}}{6} h^2 + O(h^2)$$

3rd

$$f_0^{(3)} = \frac{f_1^{(2)} - f_0^{(2)}}{h} = \bar{f}_0^{(3)} + \frac{\bar{f}_0^{(2)}}{4h} + \left(\frac{\bar{f}_2^{(3)}}{12} - \frac{\bar{f}_1^{(3)}}{6} - \frac{\bar{f}_0^{(3)}}{2}\right) + \left(\frac{\bar{f}_0^{(4)}}{2} + \frac{\bar{f}_1^{(4)}}{6}\right) h + O(h^2)$$

⋮ ⋮

Error-order decreases as the derivative goes higher order

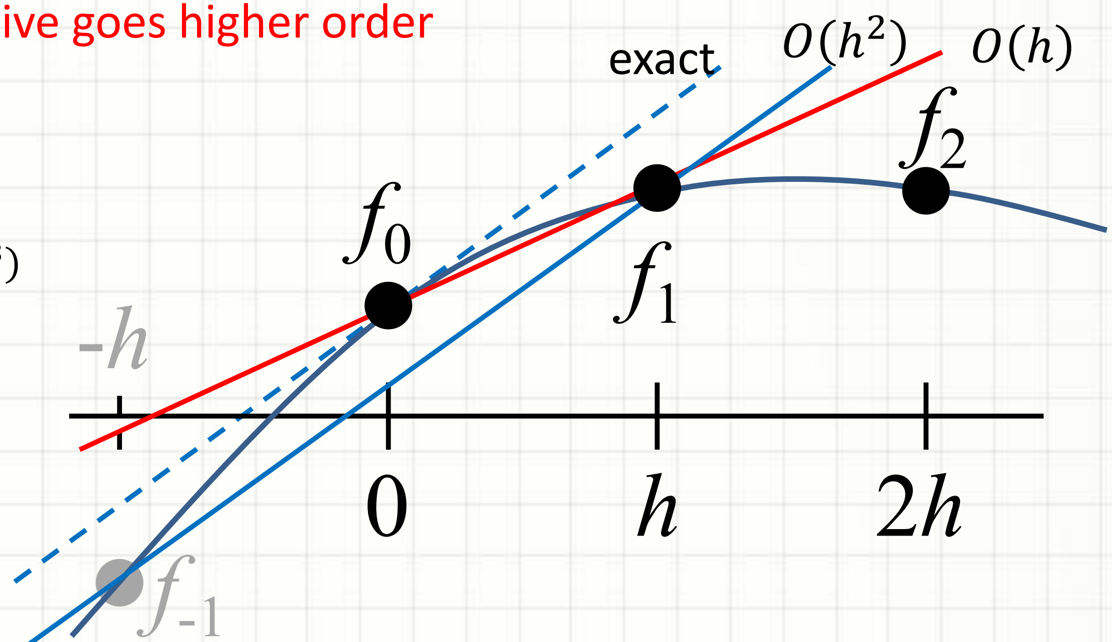
In the middle

$$f_k^{(1)} = \frac{\bar{f}_{k+1} - \bar{f}_{k-1}}{2h} = \bar{f}_k^{(1)} + \frac{\bar{f}_k^{(3)}}{6} h^2 + O(h^3)$$

$$f_k^{(2)} = \frac{f_{k+1}^{(1)} - f_{k-1}^{(1)}}{2h} = \bar{f}_k^{(2)} + \frac{\bar{f}_k^{(4)}}{3} h^2 + O(h^3)$$

⋮

Error-order is preserved when the derivative goes higher order



Richardson Extrapolation

- Richardson extrapolation is a technique to accelerate the convergence of a sequence.

- Suppose $A(h)$ approximates the exact value A up to order h^k.
- Here h corresponds to the time step dt in ODE or dx in numerical integration, etc.

$$A = A(h) + Kh^k + O(h^{k+1})$$

- To increase the accuracy, usually decrease h. Here decrease it by half.

$$A = A(h/2) + K(h/2)^k + O(h^{k+1})$$

- The error has been reduced by a factor of $1/2^k$. However, the error reduction can be accelerated by eliminating the K-term. Multiply 2^k to the 2nd equation and subtract the 1st from that.

$$(2^k - 1)A = 2^k A(h/2) - A(h) + O(h^{k+1})$$

$$\rightarrow \quad A = \frac{2^k A(h/2) - A(h)}{2^k - 1} + O(h^{k+1})$$

Richardson Extrapolation

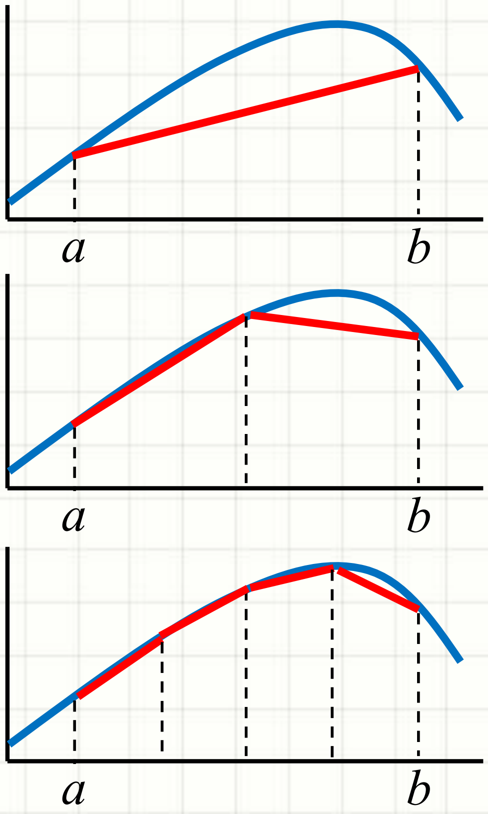
- Let's denote the expression in the left side by $R(h)$.

$$R(h) \equiv \frac{2^k A(h/2) - A(h)}{2^k - 1} \quad \text{and} \quad A = R(h) + O\left(h^{k+1}\right)$$

- By transformation to $R(h)$, the error order has been increased by one; obtained faster convergence to the exact value A than just by reducing h.
- The $R(h)$ is called the Richardson extrapolation.
- We divided h by 2, but it can be general; e.g., h/t can be used.

Romberg Integration

- The Richardson extrapolation can be applied to the trapezoid numerical integration method.



$$R(1,1) = \frac{b-a}{2^0}\left(\frac{f_a}{2} + \frac{f_b}{2}\right) + O(f'' h_1^2)$$

$$R(2,1) = \frac{b-a}{2^1}\left(\frac{f_a}{2} + \cdots + \frac{f_b}{2}\right) + O(f'' h_2^2)$$

$$R(3,1) = \frac{b-a}{2^2}\left(\frac{f_a}{2} + \cdots + \frac{f_b}{2}\right) + O(f'' h_3^2)$$

$$R(n,1) = \frac{b-a}{2^{n-1}}\left(\frac{f_a}{2} + \cdots + \frac{f_b}{2}\right) + O(f'' h_n^2)$$

Richardson Extrapolation

$$R(2,2) = \frac{2^2 R(2,1) - R(1,1)}{2^2 - 1} + O(h_1^4)$$

$$R(3,2) = \frac{2^2 R(3,1) - R(2,1)}{2^2 - 1} + O(h_2^4)$$

$$R(n,2) = \frac{2^2 R(n,1) - R(n-1,1)}{2^2 - 1} + O(h_n^4)$$

Romberg Integration

- The trapezoid integral with the interval divided by 2^{n-1} is written by $R(n,1)$.
- The 2nd argument represent the error order, i.e. $(h^2)^1$.
- Increase the error order by pairing $R(n\text{-}1,1)$ and $R(n,1)$ to get $R(n,2)$. The error becomes $(h^2)^2$.
- Repeat this procedure until desired accuracy is obtained. The iteration can go up to $R(n, N+1)$, when the number of data points is 2^N.

Chebyshev Polynomial

- The Chebyshev polynomial of order k can be obtained from the following relation.

$$T_k(\cos\theta) = \cos k\theta$$

- Set $\cos\theta = x$. Then for k=0, $T_0(x) = 1$
- For k=1, $T_1(x) = x$
- For k=2, $T_2(x) = \cos 2\theta = 2\cos^2\theta - 1 = 2x^2 - 1$
- For k=3, $T_3(x) = \cos 3\theta = 4\cos^3\theta - 3\cos\theta = 4x^3 - 3x$
- From the 1st equation, T_k is defined over [-1,1].
- Chebyshev polynomials are orthogonal. Hence an arbitrary function defined over [-1,1] can be expanded by T_k.

$$\int_{-1}^{1} \frac{T_m(x)T_n(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0 & m \neq n \\ \pi & m = n = 0 \\ \pi/2 & m = n > 0 \end{cases}$$

- Chebyshev polynomials are discretely orthogonal for the Chebyshev nodes.

$$\sum_{k=0}^{N-1} T_m(x_k)T_n(x_k) = \begin{cases} 0 & m \neq n \\ N & m = n = 0 \\ N/2 & m = n > 0 \end{cases} \quad \text{where} \quad x_k = \cos\left(\pi \frac{2k+1}{2N}\right) \qquad k = 0,1,\cdots N-1$$