Lecture 2-1: Numerical Integration

(Adapted from slides by Gerald Fux)

Zejian Li
(li.zejian@ictp.it)

21. Oct. 2024

Numerical Integration - Introduction

Numerical integration is ...

- ... about calculating the numerical value of a definite Integral of some function $f(x)$, such as (in 1 dimension):

$$\int_a^b f(x)\,\mathrm{d}x.$$

Numerical integration is not necessary if ...

- ... we know an elementary expression for the primitive $F(x) = \int f(x)\mathrm{d}x$. Then

$$\int_a^b f(x)\,\mathrm{d}x = F(b) - F(a).$$

For example:

$$\int_0^t x^2 \sin(x)\,\mathrm{d}x = -2 + (2 - t^2)\cos(t) + 2t\sin(t).$$

Numerical Integration - Introduction

We need numerical integration because ...

- ... for many functions $f(x)$ the primitive function $F(x)$ is either
  - unknown, or
  - not expressible as elementary functions, such as $\int e^{-x^2}\,\mathrm{d}x$.

  For example:

$$\int_{-\infty}^{1} e^{-x^2}\,\mathrm{d}x \simeq 1.63305$$

- ... sometimes we don't even know an elementary expression for $f(x)$, but it is itself the result of some numerical computation. For example:

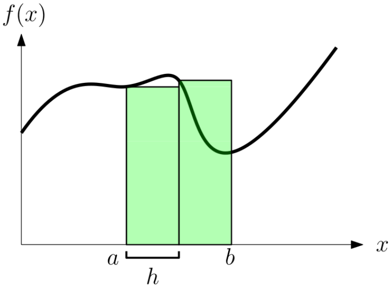
$$f(t) = \int_{-\infty}^{t} e^{-x^2}\,\mathrm{d}t \quad \rightarrow \quad \int_{0}^{2} f(t)\,\mathrm{d}t = ??$$

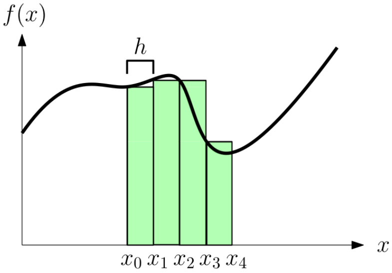
Numerical Integration Through the Riemann Sum

Consider the definition of integrals via the **Riemann sum**:

$$\int_a^b f(x)\,\mathrm{d}x \equiv \lim_{N\to\infty}\left[\sum_{k=0}^{N-1} f(x_k)\cdot h\right] \quad \text{with} \quad \begin{cases} h \equiv \frac{b-a}{N} \\ x_k \equiv a + k\cdot h \\ k = 0,\ldots,N-1 \end{cases}$$

This is an approximation for finite N, but improves for growing N and is exact for $N\to\infty$ (if the function is "well-behaved", i.e. *Riemann integrable*).





Numerical Method: Left Riemann Sum

Define $I_L(N)$ as the approximated integral:

$$I_L(N) \equiv \sum_{k=0}^{N-1} f(x_k) \cdot h$$

with

$$h \equiv \frac{b-a}{N}$$

$$x_k \equiv a + k \cdot h$$

$$k = 0, \ldots, N-1.$$

Algorithm: Left Riemann Sum

Input: function $f(x)$; boundaries a and b; small threshold ϵ.

1. Set N to some initial value, e.g. $N := 32$
2. Compute $I_{\text{old}} := I_L(N)$
3. Loop:
   - Increase N, e.g. $N := 2N$
   - Compute $I_{\text{new}} := I_L(N)$
   - If $|I_{\text{new}} - I_{\text{old}}| < \epsilon$ then exit the loop, otherwise set $I_{\text{old}} := I_{\text{new}}$.

Output: I_{new}, which is an approximate value of $\int_a^b f(x)\,\mathrm{d}x$

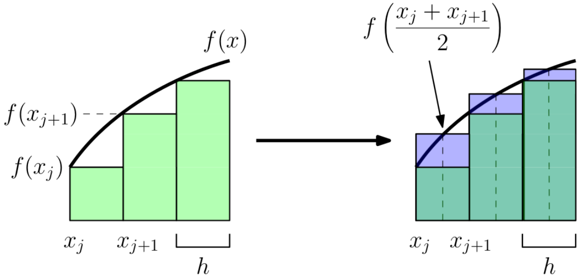
Numerical Method: Left Riemann Sum

Error scaling of the left Riemann sum

For large enough N the error decreases faster or as fast as $(1/N)^1$, i.e.

$$\int_a^b f(x)\,\mathrm{d}x = I_L(N) + \mathcal{O}\left(\frac{1}{N}\right)$$

Improved Method (a): Midpoint Method



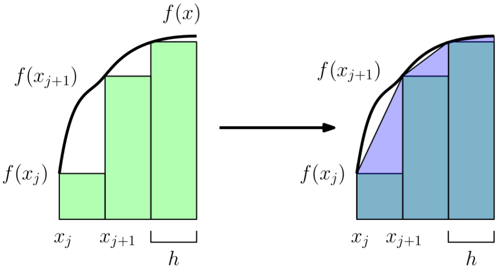
Better coverage
of area under the curve

$$I_M(N) \equiv \sum_{k=0}^{N-1} f\left(\frac{x_k + x_{k+1}}{2}\right) \cdot h \quad \text{with} \quad \begin{cases} h \equiv \frac{b-a}{N} \\ x_k \equiv a + k \cdot h \\ k = 0, \ldots, N-1 \end{cases}$$

Better convergence:

$$\int_a^b f(x)\, \mathrm{d}x = I_M(N) + \mathcal{O}\left(\frac{1}{N^2}\right) = I_M(h) + \mathcal{O}\left(h^2\right)$$

Improved Method (b): Trapezoidal Method



Area of trapeze starting in $x = x_j$:

$$A_j = \frac{h}{2}\left(f(x_j) + f(x_{j+1})\right)$$

$$I_T(N) \equiv \sum_{k=0}^{N-1} [f(x_k) + f(x_{k+1})] \cdot \frac{h}{2} \quad \text{with} \quad \begin{cases} h \equiv \frac{b-a}{N} \\ x_k \equiv a + k \cdot h \\ k = 0, \ldots, N \end{cases}$$

Convergence (same as the midpoint method):

$$\int_a^b f(x)\,\mathrm{d}x = I_T(h) + \mathcal{O}\left(h^2\right)$$

Improved Method (b): Trapezoidal Method

$$
\begin{aligned}
I_T(N) &\equiv \sum_{k=0}^{N-1} [f(x_k) + f(x_{k+1})] \cdot \frac{h}{2} \\
&\equiv \frac{h}{2} \left[\underbrace{f(x_0) + f(x_1)} + \underbrace{f(x_1) + f(x_2)} + \underbrace{f(x_2) + f(x_3)} + \dots \right]
\end{aligned}
$$

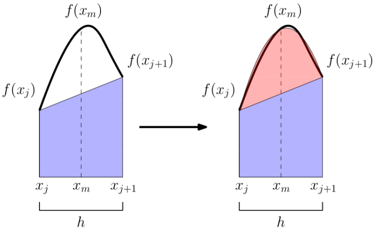
(Better) reformulation of the trapezoidal method

Note that $f(x_1), f(x_2), \dots, f(x_{N-2})$ each appear twice in the sum. Because it might be very hard to evaluate $f(x)$ it is better to **calculate each** $f(x_j)$ **only once instead of twice**. We thus implement the method in the rewritten form ...

$$
I_T(N) = \frac{h}{2} \left[f(x_0) + \left(\sum_{k=1}^{N-1} 2f(x_k) \right) + f(x_N) \right].
$$

Improved Method (c): Simpson Method

Next improvement: from Trapeziods $\rightarrow$ to **parabolic arcs.**



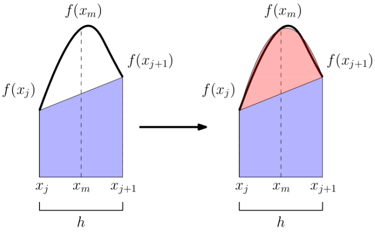
Each arc passes for

- $(x_j, f(x_j))$
- $\left(\frac{x_j + x_{j+1}}{2}, f\left(\frac{x_j + x_{j+1}}{2}\right)\right)$
- $(x_{j+1}, f(x_{j+1}))$

Algebra yields that the area is: $A_j = \frac{h}{6}\left[f(x_j) + 4f\left(\frac{x_j + x_{j+1}}{2}\right) + f(x_{j+1})\right]$

$$I_S(N) \equiv \frac{h}{6}\left[f(x_0) + 2\sum_{k=1}^{N-1} f(x_k) + 4\sum_{k=0}^{N-1} f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_N)\right] \quad \text{with} \quad \begin{cases} h \equiv \frac{b-a}{N} \\ x_k \equiv a + k \cdot h \\ k = 0, \ldots, N \end{cases}$$

Improved Method (c): Simpson Method



Each arc passes for

- $(x_j, f(x_j))$
- $\left(\frac{x_j + x_{j+1}}{2}, f\left(\frac{x_j + x_{j+1}}{2}\right)\right)$
- $(x_{j+1}, f(x_{j+1}))$

$$I_S(N) \equiv \frac{h}{6}\left[f(x_0) + 2\sum_{k=1}^{N-1} f(x_k) + 4\sum_{k=0}^{N-1} f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_N)\right] \quad \text{with} \quad \begin{cases} h \equiv \frac{b-a}{N} \\ x_k \equiv a + k \cdot h \\ k = 0, \ldots, N \end{cases}$$

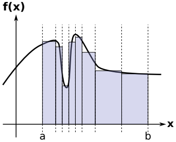
Convergence:

$$\int_a^b f(x)\,\mathrm{d}x = I_S(h) + \mathcal{O}\left(h^4\right)$$

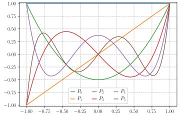
Advanced Integration Methods

Beyond these basic approaches many advanced / specialized methods exist. E.g.:

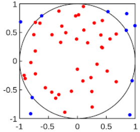
Adaptive integration:
Make the grid finer where the function changes faster.



Gaussian quadratures:
Mathematically optimal grid.



Monte Carlo integration:
Use a randomized grid; best in high dimensions.



Assignment 11

Write a FORTRAN program that computes $\int_a^b f(x)\,\mathrm{d}x$ for $f(x) = \dfrac{16x - 16}{x^4 - 2x^3 + 4x - 4}$.

- Write a function that takes the bounds a and b, and the desired precision ϵ.
- The function should integrate with the left Riemann sum, increasing N until the precision is achieved.
- The function should print the result at each step together with the current value for N (this is just for us to see what is happening).
- Test the function by calculating $\int_0^1 f(x)\,\mathrm{d}x$ with error threshold $\epsilon = 10^{-5}$ in the main program and print the result. (Can you recognize the result?)
- Submit your code as `Ass11.YourLastName.f90` to `li.zejian@ictp.it` before the next lesson.

Hints:

- Create separate functions for $f(x)$, the Riemann sum and the integration.

Bonus question:

- Implement one (or as as many as you like) of the improved methods and compare their performance.