AM213B Numerical Methods for the Solution of Differential Equations

Lecture 11

List of topics in this lecture

- Overall theme: round-off errors and propagation of round-off errors
- IEEE double precision system, base, sign, mantissa, exponent, machine precision, round-off error, "zero", arithmetic exceptions, overflow and underflow
- Loss of accuracy due to numerical cancellation
- Total error in numerical differentiation

Floating point representation (FPR)

In computers, a non-zero real number x is represented in the form

$$\mathrm{fl}(x) = \sigma \times (.a_1 a_2 \cdots a_t)_\beta \times \beta^p$$

Caution: "Zero" is stored in a special way. We will see that.

Components of the floating point representation (FPR)

- $\beta = 2$ is the base of the number system.

  β is fixed. It does not change with x.

  For illustration purpose, in some examples, we will use $\beta = 10$.

  A decimal example: $-(0.295)_{10} \times 10^4$

- $\sigma = +1$ or -1 is the sign.

  σ changes with x. It occupies 1 bit.

- $(.a_1 a_2 \cdots a_t)_\beta$ is called the mantissa.

  Mantissa changes with x. For $x \neq 0$, we can always make $a_1 \neq 0$.

  Let us look at the space required for storing the mantissa.

  t : the number of bits in the mantissa. t is fixed.

  $$0 \le a_j \le \beta - 1, \qquad j = 1, 2, \ldots, t$$

  In the decimal system, $\beta = 10$ and $0 \le a_j \le 9$.

  In the binary system, $\beta = 2$ and $0 \le a_j \le 1$

==> $a_j = 0$ or $a_j = 1$

==> $a_1 = 1$ (since $a_1 \neq 0$!)

==> We do not need to store a_1.

==> The mantissa occupies only $(t-1)$ bits.

Mathematical meaning of the mantissa:

$$(.a_1 a_2 \cdots a_t)_\beta = \frac{a_1}{\beta} + \frac{a_2}{\beta^2} + \cdots + \frac{a_t}{\beta^t}$$

Example:

In the decimal system, the mantissa has the meaning

$$(.352)_{10} = \frac{3}{10} + \frac{5}{100} + \frac{2}{1000}$$

- p is the exponent (an integer).

  p changes with x. p is stored in the form:

  $$(p + bias) = (b_k b_{k-1} \cdots b_1)_\beta$$

  k : the number of bits used to store p. k is fixed.

  bias : the shift added to make (p + *bias*) positive. *bias* is fixed.

  The range of p is limited (because k and *bias* are fixed).

  $L \le p \le U$ where $L < 0$ and $U > 0$

  L and U are determined by k and *bias*. We will see specific values of L and U when we discuss the IEEE double precision system.

Summary of the floating point representation:

In computers, a non-zero real number x is represented as

$$\text{fl}(x) = \sigma \times (.a_1 a_2 \cdots a_t)_\beta \times \beta^p$$

σ occupies 1 bit.

The mantissa $(.a_1 a_2 \cdots a_t)_\beta$ occupies $(t-1)$ bits. $a_1 = 1$ is not stored.

The exponent p occupies k bits.

Therefore, fl(x) occupies $t + k$ bits.

$$\underbrace{1}_{\text{sign}} + \underbrace{(t-1)}_{\text{mantissa}} + \underbrace{k}_{\text{exponent}} = t + k$$

Mathematical meaning of the floating point representation (FPR):

$$\sigma\times(.a_1\,a_2\cdots a_t)_\beta\times\beta^p=\sigma\times\left(\frac{a_1}{\beta}+\frac{a_2}{\beta^2}+\cdots+\frac{a_t}{\beta^t}\right)\times\beta^p$$

Machine precision

The smallest number above "1" in the floating point representation (FPR)

In the floating point representation (FPR), we have

$$\left(.1\underbrace{0\cdots0}_{t-1}\right)_\beta\times\beta^1=\left(\frac{1}{\beta}\right)\times\beta^1=1$$

"1" is in FPR. The smallest number above "1" in the FPR is

$$\left(.1\underbrace{0\cdots0}_{t-2}1\right)_\beta\times\beta^1=\left(\frac{1}{\beta}+\frac{1}{\beta^t}\right)\cdot\beta=1+\beta^{-(t-1)}$$

The difference, $\beta^{-(t-1)}$, is called the machine precision.

(Draw the real axis with 1 and $1+\beta^{-(t-1)}$ **to show the machine precision).**

Remark: machine precision is the grid size of the FPR system.

Behavior of fl(x) slightly above 1:

- "1" is in the FPR.

  $\text{fl}(1) = 1$

- $1+\beta^{-(t-1)}$ is in the FPR.

  $\text{fl}(1+\beta^{-(t-1)})=1+\beta^{-(t-1)}$

- For x between 1 and $1+\beta^{-(t-1)}$, x is NOT in the FPR:

  $$\boxed{\text{fl}(x)\neq x \quad \text{for } 1<x<1+\beta^{-(t-1)}}$$

  x is represented either as 1 or as $1+\beta^{-(t-1)}$.

- The middle point between 1 and $1+\beta^{-(t-1)}$ is $1+\beta^{-t}$.

  For $1\le x<1+\beta^{-t}$ (below the middle point), we have $\text{fl}(x)=1$

  For $x>1+\beta^{-t}$ (above the middle point), we have $\text{fl}(x)>1$

Example:

Consider the decimal system ($\beta=10$) with 3-digit mantissa ($t=3$).

$(.1\,0\,0)_{10}\times10^1=1$ is in the FPR.

The smallest number above 1 in the FPR is

$$(.101)_{10}\times 10^1 = \underbrace{1.01}_{1+\beta^{-(t-1)}}$$

For $1 < x < 1.01$, x is not represented exactly.

$$\text{fl}(1.003) = (.100)_{10}\times 10^1 = 1 \neq 1.003$$

$$\text{fl}(1.006) = (.101)_{10}\times 10^1 = 1.01 \neq 1.006$$

The largest number below "1" in the floating point representation (FPR)

If we use infinitely many bits in the mantissa, "1" can be represented as

$$1 = (.11\cdots 1\cdots)_\beta \times \beta^0, \qquad \beta = 2$$

Check: $1 = (.11\cdots 1\cdots)_2 \times 2^0 = 2^{-1} + 2^{-2} + \cdots + 2^{-t} + \cdots = 1$

With t bits in the mantissa, the largest number below "1" in the FPR is

$$\left(.\underbrace{11\cdots 1}_{t \text{ bits}}\right)_\beta \times \beta^0 = \frac{1}{\beta} + \frac{1}{\beta^2} + \cdots + \frac{1}{\beta^t} = 1 - \beta^{-t}$$

(Draw the real axis with 1 and $1 - \beta^{-t}$**).**

Behavior of fl(x) slightly below 1:

- "1" is in the FPR.

  fl(1) = 1

- $1 - \beta^{-t}$ is in the FPR.

  $\text{fl}(1-\beta^{-t}) = 1-\beta^{-t}$

- For x between $1 - \beta^{-t}$ and 1, x is NOT represented exactly:

  $$\boxed{\text{fl}(x) \neq x \quad \text{for } 1-\beta^{-t} < x < 1}$$

  x is represented either as $1 - \beta^{-t}$ or as 1.

- The middle point between $1 - \beta^{-t}$ and 1 is $1 + \beta^{-(t+1)}$.

  For $1 - \beta^{-(t+1)} < x \leq 1$ (above the middle point), we have fl(x) = 1

  For $x < 1 - \beta^{-(t+1)}$ (below the middle point), we have fl(x) < 1

(Draw 1, $1 - \beta^{-t}$ and $1 - \beta^{-(t+1)}$ **for $x \leq 1$ along with** 1, $1 + \beta^{-(t-1)}$ and $1 + \beta^{-t}$ **for $x \geq 1$ to illustrate the FPR system near $x = 1$).**

Example:

Consider the binary system ($\beta = 2$) with 53-bit mantissa ($t = 53$). Find whether or not each item below is true or false.

a) $\mathrm{fl}(1-2^{-50})=1$? b) $\mathrm{fl}(1-2^{-60})=1$?

Solution:

The middle point between $1-\beta^{-t}$ and 1 is

$$1-\beta^{-(t+1)} = 1-2^{-54}$$

a) We compare $1-2^{-50}$ with the middle point.

$$1-2^{-50} < 1-2^{-54} \quad ==> \quad \mathrm{fl}(1-2^{-50})<1 \quad ==> \quad \text{False}$$

b) For $1-2^{-60}$, we have

$$1 > 1-2^{-60} > 1-2^{-54} \quad ==> \quad \mathrm{fl}(1-2^{-60})=1 \quad ==> \quad \text{True}$$

Round-off error

Round-off error is the difference between $\mathrm{fl}(x)$ and x.

Case 1: For simplicity, we first consider the case of truncating.

With infinitely many bits in the mantissa, x is represented exactly.

$$x=\sigma\times(.a_1\,a_2\cdots a_t\,a_{t+1}\cdots)_\beta\times\beta^p$$

The floating point representation obtained by truncating is

$$\mathrm{fl}(x)=\sigma\times(.a_1\,a_2\cdots a_t)_\beta\times\beta^p$$

$$==> \quad \mathrm{fl}(x)-x=-\sigma\times\left(.\underbrace{0\cdots0}_{t}\,a_{t+1}\,a_{t+2}\cdots\right)_\beta\times\beta^p=-\sigma\times(.a_{t+1}\,a_{t+2}\cdots)_\beta\times\beta^{p-t}$$

The absolute error (the case of truncating) is

$$\left|\mathrm{fl}(x)-x\right|=(.a_{t+1}\,a_{t+2}\cdots)_\beta\times\beta^{p-t}\le\beta^{p-t}$$

The relative error (the case of truncating) is

$$\frac{|\mathrm{fl}(x)-x|}{|x|}\le\frac{\beta^{p-t}}{(.a_1\,a_2\cdots a_t\,a_{t+1}\cdots)_\beta\times\beta^p}\le\frac{\beta^{p-t}}{\beta^{-1}\cdot\beta^p}=\beta^{-(t-1)}$$

Here we have used $(.a_1\,a_2\cdots a_t\,a_{t+1}\cdots)_\beta\ge(.1)_\beta=\beta^{-1}$

Case 2: The case of rounding. We have

$$|\mathrm{fl}(x)-x|\le\frac{1}{2}\beta^{p-t}$$

$$\frac{|\mathrm{fl}(x)-x|}{|x|}\le\frac{1}{2}\beta^{-(t-1)}$$

Basically, both the absolute and relative errors are halved in the case of rounding.

A mathematical expression of fl(x) for error analysis

We can write fl(x) as

$$\text{fl}(x) = x + \text{fl}(x) - x = x + x \cdot \frac{\text{fl}(x) - x}{x} = x\left(1 + \frac{\text{fl}(x) - x}{x}\right)$$

We write it as fl(x) = x (1+ε) where $\varepsilon = \dfrac{\text{fl}(x) - x}{x}$ is bounded by

$$|\varepsilon| = \left|\frac{\text{fl}(x) - x}{x}\right| \le \frac{1}{2}\beta^{-(t-1)}.$$

Thus, we have

$$\boxed{\text{fl}(x) = x(1+\varepsilon), \quad |\varepsilon| \le \frac{1}{2}\beta^{-(t-1)}}$$

Note: This form of fl(x) is very useful in error analysis.

IEEE double precision FPR

$$\text{fl}(x) = \sigma \times (.a_1 a_2 \cdots a_t)_\beta \times \beta^p$$

$\beta = 2$, $t = 53$

$(p + bias) = (b_k b_{k-1} \cdots b_1)_\beta$, $bias = 1023$, $k = 11$

$L \le p \le U$, $L = -1022$, $U = 1023$

We discuss a few items about IEEE double precision:

- fl(x) occupies

  $1 + (t - 1) + k = 64$ bits = 8 bytes (1 byte = 8 bits)

- Machine precision:

  $\beta^{-(t-1)} = 2^{-52} \approx 2.22 \times 10^{-16}$

- Round-off error:

  $$\text{fl}(x) = x(1+\varepsilon), \quad |\varepsilon| \le \frac{1}{2}\beta^{-(t-1)} = 2^{-53} \approx 1.11 \times 10^{-16}$$

- Question: How is "0" represented?

  p is stored in an 11-bit number: $(p + bias) = (b_k b_{k-1} \cdots b_1)_\beta$.

  The smallest of $(b_{11} b_{10} \cdots b_1)_\beta$ is $(0\,0 \cdots 0)_\beta = 0$

The largest of $(b_{11}\, b_{10} \cdots b_1)_\beta$ is $(1\,1 \cdots 1)_\beta = 2^0 + 2^1 + 2^2 + 2^{10} = 2^{11} - 1 = 2047$

==> $0 \le (b_{11}\, b_{10} \cdots b_1)_\beta \le 2047$

In IEEE double precision, the range of p is given as

$L \le p \le U, \qquad L = -1022, \qquad U = 1023, \qquad bias = 1023$

==> $1 \le (p + bias) \le 2046$

Comparing the range of $(p + bias)$ and the range of $(b_{11}\, b_{10} \cdots b_1)_\beta$, we see that

$(0\ 0 \cdots 0)_\beta$ and $(1\ 1 \cdots 1)_\beta$ are not used in storing p.

They are used to indicate special situations.

$(0\ 0 \cdots 0)_\beta$ is used to indicate "0" (the real number zero).

$(1\ 1 \cdots 1)_\beta$ is used to indicate arithmetic exceptions (*Inf*, *–Inf*, *NaN*).

Overflow and underflow

The largest number (absolute value) in the IEEE double precision is

$$B = \max\left\{(.a_1\, a_2 \cdots a_t)_\beta \times \beta^p\right\} = (.1\,1 \cdots 1)_\beta \cdot \beta^U \approx \beta^U = 2^{1023} \approx 10^{308}$$

The smallest non-zero number (absolute value) is

$$b = \min_{a_1=1}\left\{(.a_1\, a_2 \cdots a_t)_\beta \times \beta^p\right\} = (.1\,0 \cdots 0)_\beta \beta^L = \beta^{L-1} = 2^{-1022-1} \approx 10^{-308}$$

Overflow:

If $|x|$ > B, then fl(x) = inf.

This is called overflow.

Note: overflow is a fatal error.

Underflow:

If $|x| < \frac{b}{2}$, then fl(x) = 0.

This is called underflow.

Note: underflow is a non-fatal error.

Loss of accuracy due to numerical cancellation

Suppose $A > 0$ and $B > 0$. We calculate $(A - B)$ in IEEE double precision.

$$\text{fl}(A) = A(1 + \varepsilon_1), \qquad |\varepsilon_1| \sim 10^{-16}$$

$$\mathrm{fl}(B) = B(1+\varepsilon_2), \qquad |\varepsilon_2| \sim 10^{-16}$$

$$\Longrightarrow \quad \underbrace{\mathrm{fl}(A) - \mathrm{fl}(B)}_{\text{Numerical value}} = A(1+\varepsilon_1) - B(1+\varepsilon_2)$$

$$= \underbrace{(A-B)}_{\text{Exact value}} + \underbrace{(A\varepsilon_1 - B\varepsilon_2)}_{\text{Absolute error}}$$

$$= \underbrace{(A-B)}_{\text{Exact value}} \left(1 + \underbrace{\frac{A\varepsilon_1 - B\varepsilon_2}{A-B}}_{\text{Relative error}} \right)$$

The absolute error:

$$(A\varepsilon_1 - B\varepsilon_2) = (A+B)\varepsilon_3, \qquad |\varepsilon_3| \sim 10^{-16}$$

The relative error:

$$\frac{A\varepsilon_1 - B\varepsilon_2}{A-B} = \frac{A+B}{A-B}\varepsilon_3, \qquad |\varepsilon_3| \sim 10^{-16}$$

The relative round-off error is magnified by a factor of $\left| \frac{A+B}{A-B} \right|$.

When A is close to B, the factor $\frac{A+B}{A-B}$ is large and we lose accuracy.

When $\frac{A+B}{A-B} \sim 10^{16}$, the relative error is $10^{16}\,\varepsilon_3 \sim 100\%$.

Summary:

When we find the difference of two nearly equal numbers, we lose accuracy.

This is called loss of accuracy due to numerical cancellation.

Example 1 of numerical cancellation

Calculation of one root of quadratic equation $ax^2 + bx + c = 0$

where $a = 10^{-8}$, $b = 10^8$, $c = 1.23 \times 10^8$

Numerical formula #1:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$ (we study only one of the two roots)

In IEEE double precision (i.e., in Matlab):

```
r1 = (–b+sqrt(b^2–4*a*c))/(2*a)
```

==> r1 = –0.745 (not a good solution).

The loss of accuracy is caused by numerical cancellation in the numerator. The two terms b and $\sqrt{b^2-4ac}$ are very close to each other.

$$\sqrt{b^2-4ac} = b\sqrt{1-\frac{4ac}{b^2}} \approx b, \qquad \frac{ac}{b^2} = 1.23\times 10^{-16}$$

Remedy:

We derive an alternative formula to avoid numerical cancellation. We multiply both the numerator and the denominator by the conjugate of the numerator.

$$r_1 = \frac{(-b+\sqrt{b^2-4ac})(b+\sqrt{b^2-4ac})}{2a(b+\sqrt{b^2-4ac})} = \frac{b^2-(b^2-4ac)}{2a(b+\sqrt{b^2-4ac})} = \frac{-2c}{(b+\sqrt{b^2-4ac})}$$

Numerical formula #2:

$$r_1 = \frac{-2c}{(b+\sqrt{b^2-4ac})}$$

Formula #2 is mathematically equivalent to formula #1 in exact arithmetic.

In finite precision arithmetic, however, they have very different behaviors.

In IEEE double precision (i.e., in Matlab),

```
r1 = –2*c/(b+sqrt(b^2–4*a*c))
```

==> r1 = –1.23 (a good solution).

Formula #2 dose not suffer from numerical cancellation.

The method for avoiding numerical cancellation is ad hoc (problem specific).

In general, there are 3 options for avoiding numerical cancellation.

1. Derive an exact alternative formula
2. Derive an approximate alternative formula
3. Prevent A and B from getting too close

We used option 1 in Example 1 above. In Example 2 below, we resort to option 2.

Example 2 of numerical cancellation

Calculation of $f(b) = \dfrac{\exp(b)-1-b}{b^2}$ where $b = 10^{-8}$.

Numerical formula #1: implement the expression above directly

When b is small, this formula suffers from numerical cancellation.

Numerical formula #2

We use the Taylor expansion of exp(b).

$$\exp(b) = 1 + b + \frac{b^2}{2!} + \frac{b^3}{3!} + \frac{b^4}{4!} + \cdots$$

$$f(b) = \frac{1}{b^2}\left(\frac{b^2}{2!} + \frac{b^3}{3!} + \frac{b^4}{4!} + \cdots \right) = \frac{1}{2!} + \frac{b}{3!} + \frac{b^2}{4!} + \cdots$$

Formula #2 is approximate, not exact. The accuracy of approximation is good when b is small. Formula #2 does not suffer from numerical cancellation.

Next, we study a problem where we need option 3.

When we compare the exact value with the numerical output from a computer, the difference between the two is called the total error, which contains both the discretization error and the round-off error.

The total error in numerical differentiation

The first order numerical difference method

$$\underbrace{\frac{f(x+h)-f(x)}{h}}_{\substack{\text{Numerical result} \\ \text{in exact arithmetic}}} = \underbrace{f'(x)}_{\substack{\text{Exact} \\ \text{value}}} + \underbrace{e(h)}_{\substack{\text{Discretization} \\ \text{error}}}$$

$e(h) = O(h)$

Question: How should we select h?

Answer:

In exact arithmetic, the smaller h is, the better.

In IEEE double precision, we need to study the total error.

The numerical output from a computer

$\frac{f(x+h)-f(x)}{h}$ is the numerical result in exact arithmetic.

The numerical output from a computer is (essentially)

$$\underbrace{\frac{\mathrm{fl}\left(f(x+h)\right)-\mathrm{fl}\left(f(x)\right)}{h}}_{\substack{\text{Numerical output} \\ \text{from a computer}}}$$

We examine the numerical output.

$$\mathrm{fl}\left(f(x+h)\right) = f(x+h)(1+\varepsilon_1), \qquad |\varepsilon_1| \sim 10^{-16}$$

$$\mathrm{fl}\Big(f(x)\Big) = f(x)(1+\varepsilon_2), \qquad |\varepsilon_2| \sim 10^{-16}$$

$$\Longrightarrow \qquad \frac{\mathrm{fl}\Big(f(x+h)\Big) - \mathrm{fl}\Big(f(x)\Big)}{h} = \frac{f(x+h)(1+\varepsilon_1) - f(x)(1+\varepsilon_2)}{h}$$

$$= \frac{f(x+h)-f(x)}{h} + \frac{f(x+h)\varepsilon_1 - f(x)\varepsilon_2}{h}$$

$$= \underbrace{f'(x)}_{\substack{\text{Exact} \\ \text{value}}} + \underbrace{e(h)}_{\substack{\text{Discretization} \\ \text{error}}} + \underbrace{\frac{f(x+h)\varepsilon_1 - f(x)\varepsilon_2}{h}}_{\substack{\text{Effect of} \\ \text{round-off error}}}$$

The total error in the first order method

We define the total error $E_T(h)$ as the difference between the exact value and the numerical output from a computer (with finite precision arithmetic).

$$E_T(h) = \left| \underbrace{\frac{\mathrm{fl}\Big(f(x+h)\Big) - \mathrm{fl}\Big(f(x)\Big)}{h}}_{\substack{\text{Numerical output} \\ \text{from a computer}}} - \underbrace{f'(x)}_{\substack{\text{Exact} \\ \text{value}}} \right|$$

$$= \left| \underbrace{e(h)}_{\substack{\text{Discretization} \\ \text{error}}} + \underbrace{\frac{f(x+h)\varepsilon_1 - f(x)\varepsilon_2}{h}}_{\substack{\text{Effect of} \\ \text{round-off error}}} \right|$$

The discretization error satisfies

$$e(h) \approx C\,h \to 0 \qquad \text{as} \quad h \to 0$$

The effect of round-off error satisfies

$$\frac{f(x+h)\varepsilon_1 - f(x)\varepsilon_2}{h} = \Big(|f(x+h)| + |f(x)|\Big)\frac{|\varepsilon_3|}{h}, \qquad |\varepsilon_3| \sim 10^{-16}$$

$$\sim \Big(|f(x+h)| + |f(x)|\Big)\frac{10^{-16}}{h} \to \infty \quad \text{as} \quad h \to 0$$

Let us consider the simplified situation where the total error is given by

$$E_T(h) = h + \frac{10^{-16}}{h}$$

Minimizing $E_T(h)$, we obtain

$$\arg\min_h E_T(h) = 10^{-8}, \qquad \min_h E_T(h) = 2\times 10^{-8}$$

Therefore, we should use $h \sim 10^{-8}$ in the first order numerical differentiation method.

Remark:

$\min_h E_T(h)$ is the minimum total error that can be achieved using the given numerical method and the given finite precision arithmetic. In other words, given the numerical method and the finite precision arithmetic, we cannot make the total error smaller than $\min_h E_T(h)$ no matter what value we use for h!

Total error in the second order method

$$\underbrace{\frac{f(x+h)-f(x-h)}{2h}}_{\substack{\text{Numerical}\\ \text{approximation}}} = \underbrace{f'(x)}_{\substack{\text{Exact}\\ \text{value}}} + \underbrace{e(h)}_{\substack{\text{Discretization}\\ \text{error}}}$$

$$e(h) = O(h^2)$$

The total error is

$$E_T(h) = \left| \underbrace{\frac{\text{fl}\left(f(x+h)\right)-\text{fl}\left(f(x-h)\right)}{2h}}_{\substack{\text{Numerical output}\\ \text{from a computer}}} - \underbrace{f'(x)}_{\substack{\text{Exact}\\ \text{value}}} \right|$$

$$= \left| \underbrace{e(h)}_{\substack{\text{Discretization}\\ \text{error}}} + \underbrace{\frac{f(x+h)\varepsilon_1 - f(x-h)\varepsilon_2}{h}}_{\substack{\text{Effect of}\\ \text{round-off error}}} \right|$$

where

$$e(h) \approx C h^2, \qquad |\varepsilon_3| \sim 10^{-16}$$

$$\frac{f(x+h)\varepsilon_1 - f(x-h)\varepsilon_2}{h} = \left(|f(x+h)|+|f(x-h)|\right)\frac{|\varepsilon_3|}{h}, \qquad |\varepsilon_3| \sim 10^{-16}$$

$$\sim \left(|f(x+h)|+|f(x-h)|\right)\frac{10^{-16}}{h} \to \infty \quad \text{as} \quad h \to 0$$

In a simplified situation, the total error is

$$E_T(h) = h^2 + \frac{10^{-16}}{h}$$

Minimizing $E_T(h)$, we obtain

$$\arg\min_h E_T(h) = 10^{-5.43}, \qquad \min_h E_T(h) = 10^{-10.39}$$

Therefore, we should use $h \sim 10^{-5}$ in the 2nd order numerical differentiation method.

Remark:

The minimum total error of the second order method is smaller than that of the first order method. By using the second order method with a suitable value of h, we can achieve a lower total error than what can be achieved using the first order method. This is an advantage of high order methods.

Total error in the fourth order method

$$\underbrace{\frac{-f(x+2h)+8f(x+h)-8f(x-h)+f(x-2h)}{12h}}_{\text{Numerical approximation}} = \underbrace{f'(x)}_{\text{Exact value}} + \underbrace{e(h)}_{\text{Discretization error}}$$

$$e(h) = O(h^4)$$

In a simplified situation, the total error is given by

$$E_T(h) = h^4 + \frac{10^{-16}}{h}$$

Minimizing $E_T(h)$, we obtain

$$\arg\min_h E_T(h) = 10^{-3.32}, \qquad \min_h E_T(h) = 10^{-12.58}$$

Therefore, we should use $h \sim 10^{-3}$ in the 4th order numerical differentiation method.

Advantage of high order methods:

First order method: $\min_h E_T(h) = 2\times 10^{-8}$

Second order method: $\min_h E_T(h) = 10^{-10.39}$

Fourth order method: $\min_h E_T(h) = 10^{-12.58}$

Remarks:

- Given the numerical method and the finite precision arithmetic, we cannot make the total error smaller than $\min_h E_T(h)$.
- With a higher order method, we can achieve a smaller total error $\min_h E_T(h)$.