Tu: 9/3/13 Math 471, Fall 2013, Section 001 Lecture 1

1 Course intro

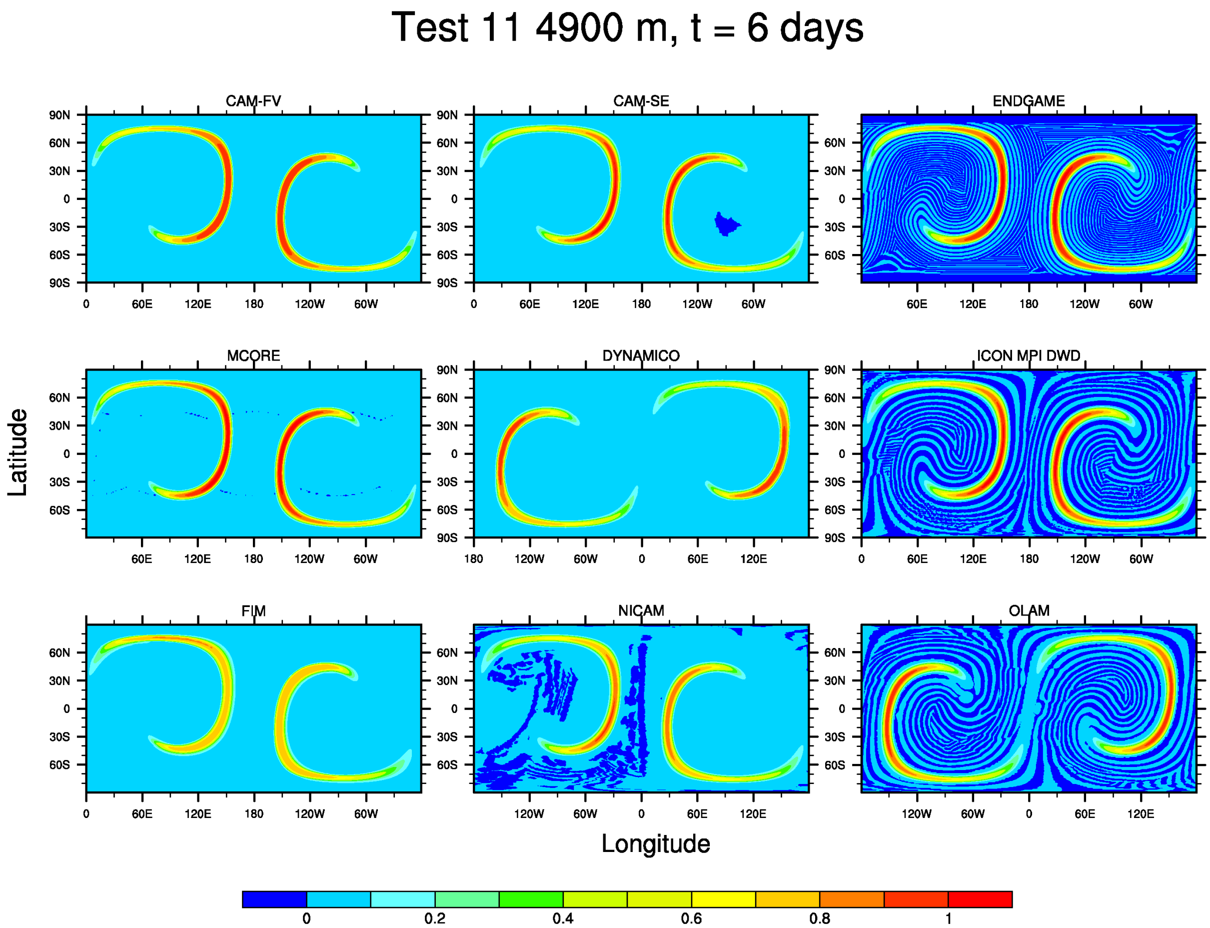
Notes :

- Take attendance.
- Instructor introduction.
- Handout : Course description.
  - Note the exam days (and don't be absent).
  - Bookmark the course webpage.
  - Matlab: We'll use it but I won't have time to teach coding– note the office hours.
  - Tests : non-programmable calculators only, plus one-page of notes (one-sided for midterm, two-sided for final).
- **Assigned reading :** Bradie, chapter 1.
- Not all topics from the text will be covered in class, some homework problems / test problems may come from material in the text or course webpage, even if they aren't explicitly covered in class.

Numerical methods

- How to solve equations with computers
- Building blocks of all computer models
- How to use them – we cannot always trust a computed result
- "Black box" syndrome: Modelers beware!

The following graphic depicts 9 different models' solutions to the linear advection equation in spherical geometry. The models are all either operational climate models.



2 Representing numbers

Question 1:

- What representations are exact?
- Which are approximate?

△

2.1 Symbolic representation

Example 1:

- π
- e
- $\frac{2}{3}$
- $\sqrt{2}$

△

2.2 Numerical representation

We use a positional system. The position of each digit relative to the point guides our understanding of the number.

$$
\begin{aligned}
x &= \pm \left(d_n d_{n-1} d_{n-2} \cdots d_1 d_0 . d_{-1} d_{-2} \cdots \right)_\beta \\
&= \pm \left(d_n \beta^n + d_{n-1}\beta^{n-1} + \cdots + d_0 \beta^0 + d_{-1}\beta^{-1} + d_{-2}\beta^{-2} + \cdots \right) \\
\beta &= \text{ base}, \\
d_i &= \text{ digits}, \quad 0 \le d_i \le \beta - 1 \quad \text{ for all } i
\end{aligned}
$$

Other common bases are $\beta = 8$ and $\beta = 16$; perhaps $\beta = 60$?

Example 2: $\beta = 10$: decimal

- $(2013)_{10} = 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 3 \cdot 10^0$
- $(0.360)_{10} = 3 \cdot 10^{-1} + 6 \cdot 10^{-2} + 0 \cdot 10^{-3}$

△

Example 3: $\beta = 2$: binary

- $(101011.01)_2 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$

  $= 32 + 8 + 2 + 1 + 0.25 = (43.25)_{10} = 4 \cdot 10^1 + 3 \cdot 10^0 + 2 \cdot 10^{-1} + 5 \cdot 10^{-2}$

△

Question 2: Are the above numerical representations exact?

△

Question 3: Why do humans prefer the decimal representation? Why would binary make sense for computers?

△

2.3 Floating point representation

Text: section 1.3

- Computers use a 'floating point representation' for real numbers.
- Constant number of significant digits.

Unlike the previous two sections, floating point representations presume a finite number of digits, and hence necessarily approximate irrational numbers and even some rationals.

Question 4: Is a floating point representation exact?

△

Question 5: Why would this be necessary in a typical computer, but not necessarily so for a human?

△

A floating point number is represented as

$$x = (0.d_1 d_2 \cdots d_n)_\beta \cdot \beta^e \qquad d_1 \neq 0 \tag{1}$$

where n is the number of significant digits, β is the base, and e is the exponent. The string of digits, $d_1 d_2 \cdots d_n$ is called the mantissa.
A floating point number system is defined by n, β and M, where M is an integer such that $-M \leq e \leq M$.
In IEEE double precision (the standard for most scientific computing), $\beta = 2$, $n = 53$, and $M = 1023$. Numbers are stored across 64 bits (bit = **b**inary dig**it**). 1 bit = sign of mantissa, 1 bit = sign of exponent, the mantissa is stored across 52 bits, which leaves 10 bits for the exponent.

Notes

- Floating point number systems are discrete (finite and not continuous) sets
- They have a maximum element and a minimum element
- They contain the number zero, and have a smallest positive element and a largest negative element

Question 6: What does multiplication by β^e do in (1)?

△

Example 4: Consider the floating point system defined by $\beta = 2$, $n = 4$, and $M = 3$.

1. What is the largest element in this set?

   The largest element in any floating point system will have every $d_i = \beta - 1$, and the largest possible exponent. Thus,

$$\begin{aligned} x_{\max} &= (0.1111)_2 \cdot 2^3 \\ &= (2^{-1} + 2^{-2} + 2^{-3} + 2^{-4}) \cdot 2^3 = 2^2 + 2 + 1 + 0.5 \\ &= (7.5)_{10} \end{aligned}$$

2. What is the smallest positive element of this set?

   The smallest positive number in the system will have only 1 nonzero significant digit, and the minimum exponent:

$$\begin{aligned} x_{\min} &= (0.1000)_2 \cdot 2^{-3} \\ &= 2^{-4} \\ &= (0.0625)_{10} \end{aligned}$$

△

Definition 1. *Absolute error, E_A.* Let $p \in \mathbb{R}$ be a real number and let p^* be an approximation of p.

$$E_A = |p - p^*|$$

Definition 2. *Relative error, E_R.* Let $p \in \mathbb{R}$ be a real number and let p^* be an approximation of p.

$$E_R = \frac{|p - p^*|}{|p|}$$

Let $\mathrm{fl}(x)$ be the floating point representation associate with $x \in \mathbb{R}$.
Then $x - \mathrm{fl}(x)$ is roundoff error.

Example 5:

$$\begin{aligned} \pi &= 3.14159265358797\cdots \\ &= (11.00100100001\cdots)_2 \end{aligned}$$

1. For the system discussed earlier, with $\beta = 2, n = 4, M = 3$, the representation of π is rounded to

   $$\mathrm{fl}(\pi) = (0.1101)_2 \cdot 2^2 = (3.25)_{10}.$$

   This is the closest floating point number to π in that system.

2. In reality, with $n = 52$, the roundoff error in $\mathrm{fl}(\pi)$ is approximately $2^{-52} \approx 10^{-15}$.

$\triangle$

The numbers that define a floating point system are determined by the hardware and software you use (loosely, your "machine").

Definition 3. *Machine precision.* The largest *relative* gap between floating point numbers is defined as a machine unit, u, and is given by

$$u = \frac{1}{2}\beta^{1-n}.$$

See page 36 for the derivation of this quantity.

2.3.1 Floating point arithmetic

Text: section 1.4

Assumption 4. *For all $x \in \mathbb{R}$, there is an ϵ with $|\epsilon| < u$ such that*

$$fl(x) = x(1 + \epsilon).$$

Thus, the difference between any real number and its floating point representation is always less than machine precision, in relative terms, i.e.,

$$|x - fl(x)| < xu.$$

Definition 5. *"Big O" notation.* To say that $f(h) = O(g(h))$ implies proportionality and a limit. If $f(h)$ and $g(h)$ are two functions of h, then

$$f(h) = O\big(g(h)\big) \quad \text{as } h \to 0$$

implies that there exists a constant C such that

$$|f(h)| < C\,|g(h)| \quad \text{for all } h \text{ sufficiently small.}$$

The interpretation is that $f(h)$ decays to zero at least as fast as $g(h)$ as $h \to 0$.

Question 7: How do roundoff errors behave under basic arithmetic operations (addition, subtraction, multiplication, division)?

1. Is $\text{fl}(x) \cdot \text{fl}(y) = xy(1+\epsilon)$ for some $|\epsilon| < u$?

$$\begin{aligned} \text{fl}(x) \cdot \text{fl}(y) &= x(1+\epsilon_x)y(1+\epsilon_y) \\ &= xy(1+\epsilon_x+\epsilon_y+\epsilon_x\epsilon_y) \end{aligned}$$

   We define $\epsilon_{xy} = \epsilon_x + \epsilon_y$ and note that since both ϵ_x and ϵ_y are very small, $\epsilon_x\epsilon_y$ is much smaller. Thus,

$$\text{fl}(x) \cdot \text{fl}(y) = xy(1+\epsilon_{xy}) + O(\epsilon^2) \quad \text{as } \epsilon \to 0.$$

2. Is $\text{fl}(x) + \text{fl}(y) = (x+y)(1+\epsilon)$ for some $|\epsilon| < u$?

$$\begin{aligned} \text{fl}(x) + \text{fl}(y) &= x(1+\epsilon_x) + y(1+\epsilon_y) \\ &= x + x\epsilon_x + y + y\epsilon_y \\ &= (x+y)\left(1 + \frac{x\epsilon_x + y\epsilon_y}{x+y}\right) \end{aligned}$$

   Beware!

   - Although we may assume that ϵ_x and ϵ_y are small, $x\epsilon_x$ and $y\epsilon_y$ might not be.
   - Further more, if $x + y \approx 0$, the denominator becomes unbounded, and $\text{fl}(x) + \text{fl}(y)$ may not be close to $x+y$ at all!

△

Example 6: Consider a 4 decimal digit floating point system with $x = 0.1234$ and $y = -0.1233$. Then

$$x + y = 0.0001 = (0.1000)_{10} \cdot 10^{-3}.$$

This result has only 1 significant digit. This is known as cancellation error.

△

Example 7: Quadratic formula, page 45.

$$ax^2 + bx + c = 0 \;\Rightarrow\; x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$0.2x^2 - 47.91x + 6 = 0 \;\Rightarrow\; x = 239.4247, 0.1253 : \text{ Matlab}$$

Now suppose we use 4 decimal digit arithmetic.

$$x = \frac{47.91 \pm \sqrt{47.91^2 - 4(0.2)6}}{2(0.2)} = \frac{47.91 \pm \sqrt{2295 - 4.8}}{0.4} = \frac{47.91 \pm \sqrt{2290}}{0.4}$$

$$= \frac{47.91 \pm 47.85}{0.4} = \begin{cases} \dfrac{47.91 + 47.85}{0.4} = \dfrac{95.76}{0.4} = 239.4 \ : \ \text{all 4 digits are correct} \\ \\ \dfrac{47.91 - 47.85}{0.4} = \dfrac{0.06}{0.4} = 0.15 \ : \ \text{only 1 digit is correct} \end{cases}$$

The problem is due to loss of significance in the subtraction $47.91 - 47.85$. One remedy is to use higher precision arithmetic (Matlab), but another option is to reformulate the arithmetic.

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

$$= \frac{2 \cdot 6}{47.91 + 47.85} = \frac{12}{95.76} = 0.1253 \ : \ \text{now all 4 digits are correct}$$

△

3 Finite differences

Text: section 6.2

Recall : We have discussed roundoff error due to floating point representations and floating point arithmetic– each of these will appear when we use a computer to evaluate a function.

Question 8: How do we evaluate derivatives of functions?

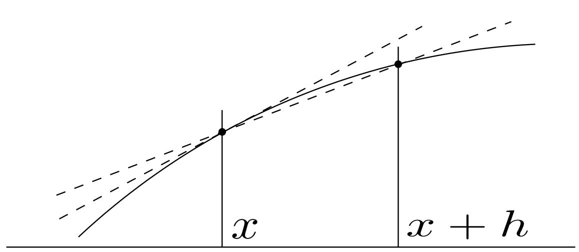
Idea : Start with the definition of a derivative...

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

then approximate for some step size $h > 0$

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} = D_+ f(x). \tag{2}$$

Graphically, we are approximating the slope of the tangent line to $f(x)$ with the slope of the secant line between $f(x)$ and $f(x+h)$.



△

Question 9: How accurate should we expect (2) to be?

Taylor series analysis : Recall:

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \frac{1}{6}f'''(a)(x-a)^3 + \cdots \tag{3}$$

We write (3) in an equivalent form. Replace x with $x+h$ and replace a with x. Then $x - a = h$ in (3) and

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + \cdots \\ \Longrightarrow \quad \underbrace{f'(x)}_{\text{exact value}} &= \underbrace{\frac{f(x+h)-f(x)}{h}}_{\text{approximation}} \underbrace{- \frac{h}{2}f''(x) - \frac{1}{6}h^2 f'''(x) - \cdots}_{\text{truncation error}} \end{aligned}$$

Thus, the error in our approximation (2) is proportional to h, since

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

and we say that $D_+f(x)$ is a first order approximation of $f'(x)$, since $D_+f(x) = f'(x) + O(h)$.

△

Thursday, 9/5/13

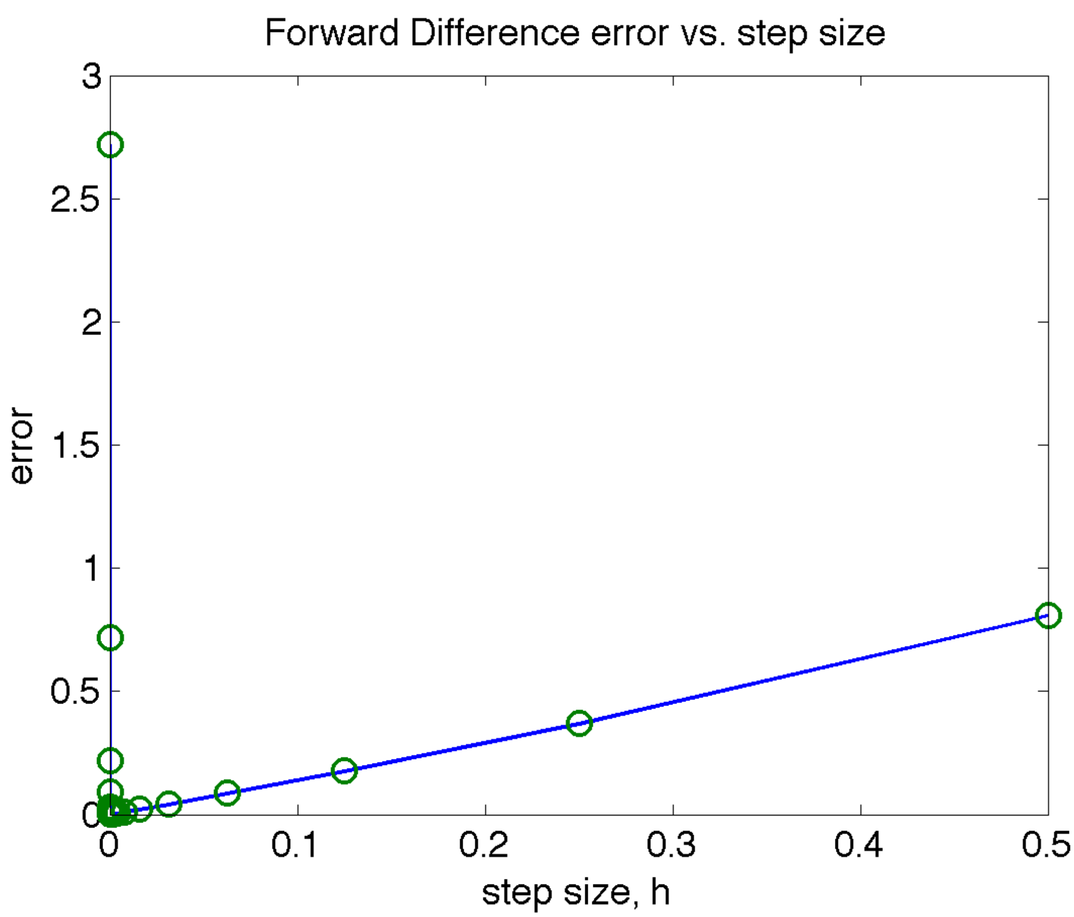
Example 8: If $f(x) = e^x, x = 1$, then $f'(1) = e = 2.71828\ldots$ is the exact value.

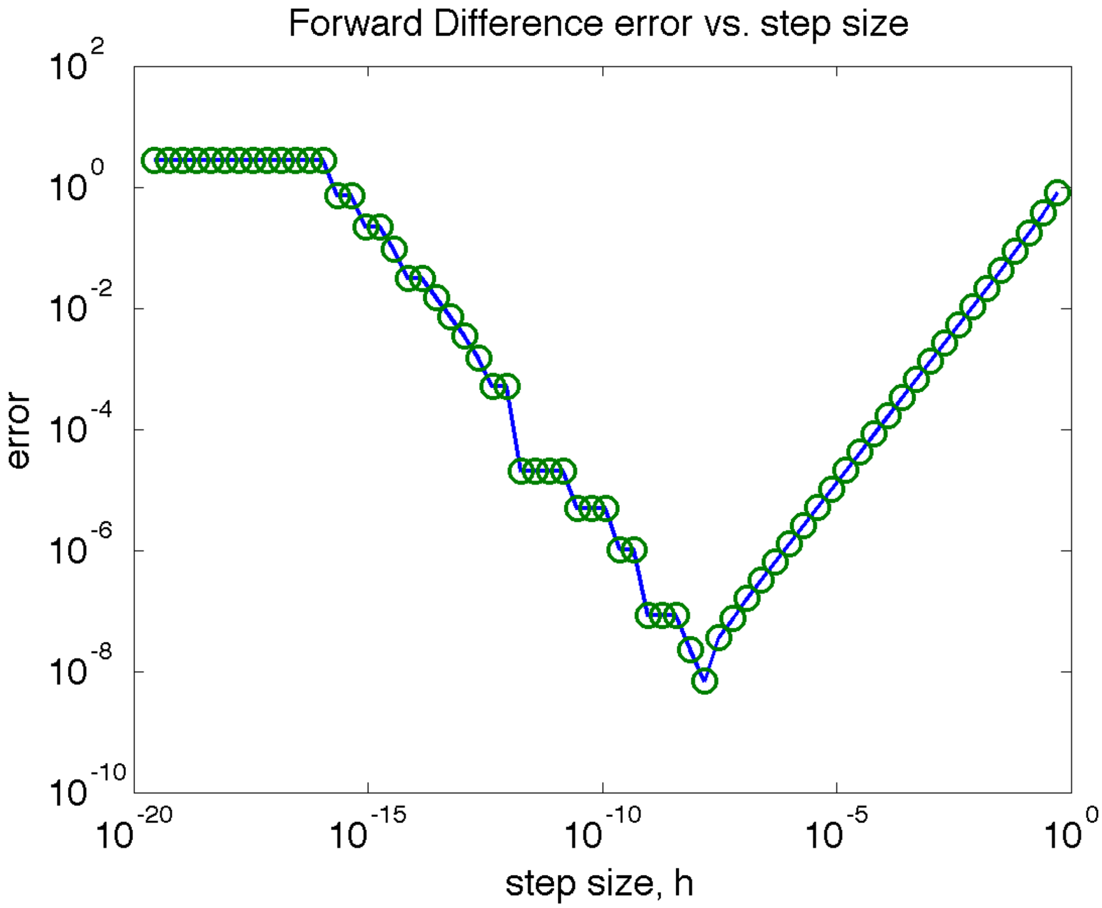
h	D_+f	$f'(x) - D_+f$	$(f'(x) - D_+f)/h$
0.1	2.8588	-0.1406	-1.4056
0.05	2.7874	-0.0691	-1.3821
0.025	2.7525	-0.0343	-1.3705
0.0125	2.7353	-0.0171	-1.3648
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
0	e	0	$-\frac{e}{2} = -\frac{1}{2}f''(1)$

Beware! In practice something unexpected happens when h is very small.

```
clear;
%% Forward difference demonstration
exact_value = exp(1);

tic

for j=1:65
       h(j) = 1/2^j;
        computed_value = (exp(1+h(j)) - exp(1))/h(j);
        error(j) = abs(exact_value - computed_value);
end

```

```
figure(1);clf;
plot(h,error,h,error,'o','LineWidth',2,'MarkerSize',12);
set(gca,'FontSize',18);
xlabel('step size, h');
ylabel('error');
title('Forward Difference error vs. step size');

figure(2);clf;
loglog(h,error,h,error,'o','LineWidth',2,'MarkerSize',12);
set(gca,'FontSize',18);
xlabel('step size, h');
ylabel('error');
title('Forward Difference error vs. step size');

toc

saveas(1,'fwdDiff_linearPlot.png');
saveas(2,'fwdDiff_logPlot.png');
```





Note : If error $\approx Ch^p$, then $\log(\text{error}) = \log C + p \log h$, *i.e.* the slope of the data on a log-log plot gives the order of convergence.

$\triangle$

Question 10: Why does error increase for very small h (the left side of the plot)?

- $D_+f(x)$ has two sources of error: truncation error due to not using the entire Taylor series, and roundoff error due to finite precision arithmetic.
- Truncation error, we have shown, is $O(h)$, and roundoff error is $O(\epsilon/h)$, where $\epsilon \approx 10^{-15}$ in Matlab.
- The total error is therefore $O(h)+O(\epsilon/h)$, hence for large h (relative to ϵ) truncation error dominates the computation, but for small h, roundoff error is dominant.

$\triangle$

Note : Other finite difference approximations of first derivatives are possible, for example,

- Backward difference: $D_-f(x) = \dfrac{f(x) - f(x-h)}{h}$.
- Centered difference: $D_0f(x) = \dfrac{f(x+h) - f(x-h)}{2h}$ (homework).

4 Matlab Intro

basic data type = complex matrix, double precision

- Loops
- preallocation
- the colon operator
- elemental vs. array operations
- transpose
- clear, mod, if, elseif, end