Chapter 1: Computer Arithmetic

Contents

1 Introduction to Numerical Computation

1.1 Numerical Methods

They are **algorithms** that compute **approximations** to functions, their derivatives, their integrations, and solutions to various equations etc. Such algorithms could be programmed on a computer.

Numerical methods are not about numbers. It is about mathematical ideas and insight. A little idea can go a long way. Some classical problems:

- Development of algoritms
- Implementation
- Some analysis, including error-estimates, convergence, stability, etc.

Matlab

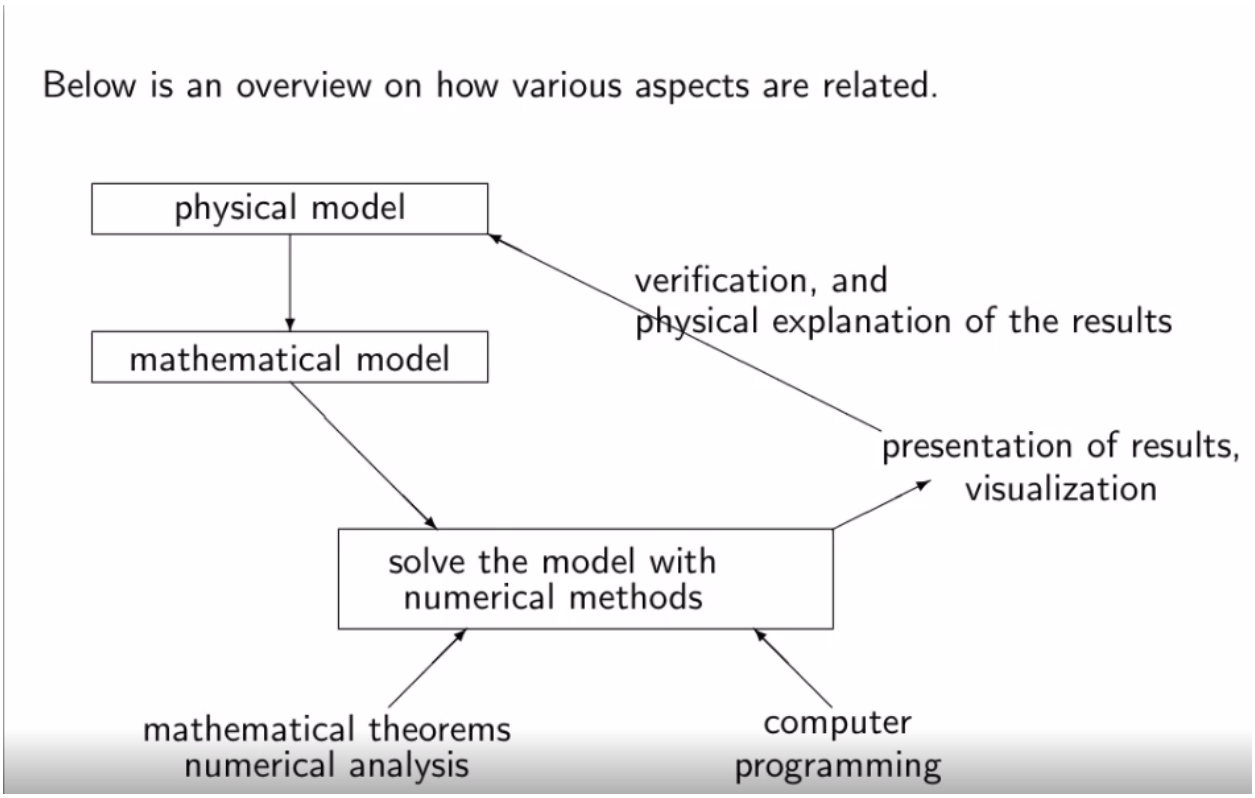

Figure 1:

2 Representation of numbers in different bases

There have been several different ways of representing numbers:

- 10: decimal, daily use
- 2: binary, computer use
- 8: octal
- 16: hexadecimal, ancient China
- 20: vigesimal, ancient France (can be seen in french language)
- 60: sexagesimal, used by Babylonians

In principle we can use any number β as the base. Writing such a number with a decimal point gives us an integer part and fraciton part.

This above formula allows us to convert a number in any base β into base 10.

2.1 Examples

2.1.1 Octal to Decimal

We have $(45.12)_8$.

$$4 * 8^1 + 5 * 8^0 + 1 * 8^{-1} + 2 * 8^{-2} = (37.15625)_{10}$$

$$
\begin{aligned}
& \overbrace{\left(\overbrace{a_n a_{n-1} \cdots a_1 a_0}^{\text{integer part}} . \overbrace{b_1 b_2 b_3 \cdots}^{\text{fractional part}}\right)_\beta} \\
= \quad & a_n \beta^n + a_{n-1} \beta^{n-1} + \cdots + a_1 \beta + a_0 \qquad \text{(integer part)} \\
& + b_1 \beta^{-1} + b_2 \beta^{-2} + b_3 \beta^{-3} + \cdots \qquad \text{(fractonal part)}
\end{aligned}
$$

Figure 2:

$$
\begin{aligned}
(1)_8 &= (1)_2 \\
(2)_8 &= (10)_2 \\
(3)_8 &= (11)_2 \\
(4)_8 &= (100)_2 \\
(5)_8 &= (101)_2 \\
(6)_8 &= (110)_2 \\
(7)_8 &= (111)_2 \\
(10)_8 &= (1000)_2
\end{aligned}
$$

Figure 3:

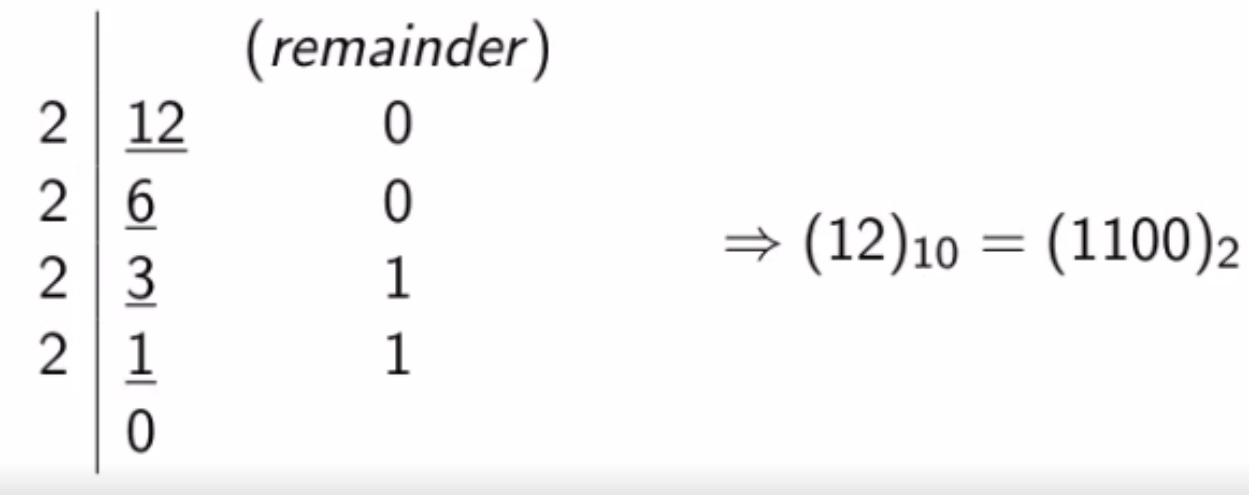

Figure 4:

2.1.2 Octal to Binary

Since 8 is a power of 2 it makes it very easy to convert from octal to binary:

$$(5034)_8 = (101000011100)_2$$

Each digit in octal gets converted to 3 digits in binary. And vice versa...

You can also go back from binary to octal:

$$(110010111001)_2 = (6271)_8$$

2.1.3 Decimal to Binary

Write $(12.45)_{10}$ in binary. (base 2) This is particularly interesting because we use decimal and computer uses binary. This conversion takes two steps. First we convert the integer part into binary.

Procedure: Divide the integer by 2 and store the remainder of each step until integer is zero. (Euclid's algorithm.)

Now to convert the fractional part to binary you multiply by 2 and store the integer part for the result.

Note that the fractional part here is not finite. Putting them together:

$$(12.45)_{10} = (1100.01110011001100...)_2$$

So how do we store this kind of a number in a computer?

3 Floating Point Representation

Recall normalized scientifi notation for a real number x:

- Decimal: $x = \pm r * 10^n$, where $1 \leq r < 10$
- Binary: $x = \pm r * 2^n$, where $1 \leq r < 2$
- Octal: $x = \pm r * 8^n$ where $1 \leq r < 8$

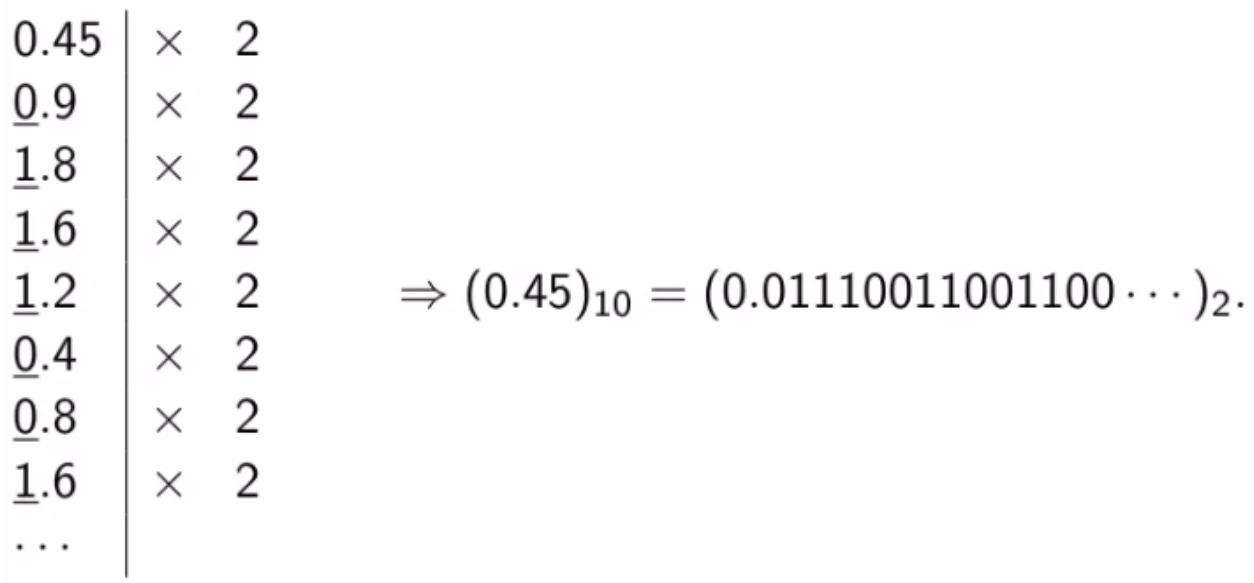

Figure 5:

So for any base β: $x = \pm r * \beta^n$ where $1 \leq r < \beta$

Information to be stored:

1. The sign
2. The exponent n
3. The value of r

The computer uses the binary version of the number system. They represent numbers with finite lenght. These are called machine numbers.

r: the normalized mantissa. For binary numbers we have

r = 1.(fractional part)

Therfore, in the computer we will only store the fractional part of the number

n: exponent. If n > 0, then x > 1, but if n < 0, then x > 1.

s: The sign of the number. If s = 0, it is positive and if s = 1, it is negative.

Each bit can store the value of either 0 or 1.

3.1 Single precision IEEE standard floating-point, in a 32-bit computer

The exponent: $2^8 = 256$. It can represent numbers from -127 to 128. The value of the number (in exponential notation):

$$(-1)^s * 2^{c-127} * (1.f)_2$$

The smallest representable number in absolute value is:

$$x_{min} = 2^{-127} \approx 5.9 * 10^{-39}$$

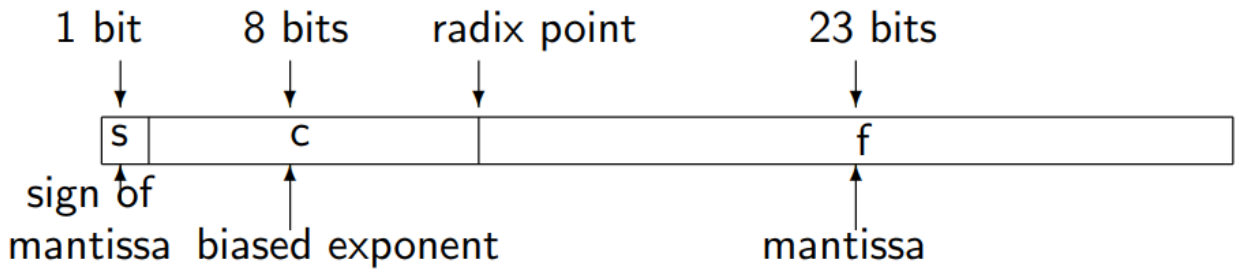

Figure 6:

The largest representable number in absolute value is:

$$x_{max} = 2^{128} \approx 2.4 * 10^{38}$$

Computers can only handle numbers with absolute values between x_{min} and x_{max}.

We say that x underflows if $|x| < x_{min}$ In this case we consider x = 0. We say that x overflows if $|x| > xmax$ In this case we consdier x to be infinite.

Let fl(x) denote the floating point representation of the number x. In gener it contains error (roundoff or chopping).

$$fl(x) = x * (1 + \delta)$$

Relative Error: $\delta = \frac{fl(x)-x}{x}$ Absolute Error: $fl(x) - x = \delta * x$

Computer errors in representing numbers:

- Relative error in rounding off: $\delta \leq 0.5 * 2^{-23} \approx 0.6 * 10^{-7}$
- Relative error in chopping: $\delta \leq 1 * 2^{-23} \approx 1.2 * 10^{-7}$

3.2 Error Propogation (through arithmetic operation)

Example 1: Consider an addition, say z = x + y, done in a computer. How would the errors be propagated?

Let x > 0, y > 0, and let fl(x), fl(y) be their floating point representation.

$$fl(x) = x(1 + \delta_x), fl(y) = y(1 + \delta_y)$$

where δ_x, δ_y are the relative errors in x,y. Then...

$$\begin{aligned} fl(z) &= fl(fl(x) + fl(y)) \\ &= (x(1 + \delta_x) + y(1 + \delta_y)) + \delta_z \\ &= (x + y) + x(\delta_x + \delta_z) + y(\delta_y + \delta_z) + (x\delta_x\delta_z + y\delta_y\delta_z) \\ &\approx (x + y) + x(\delta_x + \delta_z) + y(\delta_y + \delta_z) \end{aligned}$$

Here δ_z is the round-off error for z.

Then we have

$$\begin{aligned}\text{absolute error} &= \mathrm{fl}(z) - (x+y) = x\cdot(\delta_x+\delta_z) + y\cdot(\delta_y+\delta_z)\\ &= \underbrace{x\cdot\delta_x}_{\text{abs. err. for } x} + \underbrace{y\cdot\delta_y}_{\text{abs. err. for } y} + \underbrace{(x+y)\cdot\delta_z}_{\text{round off err}}\end{aligned}$$

$$\text{relative error} = \frac{\mathrm{fl}(z)-(x+y)}{x+y} = \underbrace{\frac{x\delta_x+y\delta_y}{x+y}}_{\text{propagated err}} + \underbrace{\delta_z}_{\text{round off err}}$$

Figure 7:

4 Loss of Significance

Loss of significance typically happens when one gets too few significant digits in subtraction of two numbers very close to each other.

Let's say we have the number 1.2345678 having 8 significant digits with a second number 1.2344444 with 8 significant digits

$$1.2345678 - 1.2344444 = 0.0001234$$

We lose 4 significant digits here. This error will be propogated in future computations. When designing an algorithm we want to try to avoid this.

4.1 Examples

Find the roots of x^2 - 40x + 2 = 0. Use 4 significant digits in the computation. **Answer** The roots for the equation ax^2 + bx + c = 0 are

$$r_{1,2} = \frac{1}{2a}(-b \pm \sqrt{b^2-4ac})$$

In our case we have

$$x_{1,2} = 20 \pm \sqrt{398} \approx 20.00 \pm 19.95$$

so

$$x_1 \approx 20 + 19.95 = 39.95$$

x_1 has four significant digits, so it is okay, but...

$$x_2 \approx 20 - 19.95 = 0.95$$

Here we lost 3 whole significant digits!

To avoid this we should change our algorithm. Observe that $x_1 x_2 = c/a$. Then

$$x_2 = \frac{c}{ax_1} = \frac{2}{39.95} \approx 0.05006$$

And now we have four significant digits again.

Compute the function:

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

in a computer. Explain what problem you might run into in certain cases. Find a way to fix the difficulty.

Answer We see that, for large values of x with $x > 0$, the avlues $\sqrt{x^2 + 2x}$ and $x + 1$ are very close to each other. Therefore, in subtraction we will lose many significant digits. To avoid this problem, we maanipulate the function f(x) into an equivalent one that does not perform the subtraction. This can be achieved by multiplying both numerator and denominator by the conjugate of the denominator.

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

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

5 Review of Taylor Series

Given that $f(x) \in C^\infty$ is a smooth function. Its Taylor expansion about the point x = c is:

$$f(x) = f(c) + f'(x - c) + \frac{1}{2!} f''(c)(x - c)^2 + \frac{1}{3!} f'''(c)(x - c)^3 + ...$$

$$f(x) = \Sigma \frac{1}{k!} f^{(k)}(c)(x - c)^k$$

This is called the Taylor series of f at the point c.

If c = 0, this is the MacLaurin series:

$$f(x) = f(0) + f'(0)x + \frac{1}{2!} f''(0)x^2 + \frac{1}{3!} f'''(0)x^3 + ... = \Sigma \frac{1}{k!} f^{(k)}(0)x^k$$

Familiar Exmpales of MacLaurin Series:

Since the computer can only perform algebraic operation, these series are actually how a computer calculates "fancier" functions. For example, the exponential function is calculated as:

$$e^x \approx \Sigma \frac{x^k}{k!}$$

for some large integer N such that the error is sufficiently small.

Note that this is rather expensive in computing time. This is why many algorithm we take care in doing fewer function evaluations.

$$
\begin{aligned}
e^x &= \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad |x| < \infty \\
\sin x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \qquad |x| < \infty \\
\cos x &= \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \qquad |x| < \infty \\
\frac{1}{1-x} &= \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + x^3 + x^4 + \cdots, \qquad |x| < 1
\end{aligned}
$$

Figure 8:

5.1 Example

Compute e to 6 digit accuracy **Answer** We have

$$e = e^1 = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + ...$$

$$\frac{1}{2!} = 0.5$$

$$\frac{1}{3!} = 0.166667$$

$$\frac{1}{4!} = 0.041667$$

...

$$\frac{1}{9!} = 0.0000027$$

so...

$$e = 1 + 1 + 0.5 + 0.166667 + 0.041667 + ... + 0.0000027 = 2.71828$$

5.2 Error and Convergence

Assume $f^{(k)}(x)(0 \leq k \leq n)$ are continous functions. The partial sum is:

$$f_n(x) = \Sigma \frac{1}{k!} f^{(k)}(c)(x-c)^k$$

Taylor Theorem:

$$E_{n+1} = f(x) - f_n(x) = \Sigma \frac{1}{k!} f^{(k)}(c)(x-c)^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-c)^{n+1}$$

where ξ is some value between x and c.

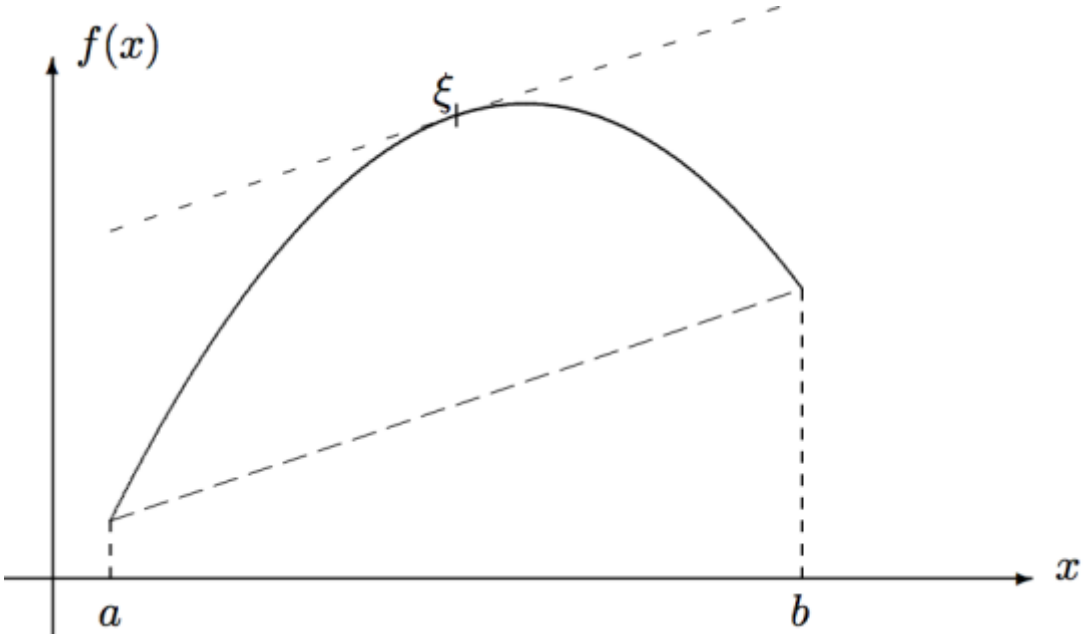

Figure 9:

This says, for the infinite sum for the error, if it converges, then the sum is "dominated" by the first term in its series.

Observation: A Taylor Series convergence rapidly if x is near c, and slowly (or not at all) if x is far away from c.

Geometric interpretation for the error estimate with n = 0.

$$f(b) - f(a) = (b - a)f'(\xi)$$

for some $\xi \in (a, b)$

This implies

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}$$

know as the mean value theorem.

6 Finite Difference Approximation

Given h > 0 is sufficiently small, we have 3 was of approximating f'(x):

1. Forward Euler: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$
2. Backward Euler: $f'(x) \approx \frac{f(x)-f(x-h)}{h}$
3. Central Finite Difference: $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$

For the second derivative f", we also have a central finite difference approximation:

$$f''(x) \approx \frac{1}{h^2}(f(x + h) - 2f(x) + f(x - h))$$

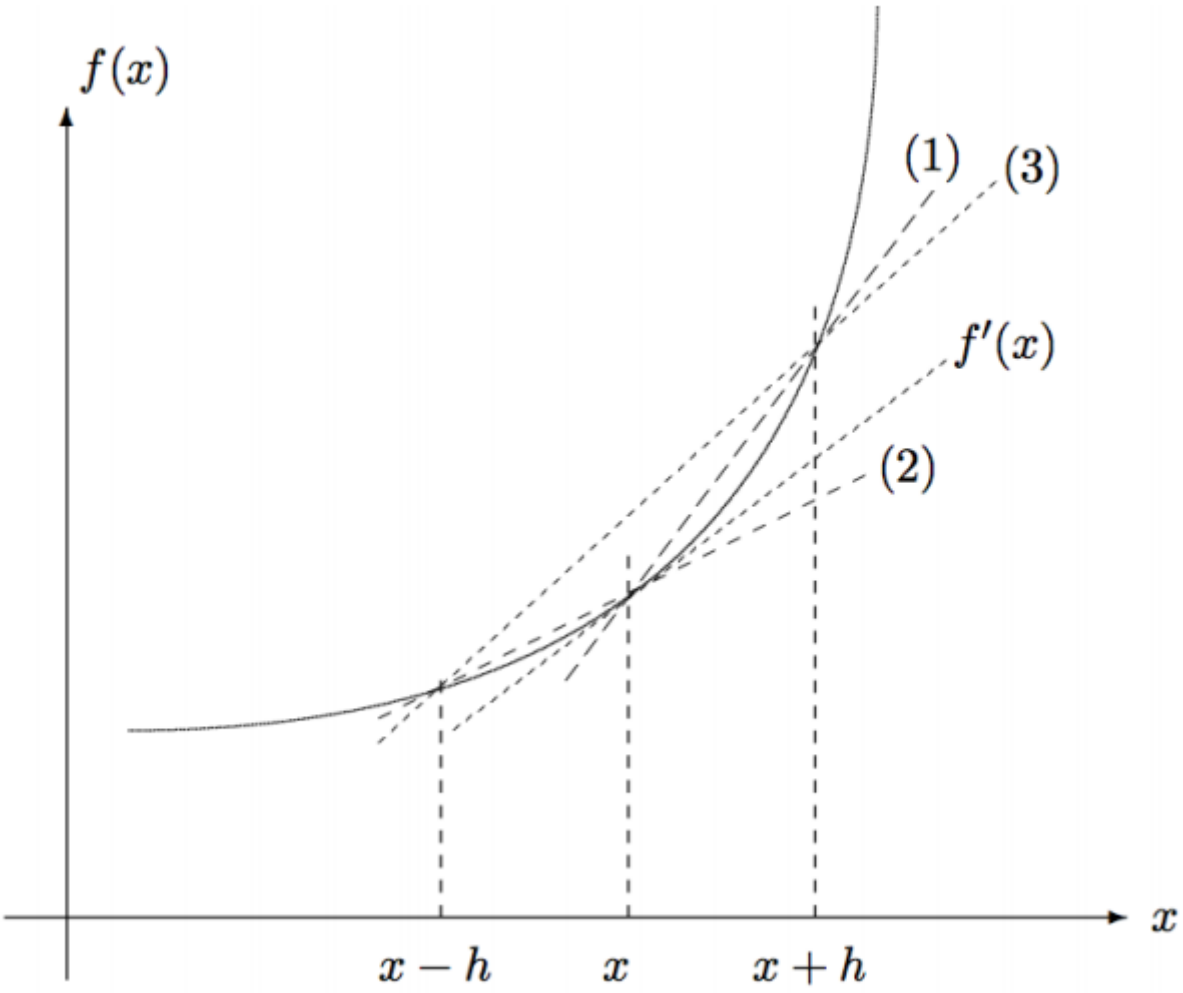

Figure 10:

$$
\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4), \\
f(x-h) &= f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{6}h^3 f'''(x) + \mathcal{O}(h^4).
\end{aligned}
$$

Forward Euler:

$$
\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^1), \quad (1^{st}\text{order}),
$$

Backward Euler:

$$
\frac{f(x) - f(x-h)}{h} = f'(x) - \frac{1}{2}hf''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^1), \quad (1^{st}\text{order}),
$$

Figure 11:

6.1 Local Truncation Error

The Taylor Expansion for f(x + h) about x:

$$
f(x+h) = \Sigma \frac{1}{k!} f^{(k)}(x)(h)^k = \Sigma \frac{1}{k!} f^{(k)}(x)(h)^k + E_{n+1}
$$

where

$$
E_{n+1} = \Sigma \frac{1}{k!} f^{(k)}(x)(h)^k = \frac{1}{(n+1)!} f^{(n+1)}(\xi) h^{n+1} = O(h^{n+1})
$$

Here the notation $O(h^4)$ indicate a quality bounded by Ch^4 where C is a bounded constant.

Central finite difference:

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x) - \frac{1}{6}h^2 f'''(x) + \mathcal{O}(h^2) = f'(x) + \mathcal{O}(h^2), \quad (2^{nd}\text{order}),$$

Central finite difference for the second derivative

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2} = f''(x) + \frac{1}{12}h^2 f^{(4)}(x) + \mathcal{O}(h^4) = f''(x) + \mathcal{O}(h^2),$$

second order method.

Figure 12: