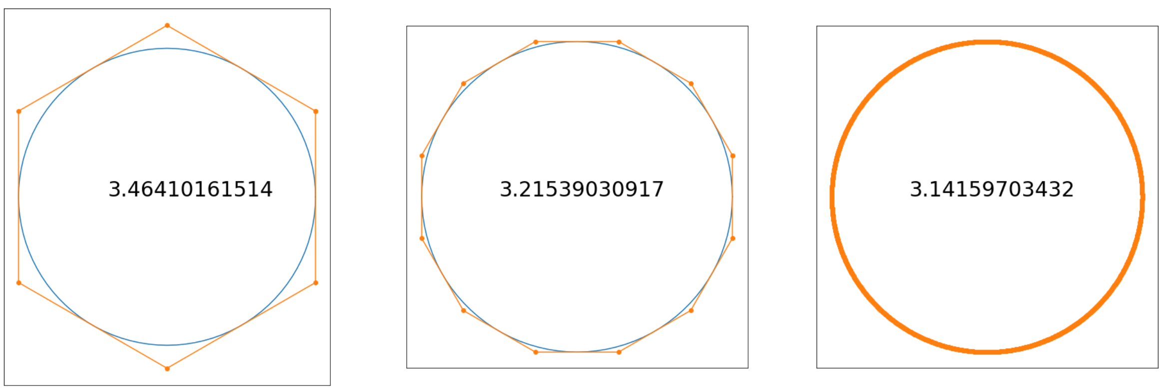
Bachelor of Ecole Polytechnique
MAA106: Introduction to Numerical Analysis, year 1, semes
Maxime Breden

Introduction to Computational Mathematics




```python
## loading python libraries

# necessary to display plots inli
%matplotlib inline

# load the libraries
import matplotlib.pyplot as plt #
plt.rcParams.update({'font.size':
import numpy as np               #

```

```python
from math import *
```

Introduction

We will deal with two different types of approximation

- Mathematical approximation, or truncation errors.

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

- Approximations made by the computer, or rounding errors.

```python
x = sqrt(2)
y = x*x
print(y)
```

```
2.0000000000000004
```

Definition. If x is an approximation of x^*,

- the absolute error between x and x^* is $|x - x^*|$
- the relative error between x and x^* is $\frac{|x - x^*|}{|x^*|}$.

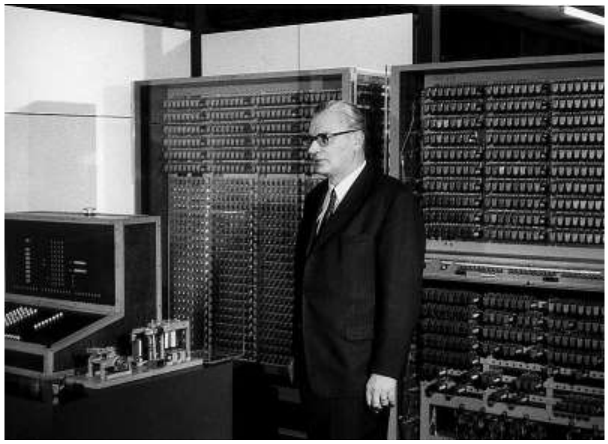
Example. Take

$$x = 11, \; x^* = 10 \quad \text{and} \quad y = 1001, \; y^* = 1000.$$

In both cases, the absolute error is 1, but one might consider that y approximates y^* more accurately than x does x^*. This is reflected in the relative error:

$$\frac{|x - x^*|}{|x^*|} = 0.1 \qquad \frac{|y - y^*|}{|y^*|} = 0.001.$$

Machine representation of numbers: rounding errors



Konrad Zuse (1910-1995) and the Z3-computer (1941).

```
x = sqrt(2)
y = x**2
print(y)
```

```
2.0000000000000004
```

Machine representation of numbers

Example. The decimal system (or base-10 system).

$$[1000]_{10} = 1 \times 10^3$$

$$[6743.7]_{10} = 6 \times 10^3 + 7 \times 10^2 + 4 \times 10^1 + 3 \times 10^0 + 7 \times 10^{-1}$$

Example. The binary system (or base-2 system).

$$[1000]_2 = 1 \times 2^3$$

$[1011.1]_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \quad (= [11.5]_{10}$

Definition.
Normalized exponent representation in base 10.

A real number x can be written as

$$x = (-1)^{s_a} \times [0.a]_{10} \times 10^{(-1)^{s_b}\ [b]_{10}}$$

where the first digit of a is different from 0, and b is an integer.

Example.

$$-0.0047258 = -0.47258 \times 10^{-2} = (-1)^{1} \times 0.47258 \times 10^{(-1)^{1} \times 2}$$

Example.

$$\frac{10}{3} = 3.3333333\ldots = (-1)^{0} \times 0.33333333\ldots \times 10^{(-1)^{0} \times 1}$$

Definition.
Machine numbers.

Let m and n be given integers. A *machine number* is a real number x as above, with the length of a and b respectively lower than m and n. It can be exaclty represented by the follwing word of size $N = n + m + 2$:

$$| \quad s_a \quad | \quad a \quad | \quad s_b \quad | \quad b \quad |$$

Example. Take $(m, n) = (5, 2)$

$$-0.0047258 = (-1)^{1} \times 0.47258 \times 10^{(-1)^{1} \times 2} \quad \text{is a machine number}$$

$$| \quad 1 \quad | \quad 47258 \quad | \quad 1 \quad | \quad 02 \quad |$$

Example. Take $(m, n) = (5, 2)$

$$\frac{10}{3} = (-1)^{0} \times 0.33333333\ldots \times 10^{(-1)^{0}\times 1} \quad \text{is NOT a machine number}$$

$$|\quad 0 \quad|\quad 3333333\ldots \quad|\quad 0 \quad|\quad 01 \quad|$$

Definition. Floating-point representation and rounding error.

- $rd(x)$ is the closest machine number to x, called *floating-point* representation of x.
- We call $\frac{|x-rd(x)|}{|x|}$ the rounding error.
- The smallest ε such that $\frac{|x-rd(x)|}{|x|} \leq \varepsilon \; \forall \; x$ is called the *machine precision*.

Example. Consider $m = 5$, $n = 2$, and $\pi = 3.14159265\ldots$.

$$rd(\pi) = 0.31416 \times 10^{1} \quad \Rightarrow \quad |\quad 0 \quad|\quad 31416 \quad|\quad 0 \quad|\quad 01 \quad|$$

$$\left|\frac{\pi - rd(\pi)}{\pi}\right| \approx 2.34 \times 10^{-6}$$

```python
print(pi)
```

```
3.141592653589793
```

```python
x = 0.1111111111111111   # 16 ones
y = 0.11111111111111111  # 17 ones
print('Is x equal to y?', x==y)
```

```
Is x equal to y? True
```

```python
z = 0.111111111111111    # 15 ones
```

```
print('Is x equal to z?', x==z)
```

```
Is x equal to z? False
```

```
lie = 0.1
print(lie)
```

```
0.1
```

Remark.

$$[0.1]_{10} = [0.\underbrace{110011001100\ldots}_{\infty \text{ many times } 1100}]_2 \times 2^{-3}$$

$$\text{lie} = rd([0.1]_{10}) = [0.\underbrace{110011001100\ldots}_{\text{only 52 digits}}]_2 \times 2^{-3}$$

Floating-point arithmetic

Example.

Are the following numbers machine numbers for $m = 5$ and $n = 2$?

- $x = 3.1416$
- $y = 0.00011$
- $s = x + y$

Answer.

- $x = 0.31416 \times 10^{01}$ ☑
- $y = 0.11000 \times 10^{-03}$ ☑
- $s = x + y = 0.314171 \times 10^{01}$ ☒

Definition.
Floating-point arithmetic (finite-digits arithmetic)

$$x \oplus y = rd(\, rd(x) + rd(y)\,), \qquad x \ominus y = rd(\, rd(x) - rd(y)\,)$$
$$x \otimes y = rd(\, rd(x) \times rd(y)\,), \qquad x \oslash y = rd(\, rd(x) \,/\, rd(y)\,)$$

Example. For $m = 5$ and $n = 2$.

- $x = 3.1416$
- $y = 0.00011$
- $s = x + y = 3.14171$
- $s_{num} = x \oplus y = rd(x + y) = 3.1417$

$$\frac{|s - s_{num}|}{|s|} \approx 3.18 \times 10^{-6}$$

Example.
Addition of a large and a small number.

- $x = 1/3$
- $y = 6/7 \times 10^4$
- $s = x + y = 8571.7619 \dots$

Answer. For $m = 5$ and $n = 2$.

$$rd\,(x) = 0.33333 \times 10^0$$
$$rd\,(y) = 0.85714 \times 10^4$$

and

$$\begin{aligned} x \oplus y &= rd(0.33333 \times 10^0 + 0.85714 \times 10^4) \\ &= rd(0.85717{\color{red}3333} \times 10^4) \\ &= 8571.7 \end{aligned}$$

Example.
Subtraction of nearly equal numbers.

- $y = 6/7 \times 10^4$
- $z = 0.85717 \times 10^4$
- $t = z - y = 0.2714285 \ldots$

Answer. For $m = 5$ and $n = 2$.

$$\begin{aligned} rd\,(y) &= 0.85714 \times 10^4 \\ rd\,(z) &= 0.85717 \times 10^4 \end{aligned}$$

and

$$\begin{aligned} z \ominus y &= rd(0.85717 \times 10^4 - 0.85714 \times 10^4) \\ &= 0.3 \end{aligned}$$

The relative error with the exact value is quite large $\approx 10\%$

Example. The order of operations matters for floating-point arithmetic!

- $x = 1/3$
- $y = 6/7 \times 10^4$
- $s = x + y - y = 0.333333 \ldots$

Answer.

$$\begin{aligned} s_1 &= {\color{red}(x \oplus y) \ominus y} \\ &= 0.85717 \times 10^4 \ominus y \\ &= 0.3 \qquad\qquad (\text{relative error} \approx 10\%) \end{aligned}$$

Answer.

$$\begin{aligned} s_2 &= x \oplus (y \ominus y) \\ &= 0.33333 \ominus 0 \\ &= 0.33333 \qquad\qquad (\text{relative error} \approx 0.001\%) \end{aligned}$$

```python
## x + y - x = y ?

x = 10**15
y = 0.1
s1 = (x + y) - x
s2 = (x - x) + y
s = y
print('s1 =',s1)
print('s2 =',s2)
print('The "exact" answer s =',s)
```

```
s1 = 0.125
s2 = 0.1
The "exact" answer s =
0.1
```

Mathematical approximations: truncation error

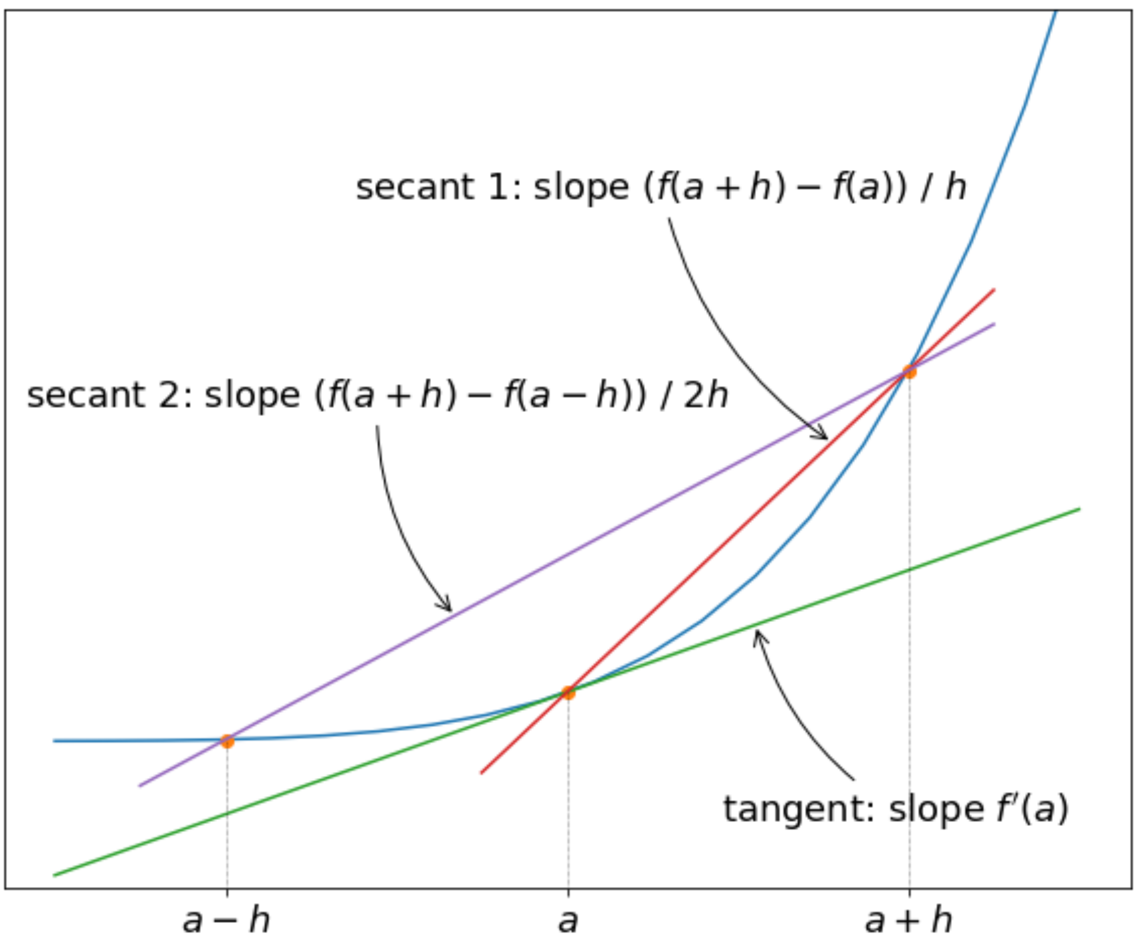
- We ignore rounding errors for the moment and assume that the computer can compute with real numbers exactly.
- In most problems, there is still an important issue: there is no explicit and computable formula for the solution.

- Therefore, we replace the initial problem by an easier one, for which we can actually compute the solution.
- This leads to mathematical approximations, or truncation errors, that also have to be controlled.

Example. Approximate computation of a derivative:

- f a given derivable function, but no formula known for f'.
- a a given real number.

How to approximate $x^* = f'(a)$?



Algorithm. Two options:

$$x_h = \frac{f(a+h) - f(a)}{h} \qquad \text{and} \qquad \bar{x}_h = \frac{f(a+h) - f(a-h)}{2h}.$$

Definition.
Truncation error.

$$e_h = \left| f'(a) - x_h \right| \qquad \text{and} \qquad \bar{e}_h = \left| f'(a) - \bar{x}_h \right|.$$

Questions.

- Does each algorithm converge, meaning that $e_h, \bar{e}_h \xrightarrow[h \to 0]{} 0$?
- If yes, can we predict the *rate* or *speed* at which they converge ?
- Can we make some quantitative prediction about the accuracy for a given h ?

```python
def f(x):
    return x**5

def ApproxDerivative1(f, a, h):
    return (f(a+h) - f(a))/h

def ApproxDerivative2(f, a, h):
    return (f(a+h) - f(a-h))/(2*h
```

```python
a = 1
h = 0.000000000001
x1 = ApproxDerivative1(f, a, h)
x2 = ApproxDerivative2(f, a, h)
print('a =', a, ', h =', h, ', exa
print('First algorithm:', x1)
print('Second algorithm:', x2)
```

```
a = 1 , h = 1e-12 , ex
act value: f'(1)=5
First algorithm: 5.000
444502911705
Second algorithm: 5.00
```

Questions.

- It looks like $e_h, \bar{e}_h \xrightarrow[h\to 0]{} 0$. Can we prove it? Under which assumptions?
- It also looks like $\bar{e}_h$ goes to 0 faster than e_h does. Can we also prove that?

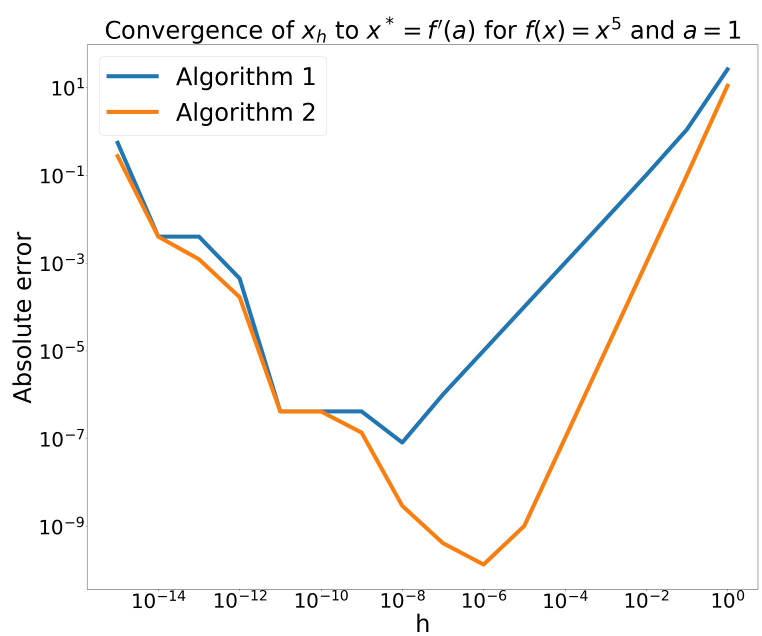
Remark.
Usually, it is hard to exactly compute the error e_h. In practice, one often tries to find an error estimator β_h such that

- $e_h \leq \beta_h$,
- $\beta_h \xrightarrow[h\to 0]{} 0$.

Total numerical error

```
a = 1

n = np.arange(16)
h = 10.**(-n) #sequence of h = [1

Der1 = ApproxDerivative1(f, a, h)
Err1 = abs(Der1 - 5.)
```

```
Der2 = ApproxDerivative2(f, a, h)
Err2 = abs(Der2 - 5.)

# plot of the errors versus h
fig = plt.figure(figsize=(30, 25))
plt.loglog(h, Err1, linewidth=10,
plt.loglog(h, Err2, linewidth=10,
plt.legend(loc='upper left', fonts
plt.xlabel('h', fontsize=60)
plt.ylabel('Absolute error', fonts
plt.title('Convergence of $x_h$ to

plt.show()
```




```
# execute this part to modify the
from IPython.core.display import
```

```
def css_styling():
    styles = open("./style/custom_
    return HTML(styles)
css_styling()
```

```

```