Chapter 1: Mathematical Preliminaries

In Chapter 1, we mainly focus on the following contents:

- Precision – read section 1.1 page 2–7.
- Efficiency, Horner's algorithm – read section 1.1 page 8–9.
- Taylor series – read section 1.2 page 20–23 and 25–28.
- Floating-point representation – read section 1.3 until example 2 in page 44.
- Loss of significance - read section 1.4 until page 62.

More information on floating-point representation and rounding errors:

- 02635 Mathematical software programming.

Notations: sum and product (page 9)

$$\sum_{k=n}^{m} x_k = x_n + x_{n+1} + x_{n+2} + x_{n+3} + \cdots + x_{m-1} + x_m$$

$$\prod_{k=n}^{m} x_k = x_n \cdot x_{n+1} \cdot x_{n+2} \cdot x_{n+3} \cdots x_{m-1} \cdot x_m$$

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1} + a_n x^n \ .$$

Taylor's Theorem (page 25) For a function f with continuous derivatives of orders $0, 1, 2, \ldots, (n+1)$ in $[a, b]$, then for any c and x in $[a, b]$:

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{k!} (x-c)^k + E_{n+1} \qquad f^{(k)} = k\text{th derivative}$$

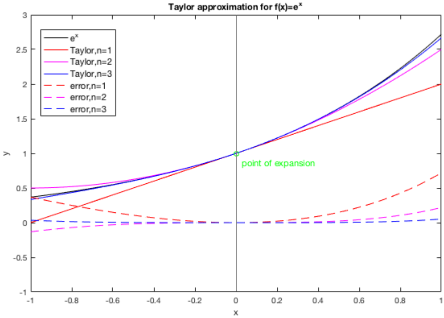
$$E_{n+1} = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-c)^{n+1} \qquad \text{where } \xi \text{ lies between } c \text{ and } x.$$

Example: Taylor series

$$\boxed{f(x) = e^x} \Rightarrow f'(x) = e^x,\ f''(x) = e^x,\ f^{(3)} = e^x, \text{ etc.}$$

Taylor series at $c = 0$ with $f(c) = e^0 = 1$:

$$f(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \cdots$$



Taylor series will be used in the week 2, 3, 4, 5, 6, 7, 8, 9, 10, 13.

Efficiency: Horner's algorithm (page 8)

Following mathematical expressions is not always the most *efficient* way to do the calculation. For example, to evaluate the polynomial:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

Require n additions and $\frac{1}{2}n(n+1)$ multiplications.

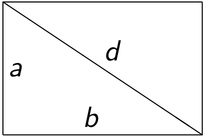
$$p(x) = a_0 + x\Big(a_1 + x\big(a_2 + \cdots + x(a_{n-1} + x\, a_n)\cdots\big)\Big)$$

Require n additions and only n multiplications.

% Calculate $p =$ the value of the polynomial at x
integer i, n; **real** p, x
real array $(a_i)_{0:n}$
$p \leftarrow a_n$
for $i = n - 1$ **to** 0
    $p \leftarrow a_i + x\, p$
end for

Significant Digits of Precision: Example (page 3–4)

We cut a 2m$\times$3m rectangular sheet into two equal triangular pieces, and the dimensions are accurate to 1mm. What is the diagonal measurement of each triangle?



$$a = 2.000 \pm 0.001$$

$$b = 3.000 \pm 0.001$$

$$d = \sqrt{a^2 + b^2}$$

There is *uncertainty* on the diagonal d:

$$\underline{3.60}42\ldots \leq d \leq \underline{3.60}69\ldots$$

We only can conclude that $d = 3.60$ and claim that d has

- 3 *significant digits*, and
- 2 *precise decimal places* or *significant decimal digits*.

We should always indicate the number of the significant digits/decimal digits.

Errors: Absolute and Relative (page 5–6)

Absolute error

When we say that the *absolute* error of $\bar{a}$ to a is ± 0.01, it means

$$a - 0.01 \leq \bar{a} \leq a + 0.01, \qquad \bar{a} = \text{an approximation.}$$

But is it a big error or a small error?

- If $a = 3.45$, then 0.01 is a small error.
- If $a = 0.0345$, then 0.01 is a really big error.

Relative error

When we say that the *relative* error to a is 0.01 (or 1%), it means

$$\frac{|a - \bar{a}|}{|a|} \leq 0.01 \ .$$

For practical reasons, the relative error is more meaningful and more commonly used.

Uncertainty: complicated example (inspired by Ex. 2)

$$0.1036\,x + 0.2122\,y = 0.4398$$
$$0.2081\,x + 0.4247\,y = 0.8981$$

```
>>> import numpy as np
>>> A = np.array([[0.1036, 0.2122],[0.2081, 0.4247]])
>>> b = np.array([0.4398, 0.8981])
>>> x = np.linalg.solve(A,b)
x =
23.7258
-9.5108
```

We perturb the 4th decimal place in b (error in `b1` $\approx 0.02\%$):

```
>>> b1 = np.array([0.4396, 0.8982])
>>> x1 = np.linalg.solve(A,b1)
x1 =
24.3897
-9.8359
```

It leads to a *big* perturbation in solution (error in `x1` $\approx 3\%$) $\Rightarrow$ Chapter 8.

Error: Rounding and Chopping (page 7)

We have seen that the calculated results are affected by errors in data:

- How the error is propagated from the data to the results depends on the problem that we solve – equation, interpolation, etc.

But there are also other sources which introduce errors:

- The computer is not a continuum, and only can accept finite format!

What will computers do?

- *Rounding*: reduce the number of significant digits in one digit.
- *Chopping*: discard all digits that follow the nth digit, but none of the remaining n digits are changed.

```
>>> import numpy as np
>>> n = 1000; r = np.random.rand(n,1)
>>> d = [abs(1-(ri*(1/ri))) for ri in r]
>>> np.count_nonzero(d) # number of non-zero elements
144
>>> max(d)
1.11022302e-16
```

Floating-Point Representation

In the decimal system, a real number x can be represented as:

$$x = \pm \overbrace{d_0.d_1 d_2 d_3}^{\text{mantissa}} \ldots \times 10^n, \quad d_i = \text{decimal digits } (0,1,\ldots,9), \quad n = \text{exponent}$$

In the binary system, x can be written as:

$$x = \pm 1.b_1 b_2 \ldots b_{52} \times 2^k, \quad b_i = \text{binary digits } (0,1), \quad k = \text{exponent}$$

Note that computers can only work with finite numbers!

The standard IEEE-754 has a word length of 64 bits:

- 1 bit for the sign $\pm$
- 52 bits for the mantissa $b_1, \ldots, b_{52}$
- 11 bits for the exponent, i.e., $-1022 \leq k \leq 1023$.
- Special values of k is reserved for $\pm$`Inf` and `NaN`.

Computer Errors in Representing Numbers

Since there are only 52 digits in the mantissa, errors can occur when we attempt to represent a real number x.

- We have to use the closest *machine number*, $\mathrm{fl}(x)$, replacing x.
- It would lead to the *roundoff error*, $x - \mathrm{fl}(x)$.
- It is calculated by *rounding* to obtain the last binary digit b_{52}.

The relative error of this representation is bounded by:

$$\left|\frac{x - \mathrm{fl}(x)}{x}\right| \leq 2^{-53} \approx 1.1102 \times 10^{-16} ,$$

where $u = 2^{-53}$ is **unit roundoff error**.

What is `np.finfo(float).eps` $\approx 2.2204 \times 10^{-16}$ in Python?

- It is the difference between 1 and the next larger number that can be stored in this machine, and we have $\texttt{eps} = 2u$. It is usually called as **machine epsilon** or **machine precision**.

Note that $\mathrm{fl}(1+u) = 1$, while $\mathrm{fl}(1+\texttt{eps}) \neq 1$.

Loss of Significance (page 57)

Assume that our calculator works with floating-point numbers that have 10 decimal digits in the decimal system. Set $x = \frac{1}{15}$ and calculate $z = x - \sin(x)$ in this calculator:

$$\begin{array}{lrl}
\text{input:} & x \leftarrow & 0.66666\,66667 \times 10^{-1} \\
\text{calculate:} & y = \sin(x) \leftarrow & 0.66617\,29492 \times 10^{-1} \\
\text{calculate:} & x - y = & 0.00049\,37175 \times 10^{-1} \\
\text{normalize:} & z \leftarrow & 0.49371\,75000 \times 10^{-4}
\end{array}$$

The last three 0s in z are supplied by the calculator and are not correct – they are spurious zeros.

This example shows one of the most common reasons for *loss of significance*, i.e., the subtraction of one quantity from another nearly equal quantity.

Example: Loss of significance

Don't try this at home !

Example: Loss of significance

Don't try this at home !

We use the following identity to show the loss of significance:

$$1^2 = (1 - x + x)^2 = \big((1 - x) + (x)\big)^2 = (1 - x)^2 + x^2 - 2 * x(x - 1)$$

Example: Loss of significance

Don't try this at home !

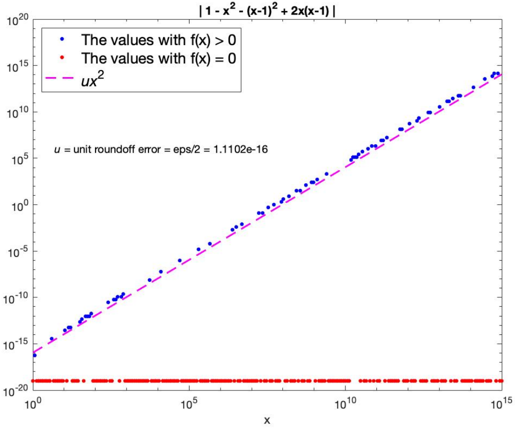
We use the following identity to show the loss of significance:

$$1^2 = (1 - x + x)^2 = \big((1 - x) + (x)\big)^2 = (1 - x)^2 + x^2 - 2 * x(x - 1)$$

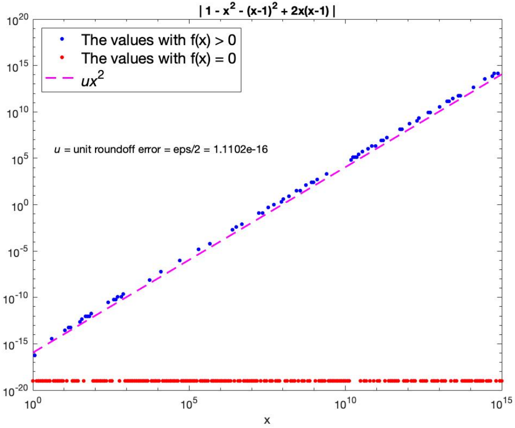
There is loss of significance in subtraction due to the x^2 terms, so the error would increase in the order of

$$c * x^2.$$

Example: Loss of significance



Example: Loss of significance



- Error becomes large, when x is large!
- The errors are oscillated!

Tricks to Avoid Loss of Significance (page 60)

Example: $f(x) = \sqrt{x^2+1} - 1$ with $x \ll 1$. Two tricks:

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

$$f(x) = \tfrac{1}{2}x^2 - \tfrac{1}{8}x^4 + O(x^6) \quad \text{Taylor-expension}$$

```
>>> x = np.array([10**(-p) for p in range(1,9+1)])
>>> np.array([x,np.sqrt(x**2+1)-1,x**2/(np.sqrt(x**2+1)+1),
0.5*x**2-x**4/8]).T
1.0000e-01 4.9876e-03 4.9876e-03 4.9875e-03
1.0000e-02 4.9999e-05 4.9999e-05 4.9999e-05
1.0000e-03 5.0000e-07 5.0000e-07 5.0000e-07
1.0000e-04 5.0000e-09 5.0000e-09 5.0000e-09
1.0000e-05 5.0000e-11 5.0000e-11 5.0000e-11
1.0000e-06 5.0004e-13 5.0000e-13 5.0000e-13
1.0000e-07 4.8850e-15 5.0000e-15 5.0000e-15
1.0000e-08          0 5.0000e-17 5.0000e-17
1.0000e-09          0 5.0000e-19 5.0000e-19
```

Quadratic Formula (not in the book)

If the quadratic equation $ax^2 + bx + c = 0$ has roots, they are given by the quadratic formula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad \text{and} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Potential loss of significance in subtraction: if $\sqrt{b^2 - 4ac} \approx |b|$, when we calculate $-b \pm \sqrt{b^2 - 4ac}$.

What to do: Calculate the root that avoids the subtraction first. Then, obtain the other root based on $x_1 x_2 = c/a$.

$$\begin{aligned} q &= -\frac{1}{2}\left(b + \operatorname{sign}(b)\sqrt{b^2 - 4ac}\right) \\ x_1 &= q/a \\ x_2 &= c/q \end{aligned}$$

Conclusion: Even very simple calculations can have danger of loss of significance. Good software takes this into account.