Summary:
- Problem: Evaluate $f(x)$ with given error bound ε
- Required: $f \in C^{n+1}$, values of derivatives $f^{(k)}(x)$
- Check interval of convergence; does Taylor series expansion work?
- Estimate the maximum error for n terms of Taylor polynomial
- Choose n such that the error bound ε is met.
- Evaluate Taylor poly. to get result

---

2. Number representations

Error types: A: Errors in data (partly because of round-off)

B: Round-off errors during computations. If two numbers with 5 significant digits are multiplied, the result will be 2s-1 or 2s significant digits; they might be rounded off.

C: Truncation errors (Discretization has finite accuracy)

Let $\tilde{a}$ be an approx. of a

Def.: $\tilde{a} - a$: absolute error

$\frac{(\tilde{a} - a)}{a}$: relative error

Error bound: magnitude of admissible error

Ex.:

0.00123 with error $0.000004 = 0.4 \cdot 10^{-5}$

Error is below $\frac{1}{2} \cdot 10^{-t}$ with $t = 5$. So there are 5 correct and 3 significant (i.e. non-zero non leading zero) digits. [corresponding to round off]

0.00123 with error $0.000006 = 0.06 \cdot 10^{-4}$

Error is below $\frac{1}{2} \cdot 10^{-4}$ but larger than $\frac{1}{2} \cdot 10^{-5}$.

So there are only 4 correct and 2 significant digits. [corresponding to rounding up].

i.e. the error is between $\pm \frac{1}{2} \cdot 10^{-t}$ with corresponding t.

Theorem: In addition/subtraction the bounds for absolute errors are added.

In division/multiplication the bounds for relative errors are added.

Ex.: Solve $x^2 - 56x + 1 = 0$

$x_1 = 28 - \sqrt{783}$ ← 5 digits

$\approx 28 - 27.982$

$= 0.018 \pm \frac{1}{2} \cdot 10^{-3}$

↑ 2 significant digits in answer for same number range $0.001 - 99.999$ (~~incl~~ + 0)

Error propagation: If $y(x)$ is a smooth (differentiable) function, $|y'(x)|$ can be interpreted as sensitivity of $y(x)$ to errors in x.

Generalized to functions of several variables we get

$$|\Delta y| \leq \sum_i \left|\frac{\partial y}{\partial x_i}\right| |\Delta x_i|$$

where $\Delta y = \tilde{y} - y$, $\Delta x_i = \tilde{x}_i - x_i$ for variables x_i, where $\sim$ is the approx.

Base representation: Let $b \in \mathbb{N} \setminus \{1\}$

Every number $x \in \mathbb{N}_0$ can be written as a unique expansion with respect to base b

$$x = a_0 b^0 + a_1 b^1 + \dots + a_n b^n = \sum_{i=0}^{n} a_i b^i$$

$a_i \in \mathbb{N}_0$, $a_i < b$, $a_i \in \{0, \dots, b-1\}$

b is base, a_i are the digits.

For real numbers we can write: $x \in \mathbb{R}$

$$x = \sum_{i=0}^{n} a_i b^i + \sum_{i=1}^{\infty} x_i b^{-i} = a_n \cdots a_0 \bullet x_1 x_2 \dots$$

Ex.: Base $b=10$: $37294 = 4 \cdot 10^0 + 9 \cdot 10^1 + 2 \cdot 10^2 + 7 \cdot 10^3 + 3 \cdot 10^4$

Base $b=2$: $1011 = 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3$

$= (1)_{10} + (2)_{10} + (0)_{10} + (8)_{10} = (11)_{10}$

↑ number with base 10.

Lecture 3 Numerical Methods, 11.02.2021

Ex.: Base $b=2$: $1011 = 1\cdot 2^0 + 1\cdot 2^1 + 0\cdot 2^2 + 1\cdot 2^3$

$= (1)_{10} + (2)_{10} + (8)_{10} = (11)_{10}$

↑ indication of base

There are algorithms that convert number systems.

Euclid's algorithm for $(x)_{10}$ to $(y)_b$

Euclid ~ 300 BC

1) Input $(x)_{10}$
2) Determine smallest n s.th. $x < b^{n+1}$
3) for $i = n$ to 0 do

$a_i := x \text{ div } b^i$ (integer division)

$x := x \bmod b^i$ (rest)

end for
4) output result $a_n a_{n-1} \dots a_0 = (x)_b$

Ex.: $(x)_{10} = (13)_{10} \rightarrow (x)_2 = ?$

Step 2): $n = 3$ since $13 < 2^4$

3) $i=3$: $a_3 = 13 \text{ div } 2^3 = 13 \text{ div } 8 = 1$

rest $x = 13 \bmod 2^3$

$= 13 \bmod 8 = 5$

next step

$i=2$: $a_2 = 5 \text{ div } 2^2 = 5 \text{ div } 4 = 1$

rest $x = 5 \bmod 4 = 1$

$i=1$: $a_1 = 1 \text{ div } 2^1 = 0$

rest $x = 1 \text{ div } 2 = 1$

$i=0$: $\quad a_0 = 1 \text{ div } 2^0 = 1 \text{ div } 1 = 1$

output $\quad (1101)_2 = (13)_{10}$

Two problems of Euclid's algorithm:

a) step 2 inefficient

b) division by large numbers b^i can be problematic

Better use Horner's scheme:

$$(a_n a_{n-1} \dots a_0)_b = a_0 + b\left(a_1 + b\left(a_2 + b\left(a_3 + \dots + b a_n\right)\dots\right)\right)$$

We can use algorithm:

1) Input $(x)_{10}$
2) $i := 0$
3) while $x > 0$ do
   $a_i := x \bmod b$ (rest of integer div)
   $x := x \text{ div } b$ (integer div)
   $i := i+1$
   end while
4) output $a_n a_{n-1} \dots a_0$

Horner
1786 - 1827

Some general remarks:

- Number with simple base rep. w.r.t. one base may be complicated w.r.t. another base

  e.g. $(0.1)_{10} = (0.000110011001100110\dots)_2$

- base 2 (binary), base 8 (octal), base 16 (hexadecimal)
- Conversion $b \to 10$: just perform computations

$$(42)_8 = 4 \cdot 8^1 + 2 \cdot 8^0 = (34)_{10}$$

- Conversion 2 and 8. Three consecutive bits represent one octal digit, e.g.

$$(551.624)_8 = (101\ 101\ 001.110\ 010\ 100)_2$$

- Conversion 2 and 16: Somelar, just four bits ~~one~~ to one hexadecimal digit.

- Horner's scheme algorithm does not need estimate of n and does not ~~da~~ need division by large numbers. Applicable to real numbers, but one needs a criterion to stop of the representation ~~sk~~ with need base is infinite.

On computers we have only finite precision (number of digits / bits)

Def.: Normalized floating point representation w.r.t. base b stores a number x as

$$x = 0.a_1 \dots a_k \cdot b^n$$

with $a_i \in \{0, 1, \dots, b-1\}$ are digits, k is called precision, n is called exponent, $a_1 \dots a_k$ is called mantissa, $a_1 \neq 0$. The fact $a_1 \neq 0$ is called ~~normal~~ normalization, it makes representation unique.

Ex.: base $b = 10$: 32.213

$$\rightarrow 0.32213 \cdot 10^2$$

• base $b=2$: $x = \pm 0.b_1 b_2 \dots b_n \cdot 2^n$, where $b_1 = 1$

↑ need one bit for sign.

↑ always 1, so you could drop this

For single precision, 4 bytes = 32 bits

1 bit sign mantissa
1 bit " exponent
7 bits for exponent (integer)
23 bits for mantissa (24 effectively)

7 bits largest number 127, i.e. $2^{127} \approx 10^{38}$

so range is $10^{-38} < |x| < 10^{38}$.

Since $2^{-24} \approx 10^{-7}$, we can represent 7 significant digits.

Better representation for example with double precision, (8 bytes)

for example.

We have some issues:

- adding numbers is commutative, i.e. $x+y = y+x$
- not associative, i.e. $(x+y)+z \neq x+(y+z)$ not always true.

Take $x = y = 0.00000033$, $z = 0.00000034$, $w = 1.000000$

we compute $x+y+z+w$ and on that order we get 1.000001. Reversed order gives 1.000000 with $b=10$ and 7 sig. digits

$0.3300000 \cdot 10^{-6} + 0.3300000 \cdot 10^{-6} + 0.3400000 \cdot 10^{-6}$

$+ 0.1000000 \cdot 10^{1}$

Add first three first you get 10^{-5} and they will count. Add last two first (and so on) contributions from small values will be rounded off.

Other example: $\sum_{1}^{10^7} 1 + 10^7 = 1 + \ldots + 1 + 10^7$

Order of summation will make a difference, either 10^7 or $2 \cdot 10^7$ etc., because $1+1=2$ but $1 + 10^7 = 10^7$ due to round off. We lose significant digits.

⇒ Avoid adding numbers of different orders of magnitude.
Add numbers in increasing order of their size.

Compute $x - \sin(x)$ for x close to 0, e.g. $x = \frac{1}{15}$

Assume $k = 10$ precision.

$x = 0.66666\ 66667 \cdot 10^{-1}$

$\sin(x) = 0.66617\ 29492 \cdot 10^{-1}$

$x - \sin(x) = 0.00049\ 37175 \cdot 10^{-1}$

$= 0.49371\ 75000 \cdot 10^{-4}$

lossy precision by 3 digits

⇒ **Avoid subtracting numbers of similar size because it leads to loss of precision**

Potential solution here: Taylor series expansion of

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} \pm \ldots$$

$$x - \sin(x) = + \frac{x^3}{3!} \mp \frac{x^5}{5!} \mp \ldots$$

Using just 3 terms we get: $0.49371\ 74328 \cdot 10^{-4}$

with error $\leq \underline{\underline{10^{-13}}}$ (which is needed for "full" precision at $0. \ldots \cdot 10^{-4}$)

Theorem: Let x, y be two normalized floating point numbers with $x > y > 0$ and base $b = 2$. If ~~those~~ there exist $p, q \in \mathbb{N}_0$ such that

$$2^{-p} \leq 1 - \frac{y}{x} \leq 2^{-q}$$

then at most p and at least q significant bits (digits at base 2) are lost during subtraction.

---

3 Linear systems of equations

Def.: A linear system of equations is given by

$$Ax = b, \quad A \in \mathbb{R}^{m \times n}, \quad x \in \mathbb{R}^n, \quad b \in \mathbb{R}^m, \text{ i.e.}$$

the matrix A has m rows and n columns, x is a vector with n unknowns, b has m entries, thus the system has m equations.

Since the degree of all x_i is equal to one it is a linear equation. If $n = m$ the system is called square. We can also write

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, \ldots, m$$

Ex.: ~~Que~~

$$a_{11} x_1 + a_{12} x_2 = b_1$$
$$a_{21} x_1 + a_{22} x_2 = b_2$$