2. Round-off and truncation errors

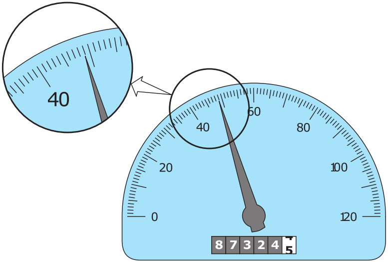
- significant figures
- numerical errors
- round-off errors
- Taylor series, truncation errors
- error propagation

Significant figures

significant figures indicate the reliability of a numerical value

- represent certain digits plus one estimated digit
- ensure confidence in computations

Example



speedometer (48.5 km/h, 3 significant figures with 2 certain digits)

odometer (87,324.45 km, 7 significant figures)

Rules for significant figures

- non-zero digits are always significant
- zeros between non-zero digits are significant
  - *e.g.*, 1002 has 4 significant figures
- leading zeros are not significant
  - *e.g.*, 0.001845 has 4 significant figures
- trailing zeros with a decimal point are significant
  - *e.g.*, 45.300 has 5 significant figures
- trailing zeros can be significant or not
  - *e.g.*, 45,300 may have 3, 4, 5 significant figures
  - use scientific notation for clarity
  - *e.g.*, 4.5300×10^4 has 5 significant figures
- computer retain only a finite number of significant figures
  - *e.g.*, $\pi = 3.14159265358979323846264 3\cdots$
  - π cannot be represented exactly in a computer
  - omission of the remaining significant figures is called round-off error

Accuracy and precision

Accuracy: how close a value is to the true value

Precision: how close repeated values are to each other

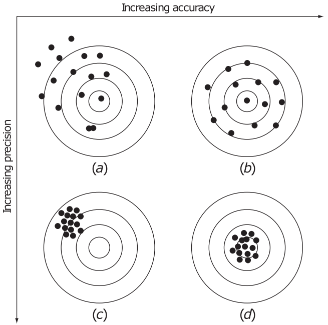
Example

(a) inaccurate (biased), imprecise (uncertain)

(b) accurate, imprecise

(c) inaccurate, precise

(d) accurate, precise



Outline

- significant figures
- **numerical errors**
- round-off errors
- Taylor series, truncation errors
- error propagation

Error sources

Errors in the problem to be solved

- mathematical model errors (model approximation)
- error in input data (physical measur. and previous approximate computation)

Truncation and discretization errors

- due to using approximate formula
  - replacing derivatives by finite differences
  - evaluating function by truncating a Taylor series (*e.g.*, $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$)
- *convergence errors* in iterative methods, which converge to the exact solution in infinitely many iterations, but are cut off after a finite number of iterations

Roundoff errors

- arise from finite precision representation of real numbers on computers
- truncation or discretization errors usually dominate roundoff errors

Example

surface area of the Earth with radius r might be computed using the formula

$$A = 4\pi r^2$$

- earth is modeled as a sphere, which is an approximation of its true shape
- $r \approx 6370\ \mathrm{km}$, is based on empirical measurements and previous computations
- π is given by an infinite limiting process, which must be truncated at some point
- numerical values for the input data, as well as the results of the arithmetic operations performed on them, are rounded in a computer or calculator

Approximations and errors

given true (actual) value x and its approximation $\hat{x}$

- *true error:* $E_t = x - \hat{x}$ (*absolute error:* $|x - \hat{x}|$)
- *relative error:*

$$\frac{|\text{true error}|}{|\text{true value}|} = \frac{|x - \hat{x}|}{|x|} \quad \text{(assuming } x \neq 0\text{)}$$

  – gives percentage of error compared to the actual value (ε_t = relative error $\times\ 100\%$)

  – accounts of the order of magnitude of quantities

Example

x	$\hat{x}$	absolute error	relative error
1	0.99	0.01	0.01
1	1.01	0.01	0.01
100	99.99	0.01	0.0001
100	99	1	0.01

- when $|x| \approx 1$, little difference between absolute and relative error
- when $|x| >> 1$, relative error more meaningful

Example: calculation of errors

Problem

(a) measurement of a bridge 9999 cm with true value 10,000 cm

(b) measurement of a rivet 9 cm with true value 10 cm

Error

(a) bridge: $E_t = 10,000 - 9999 = 1$ cm and $\varepsilon_t = \frac{1}{10{,}000} \times 100\% = 0.01\%$

(b) rivet: $E_t = 10 - 9 = 1$ cm and $\varepsilon_t = \frac{1}{10} \times 100\% = 10\%$

- same error (1 cm) but different relative impact
- conclusion: we did a good job of measuring the bridge, but not the rivet

Approximate error

- true value is typically unknown a priori
- use best available estimate

$$\begin{aligned}\varepsilon_a &= \frac{\text{approximate error}}{\text{approximation}} \times 100\% \\ &= \frac{\text{current approximation} - \text{previous approximation}}{\text{current approximation}} \times 100\%\end{aligned}$$

- typically we require $|\varepsilon_a| < \varepsilon_s$ for some small tolerance $\varepsilon_s > 0$
- we can be assured that the result is correct to *at least* n significant figures if

$$\varepsilon_s = (0.5 \times 10^{2-n})\%$$

Example: error estimates for iterative methods

Problem: estimate $e^{0.5} = 1.648721\cdots$ using *Maclaurin series*

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Goal: achieve 3 significant figures ($\varepsilon_s = 0.5 \times 10^{2-3}\% = 0.05\%$)

- add terms until $|\varepsilon_a| < \varepsilon_s$
- example: for $x = 0.5$, first term (1), second term (1.5), etc.

terms	$e^{0.5} \approx$	ε_t (%)	ε_a (%)
1	1	39.3	
2	1.5	9.02	33.3
3	1.625	1.44	7.69
4	1.645833333	0.175	1.27
5	1.648437500	0.0172	0.158
6	1.648697917	0.00142	0.0158

- stops at 6 terms: $\varepsilon_a < 0.05\%$, accurate to 5 significant figures

Example: MATLAB iterative calculation

implement code to iteratively find $e^x \approx \sum_{i=0}^{n} \frac{x^i}{i!}$ for $x = 1$ until $|\varepsilon_a| < 10^{-6}$

```
>> format long
>> [val, ea, iter] = IterMeth(1,1e-6,100)
val =
2.718281826198493
ea =
9.216155641522974e-007
```

```
function [fx,ea,iter] = IterMeth(x,es,maxit)
% Maclaurin series of exponential function
if nargin < 2||isempty(es),es = 0.0001;end
if nargin < 3||isempty(maxit),maxit = 50;end
% initialization
iter = 1; sol = 1; ea = 100;
% iterative calculation
while (1)
solold = sol;
sol = sol + x ^ iter / factorial(iter);
iter = iter + 1;
if sol~ = 0
ea = abs((sol - solold)/sol)*100;
end
if ea< = es || iter> = maxit,break,end
end
fx = sol;
end
```

Trade-offs in numerical methods

Accuracy vs. efficiency

- more accurate methods increase computation time
- *e.g.*, numerical integration with more intervals improves accs. but slows comp.

Stability

- some methods become unstable for certain problems
- *e.g.*, stiff differential equations

Convergence: does the method approach the true solution as iterations increase?

Outline

- significant figures
- numerical errors
- **round-off errors**
- Taylor series, truncation errors
- error propagation

Round-off errors

- computers retain only a fixed number of significant figures
- irrational no. ($\pi, e, \sqrt{7}$) and many base-10 rationals cannot be represented exactly
- base-2 (binary) storage leads to discrepancies when representing base-10 no.
- *round-off error*: discrepancy introduced by omitting significant figures

Example: $\sqrt{2} = 1.4142135623731\cdots \approx 1.4142$ using 5 significant figures

- $(\sqrt{2})^5 = 5.6569$ and $(1.4142)^5 = 5.6566$ (small difference)
- $(\sqrt{2})^{50} = 33554432$ and $(1.4142)^{50} = 33538346.35$ (big difference ≈ 1600)

Number systems

Base-10 (decimal)

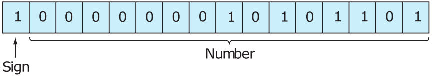
- uses digits 0–9 with place values 10^k
- example: $86,409 = 8 \times 10^4 + 6 \times 10^3 + 4 \times 10^2 + 0 \times 10^1 + 9 \times 10^0$

Base-2 (binary)

- uses digits $0, 1$ with place values 2^k
- example: $(10101101)_2 = 1 \cdot 2^7 + 0 \cdot 2^6 + \cdots + 1 \cdot 2^0 = 173$

Integer representation

- *signed magnitude*: first bit is sign (0: +ve, 1: -ve); remaining bits store magnitude
- example: -173 (16-bit signed magnitude):



Example: determine the base-10 range on a 16-bit machine

- 1 bit for sign; 15 bits for magnitude: max unsigned magnitude

$$(1 \times 2^{14}) + (1 \times 2^{13}) + \cdots + (1 \times 2^{1}) + (1 \times 2^{0}) = 2^{15} - 1 = 32,767$$

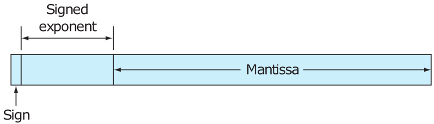
- zero is 0000000000000000, so it is redundant to use 1000000000000000
- store additional -ve number: so range -2^{15} to $2^{15} - 1 = -32{,}768$ to $32{,}767$
- numbers outside this range cannot be represented (**overflow**/**underflow**)

Floating-point representation

fractional numbers are represented in computers using *floating-point* form

$$x = \pm(.d_1 d_1 \cdots d_n) \cdot b^e = \pm\left(\frac{d_1}{b^1} + \cdots + \frac{d_n}{b^n}\right) \cdot b^e = \pm m \cdot b^e$$

- b is the *base* (an integer larger than 1); n is *precision* (number of digits)
- e is *exponent* ($e_{\min} \leq e \leq e_{\max}$)
- $d_1 d_2 d_3 \cdots$ is *mantissa* or *significand*, d_i integer with $0 \leq d_i \leq b - 1$
- stored as:



Examples

- base-10: $0.15678 \times 10^3 = 156.78$
- base-2: $-(.1101) \cdot 2^2 = -(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16}) \cdot 2^2 = -3.25$

Normalization of mantissa

to maximize significant figures, mantissa is **normalized** to remove leading zeros

- example:

$$1/34 = 0.029411765\ldots$$

so

store as 0.0294×10^{0} $\Rightarrow$ normalize to 0.2941×10^{-1}

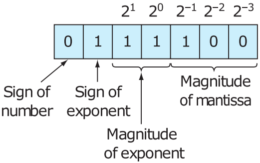
- normalization bounds mantissa:

$$\frac{1}{b} \leq m < 1$$

$0.1 \leq m \leq 1$ for base-10 and for $0.5 \leq m \leq 1$ base-2

Example: binary 7-bit floating-point set

7-bit floating point number stored as



- smallest positive normalized value (shown above):

$$m = 1 \times 2^{-1} = 0.5, \quad e = -(1 \times 2^1 + 1 \times 2^0) = -3 \;\Rightarrow\; +0.5 \times 2^{-3} = 0.0625$$

- next highest numbers are developed by increasing the mantissa, as in

$$0111101 = \left(1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^{-3} = (0.078125)_{10}$$

$$0111110 = \left(1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}\right) \times 2^{-3} = (0.093750)_{10}$$

$$0111111 = \left(1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^{-3} = (0.109375)_{10}$$

in base-10 equivalents are spaced evenly with an interval of 0.015625

Example: binary 7-bit floating-point set

- to continue increasing, we decrease the exponent to 10, which gives a value of

$$e = -(1 \times 2^1 + 0 \times 2^0) = -2$$

- mantissa is decreased back to its smallest 100; so, next number is

$$0110100 = \left(1 \times 2^{-1} + 0 \times 2^{-2} + 0 \times 2^{-3}\right) \times 2^{-2} = (0.125000)_{10}$$

  this still represents a gap of $0.125000 - 0.109375 = 0.015625$

- increasing the mantissa, the gap is lengthened to 0.03125:

$$0110101 = \left(1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^{-2} = (0.156250)_{10}$$

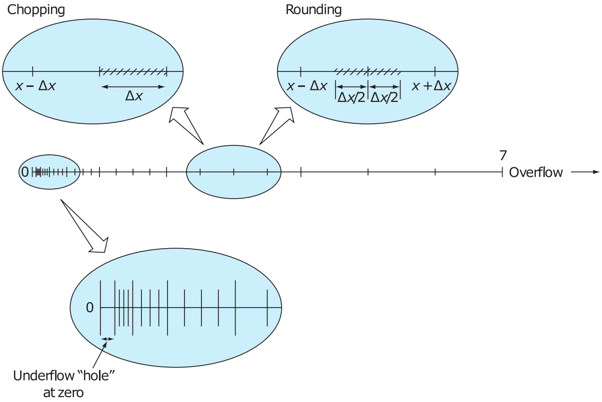
$$0110110 = \left(1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3}\right) \times 2^{-2} = (0.187500)_{10}$$

$$0110111 = \left(1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^{-2} = (0.218750)_{10}$$

- this pattern is repeated until a maximum number is reached:

$$0011111 = \left(1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}\right) \times 2^{3} = (7)_{10}$$

Floating-point consequences and errors



- limited range (overflow and underflow)
- only a finite number of numbers can be represented within the range
- the interval between numbers, Δx, increases as the numbers grow in magnitude

Chopping and rounding

Chopping (truncation)

- discard excess digits (bias toward lower endpoint)
- example (base-10, 7 sig figs): $\pi = 3.1415926535\cdots$

chop: 3.141592

Rounding

- map to nearest representable number (reduced error, unbiased overall)
- example (base-10, 7 sig figs): $\pi = 3.1415926535\cdots$

round: 3.141593

Relative error bounds and machine epsilon

for $\Delta x =$ actual number $-$ floating point representation, we have

- for chopping:

$$\frac{|\Delta x|}{|x|} \leq \mathcal{E}$$

- for rounding:

$$\frac{|\Delta x|}{|x|} \leq \frac{\mathcal{E}}{2}$$

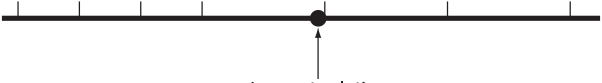
$\mathcal{E}$ is **machine epsilon**

$$\mathcal{E} = b^{1-n}$$

where b is base and n is mantissa digits (precision)

Example: machine epsilon for the 7-bit set

- base $b = 2$, mantissa bits $n = 3$ and chopping, we have $\mathcal{E} = 2^{1-3} = 0.25$
- largest error occurs just below the upper bound of the 1st normalized interval



- for example, maximum error would be a value falling just below the upper bound of the interval between $(0.125000)_{10}$ and $(0.156250)_{10}$
- for this case, the error is less than

$$\frac{|\Delta x|}{|x|} < \frac{0.03125}{0.125000} = 0.25$$

IEEE standard for binary arithmetic

- two binary ($b = 2$) floating-point number systems
- used in almost all modern computers (*e.g.*, MATLAB)

IEEE standard single precision (requires 32 bits)

$$n = 24, \quad e_{\min} = -125, \quad e_{\max} = 128$$

- 23 bits for mantissa ($d_1 = 1$ not stored)
- 1 sign bit and 8 bits for exponent
- about 7 significant base-10 digits precision with range 10^{-38} to 10^{39}

IEEE standard double precision (requires 64 bits)

$$n = 53, \quad e_{\min} = -1021, \quad e_{\max} = 1024$$

- 52 bits for mantissa ($d_1 = 1$ not stored)
- 1 sign bit and 11 bits for exponent
- about 16 significant base-10 digits precision with range 10^{-308} to 10^{308}

Arithmetic manipulations

arithmetic with floating-point numbers introduces additional round-off error

- for simplicity: use base-10 numbers with 4-digit mantissa, 1-digit exponent, and chopping
- other number bases and rounding would behave in a similar fashion
- focus on: addition, subtraction, multiplication, division
- long sequences of interdependent operations can accumulate small round-off errors

Addition

mantissa of no. with smaller exponent is modified so that exponents are the same

Example

$$0.1557 \cdot 10^1 \ + \ 0.4381 \cdot 10^{-1}$$

- write

$$0.4381 \cdot 10^{-1} \ \rightarrow \ 0.004381 \cdot 10^1$$

- then add

$$\begin{array}{r} 0.1557 \cdot 10^1 \\ 0.004381 \cdot 10^1 \\ \hline 0.160081 \cdot 10^1 \end{array} \quad \Rightarrow \quad \text{chop} \ \rightarrow \ 0.1600 \cdot 10^1$$

Adding a large and a small number: add 4000 to 0.0010:

$$0.4000 \cdot 10^4 \ + \ 0.0000001 \cdot 10^4 \ = \ 0.4000001 \cdot 10^4$$

$$\text{chop} \ \rightarrow \ 0.4000 \cdot 10^4$$

Example: MATLAB

sum 0.0001 to itself 10^4 times, which should gives 1

in MATLAB running

```
s = 0;
for i = 1:10000
s = s + 0.0001;
end
sout = s
```

gives

```
sout=0.99999999999991
```

while 0.0001 is a nice number in base-10, it cannot be expressed exactly in base-2

Subtraction

Subtraction

$$0.3641 \cdot 10^2 \; - \; 0.2686 \cdot 10^2 \; = \; 0.0955 \cdot 10^2$$

$$0.0955 \cdot 10^2 \; \rightarrow \; 0.9550 \cdot 10^1 \; = \; 9.550$$

zero added to the end is not significant but is appended to fill the empty space

Subtracting two nearly equal numbers

$$0.7642 \cdot 10^3 \; - \; 0.7641 \cdot 10^3 \; = \; 0.0001 \cdot 10^3$$

$$0.0001 \cdot 10^3 \; \rightarrow \; 0.1000 \cdot 10^0 \; = \; 0.1000$$

three nonsignificant zeros are appended

round-off induced when subtracting two nearly equal floating-point numbers is called *subtractive cancellation*

Multiplication and division

Multiplication: multiply mantissas, add exponents, then normalize and chop

$$(0.1363 \cdot 10^3) \times (0.6423 \cdot 10^{-1}) \; = \; 0.08754549 \cdot 10^2$$

$$0.08754549 \cdot 10^2 \; \rightarrow \; 0.8754549 \cdot 10^1 \; \xrightarrow{\text{chop}} \; 0.8754 \cdot 10^1$$

Division: divide mantissas, subtract exponents, then normalize and chop

Outline

- significant figures
- numerical errors
- round-off errors
- **Taylor series, truncation errors**
- error propagation

Taylor series

if f and its first $n+1$ derivatives are continuous on an interval containing a and x, then

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n$$

with remainder (integral form)

$$R_n = \int_a^x \frac{(x-t)^n}{n!} f^{(n+1)}(t)\, dt$$

- called *Taylor's theorem* or *Taylor series*
- called *Taylor approximation* if R_n is omitted
- provides a polynomial approximation of smooth functions
- predict f at a new point using values and derivatives at a nearby point
- by the integral mean-value theorem, there exists ξ between a and x such that

  $$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}$$

  this is the *derivative* or *Lagrange form* of the remainder

Taylor approximations

Zero-order (constant)

$$f(x_{i+1}) \approx f(x_i)$$

First-order (affine approximation)

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)\,(x_{i+1} - x_i)$$

Second-order (quadratic approximation)

$$f(x_{i+1}) \approx f(x_i) + f'(x_i)\,(x_{i+1} - x_i) + \frac{f''(x_i)}{2}\,(x_{i+1} - x_i)^2$$

- approximation improves if x_{i+1} is near x_i
- higher-order terms capture curvature and improve accuracy

Full series about x_i and remainder

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \frac{f''(x_i)}{2!}(x_{i+1} - x_i)^2 + \frac{f^{(3)}(x_i)}{3!}(x_{i+1} - x_i)^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}(x_{i+1} - x_i)^n + R_n$$

where

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,(x_{i+1} - x_i)^{\,n+1}$$

Step size form ($h = x_{i+1} - x_i$)

$$f(x_{i+1}) = f(x_i) + f'(x_i)\,h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots + \frac{f^{(n)}(x_i)}{n!}h^n + R_n$$

where (for $x_i \le \xi \le x_{i+1}$)

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,h^{n+1}$$

Example: polynomial approximation

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2,$$

approximate $f(1) = 0.2$ from $x_i = 0$ with $h = 1$ using $n = 0, 1, 2, 3, 4$

Zero-order: $f(x_{i+1}) \approx f(0) = 1.2$ with $E_t = 0.2 - 1.2 = -1$

First-order: $f'(x) = -0.4x^3 - 0.45x^2 - x - 0.25 \implies f'(0) = -0.25$

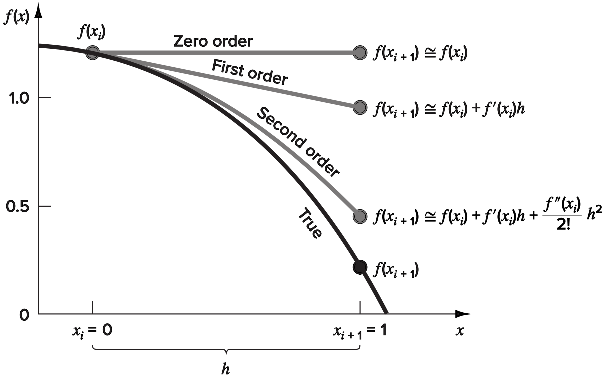
$$f(x_{i+1}) \approx 1.2 - 0.25\,h \implies f(1) = 0.95$$

$E_t = 0.2 - 0.95 = -0.75$

Second-order: $f''(x) = -1.2x^2 - 0.9x - 1 \implies f''(0) = -1$

$$f(x_{i+1}) \approx 1.2 - 0.25\,h - \frac{1}{2}h^2 \implies f(1) = 0.45$$

$E_t = 0.2 - 0.45 = -0.25$



Fourth-order

$$f^{(3)}(x) = -2.4x - 0.9, \quad f^{(3)}(0) = -0.9, \qquad f^{(4)}(x) = -2.4, \quad f^{(4)}(0) = -2.4$$

$$\begin{aligned} f(1) &\approx 1.2 - 0.25(1) - \frac{1}{2}(1)^2 - \frac{0.9}{6}(1)^3 - \frac{2.4}{24}(1)^4 \\ &= 1.2 - 0.25 - 0.5 - 0.15 - 0.1 = 0.2 \end{aligned}$$

since f is a 4th-degree polynomial, the $n = 4$ Taylor polynomial is exact and $R_4 = 0$

Truncation error order notation

with $h = x_{i+1} - x_i$,

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}\, h^{n+1} = \mathcal{O}(h^{n+1})$$

- if error is $\mathcal{O}(h)$, halving h halves the error
- if $\mathcal{O}(h^2)$, halving h quarters the error
- for sufficiently small h, only a few terms are required to get a good estimate

Example

- we use Taylor series to approximate $f(x) = \cos x$ at $x_{i+1} = \pi/3$
- base point $x_i = \pi/4$, step $h = \pi/12$, true value $f(\pi/3) = 0.5$

order n	$f^{(n)}(x)$	$f(\pi/3) \approx$	ε_t
0	$\cos x$	0.707106781	-41.4
1	$-\sin x$	0.521986659	-4.4
2	$-\cos x$	0.497754491	0.449
3	$\sin x$	0.499869147	2.62×10^{-2}
4	$\cos x$	0.500007551	-1.51×10^{-3}
5	$-\sin x$	0.500000304	-6.08×10^{-5}
6	$-\cos x$	0.499999988	2.44×10^{-6}

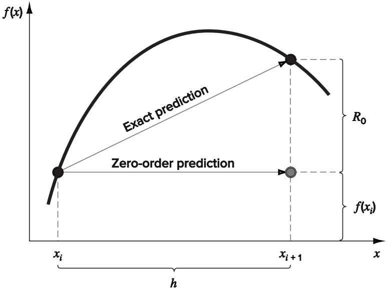
Remainder and the mean-value insight

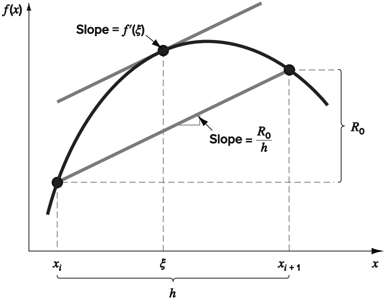
- zero-order truncation:

$$f(x_{i+1}) = f(x_i) + R_0, \qquad R_0 = f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \frac{f^{(3)}(x_i)}{3!}h^3 + \cdots$$

- derivative mean-value theorem $\Rightarrow$ there exists $\xi \in (x_i, x_{i+1})$ with

$$R_0 = f'(\xi)\, h$$





Numerical differentiation: forward difference approximation

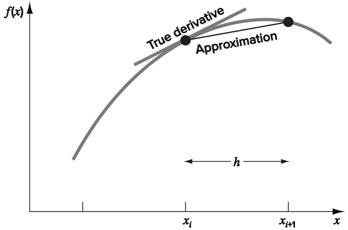
derivative forward approximation

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \quad \text{error } \mathcal{O}(x_{i+1} - x_i) = \mathcal{O}(h)$$

- follows from Taylor series

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + R_1, \quad R_1 = \frac{f''(\xi)}{2!}(x_{i+1} - x_i)^2$$

- called *finite divided difference*, $f(x_{i+1}) - f(x_i)$ is called *first forward difference*
- $\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}$ called *first finite divided difference*



(a)

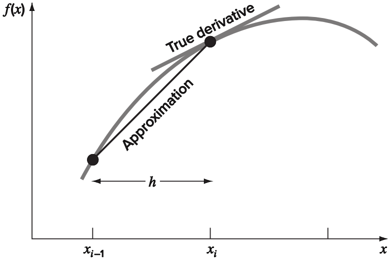
Numerical differentiation: backward difference approximation

derivative backward approximation

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{h}, \quad \text{error } \mathcal{O}(h)$$

follows from Taylor series expanded backward

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \cdots$$



(b)

Numerical differentiation: centered difference approximation

derivative centered difference approximation

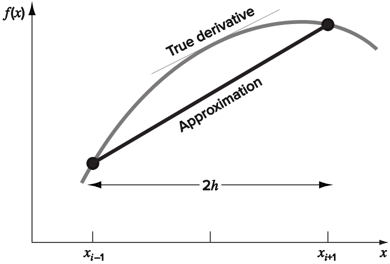
$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h}, \quad \text{error } \mathcal{O}(h^2)$$

follows by subtracting

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \cdots$$

from

$$f(x_{i-1}) = f(x_i) - f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \cdots$$



(c)

Example

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.25$$

approximate $f'(0.5)$ using $h = 0.5$ and $h = 0.25$ (true value $f'(0.5) = -0.9125$)

for $h = 0.5$, using

$$x_{i-1} = 0, \quad f(x_{i-1}) = 1.2$$

$$x_i = 0.5, \quad f(x_i) = 0.925$$

$$x_{i+1} = 1.0, \quad f(x_{i+1}) = 0.2$$

we get

$$\text{forward: } \frac{0.2 - 0.925}{0.5} = -1.45 \quad (|\varepsilon_t| = 58.9\%)$$

$$\text{backward: } \frac{0.925 - 1.2}{0.5} = -0.55 \quad (|\varepsilon_t| = 39.7\%)$$

$$\text{centered: } \frac{0.2 - 1.2}{1.0} = -1.0 \quad (|\varepsilon_t| = 9.6\%)$$

for $h = 0.25$, using

$$x_{i-1} = 0.25, \quad f(x_{i-1}) = 1.10351563$$

$$x_i = 0.5, \quad f(x_i) = 0.925$$

$$x_{i+1} = 0.75, \quad f(x_{i+1}) = 0.63632813$$

we get

$$\text{forward: } \frac{0.6363 - 0.925}{0.25} = -1.155 \quad (|\varepsilon_t| = 26.5\%)$$

$$\text{backward: } \frac{0.925 - 1.1035}{0.25} = -0.714 \quad (|\varepsilon_t| = 21.7\%)$$

$$\text{centered: } \frac{0.6363 - 1.1035}{0.5} = -0.934 \quad (|\varepsilon_t| = 2.4\%)$$

Numerical differentiation: second-order derivative

Second forward finite divided difference

$$f''(x_i) \approx \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2}, \quad \text{error } \mathcal{O}(h)$$

follows by subtracting

$$f(x_{i+2}) = f(x_i) + f'(x_i)(2h) + \frac{f''(x_i)}{2!}(2h)^2 + \cdots$$

from 2 times

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 - \cdots$$

Second backward finite divided difference

$$f''(x_i) \approx \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{h^2}, \quad \text{error } \mathcal{O}(h)$$

Second centered finite divided difference

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}, \quad \text{error } \mathcal{O}(h^2)$$

Outline

- significant figures
- numerical errors
- round-off errors
- Taylor series, truncation errors
- **error propagation**

Function error propagation

let $\tilde{x}$ be an approximation of x; we seek

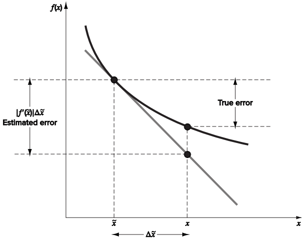
$$\Delta f(\tilde{x}) \;=\; |f(x) - f(\tilde{x})|$$

- assuming $\tilde{x}$ is close to x and f is continuous and differentiable near $\tilde{x}$,

$$f(x) \;=\; f(\tilde{x}) + f'(\tilde{x})(x - \tilde{x}) + \frac{f''(\tilde{x})}{2}\,(x - \tilde{x})^2 + \cdots$$

- dropping second and higher terms and letting $\Delta(\tilde{x}) = |x - \tilde{x}|$:

$$\Delta f(\tilde{x}) = \approx \;|f'(\tilde{x})\,(x - \tilde{x})| = |f'(\tilde{x})|\,\Delta(\tilde{x})$$



Example

- given $\tilde{x} = 2.5$ with error $\Delta(\tilde{x}) = 0.01$, estimate the error in $f(x) = x^3$
- we have

$$\Delta f(\tilde{x}) \;\approx\; |f'(\tilde{x})|\,\Delta(\tilde{x}) \;=\; |3\tilde{x}^2| \cdot 0.01 \;=\; 3(2.5)^2(0.01) \;=\; 0.1875$$

- since $f(2.5) = 15.625$,

$$f(2.5) \approx 15.625 \pm 0.1875,$$

so the true value lies in $[15.4375,\ 15.8125]$

Stability and condition

Condition (of a problem)

- sensitivity of $f(x)$ to small changes in the input x
- algorithm is *unstable* if it magnifies input/round-off uncertainties
- relative errors:

$$\frac{f(x) - f(\tilde{x})}{f(\tilde{x})} \approx \frac{f'(\tilde{x})\,(x - \tilde{x})}{f(\tilde{x})}, \qquad \frac{x - \tilde{x}}{\tilde{x}} \text{ is the relative error in } x$$

Condition number: ratio of relative errors

$$\boxed{\kappa(\tilde{x}) = \frac{\tilde{x}\, f'(\tilde{x})}{f(\tilde{x})}}$$

- $\kappa \approx 1$: output relative error $\approx$ input relative error
- $|\kappa| > 1$: relative error is **amplified** (ill-conditioning as $|\kappa|$ grows)
- $|\kappa| < 1$: relative error is **attenuated**

Example

we compute condition number for $f(x) = \tan x$

$$\kappa(\tilde{x}) \;=\; \frac{\tilde{x}\, f'(\tilde{x})}{f(\tilde{x})} \;=\; \frac{\tilde{x}\ \sec^2 \tilde{x}}{\tan \tilde{x}}$$

Case 1: $\tilde{x} = \dfrac{\pi}{2} + 0.1\left(\dfrac{\pi}{2}\right)$

$$\kappa(\tilde{x}) \approx \frac{1.7279 \times 40.86}{-6.314} \approx -11.2$$

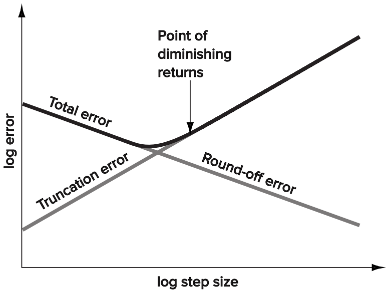
Case 2: $\tilde{x} = \dfrac{\pi}{2} + 0.01\left(\dfrac{\pi}{2}\right)$

$$\kappa(\tilde{x}) \approx \frac{1.5865 \times 4053}{-63.66} \approx -101$$

near $\pi/2$, $\sec^2 \tilde{x}$ grows and $\tan \tilde{x}$ changes rapidly $\Rightarrow$ severe ill-conditioning

Total numerical error: truncation vs. round-off

- total error = truncation + round-off
- $\downarrow h$, $\downarrow$ truncation, but $\uparrow$ round-off due to more ops/subtractive cancellation
- goal: choose step size to balance truncation and round-off contributions



Error analysis of numerical differentiation

- centered difference with truncation term:

$$f'(x_i) \;=\; \frac{f(x_{i+1}) - f(x_{i-1})}{2h} \;-\; \frac{f^{(3)}(\xi)}{6}\, h^2$$

  if the numerator values are exact, the error is due only to truncation

- with rounded values and round-off errors

$$f(x_{i-1}) = \tilde{f}(x_{i-1}) + e_{i-1}, \qquad f(x_{i+1}) = \tilde{f}(x_{i+1}) + e_{i+1}$$

- so

$$f'(x_i) = \frac{\tilde{f}(x_{i+1}) - \tilde{f}(x_{i-1})}{2h} \;+\; \frac{e_{i+1} - e_{i-1}}{2h} \;-\; \frac{f^{(3)}(\xi)}{6}\, h^2$$

- assume $|e_{i\pm1}| \leq \varepsilon$ and $|f^{(3)}(x)| \leq M$, then

$$\left| f'(x_i) - \frac{\tilde{f}(x_{i+1}) - \tilde{f}(x_{i-1})}{2h} \right| \;\leq\; \frac{\varepsilon}{h} + \frac{M}{6} h^2$$

  minimizing the bound w.r.t. h yields the optimal step size $h_{\text{opt}} \;=\; (\frac{3\,\varepsilon}{M})^{1/3}$

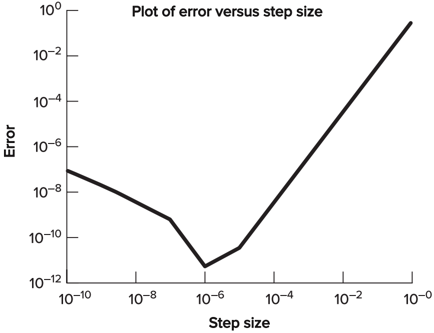
Example

- consider $f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2$
- at $x = 0.5$: $f'(0.5) = -0.9125$.
- use centered difference (order $O(h^2)$) with step sizes $h = 10^0, 10^{-1}, \ldots, 10^{-10}$

```
function diffex(func, dfunc, x, n)
format long
dftrue = dfunc(x);
h = 1; H(1) = h;
D(1) = (func(x+h) - func(x-h)) / (2*h);
E(1) = abs(dftrue - D(1));
for i = 2:n
h = h / 10; H(i) = h;
D(i) = (func(x+h) - func(x-h)) / (2*h);
E(i) = abs(dftrue - D(i));
end
L = [H' D' E']';
fprintf(' step size   finite difference      true error\n');
fprintf('%14.10f %16.14f %16.13f\n', L);
loglog(H, E), xlabel('Step Size'), ylabel('Error')
title('Plot of Error Versus Step Size')
 format short
```

Example

```
>> ff = @(x) -0.1*x.^4 - 0.15*x.^3 - 0.5*x.^2 - 0.25*x + 1.2;
>> df = @(x) -0.4*x.^3 - 0.45*x.^2 - x - 0.25;
>> diffex(ff, df, 0.5, 11)
```



- third derivative bound at $x = 0.5$: $M = |f^{(3)}(0.5)| = |-2.4(0.5) - 0.9| = 2.1$
- with machine round-off bound $\varepsilon \approx 0.5 \times 10^{-16}$,

$$h_{\text{opt}} = \left(\frac{3\varepsilon}{M}\right)^{1/3} = \left(\frac{3 \times 0.5 \times 10^{-16}}{2.1}\right)^{1/3} \approx 4.3 \times 10^{-6}$$

References and further readings

- S. C. Chapra and R. P. Canale. *Numerical Methods for Engineers* (8th edition). McGraw Hill, 2021. (Ch.3, Ch.4)

- S. C. Chapra. *Applied Numerical Methods with MATLAB for Engineers and Scientists* (5th edition). McGraw Hill, 2023. (Ch.4)