Process Model Formulation and Solution, 3E4

Section A: Scientific computing: Approximations and round-off error

Instructor: Kevin Dunn dunnkg@mcmaster.ca

Department of Chemical Engineering

22 September 2010

Where are we? Where do we want to go?

What have we learned so far?

- How to formulate mathematical models for lumped systems

What do we want to do next?

- Apply numerical methods to solve the models

But, before that:

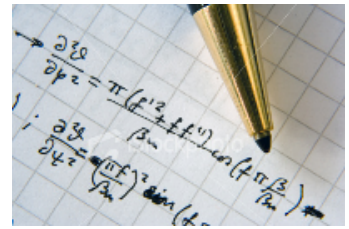
- Need to understand the concept and potential sources of errors
  - Effective use of numerical methods
  - Errors can be costly and sometimes catastrophic in professional practice

> *"At the source of every error which is blamed on the computer you will find at least two human errors, including the error of blaming it on the computer"*

Various sources of errors

Model formulation errors:

- Neglected controlling mechanisms
- Inaccurate physical data or model parameters
- Random disturbances



Numerical errors:

- **Truncation/discretization errors**, due to approximations in the numerical method; e.g., $\frac{\mathrm{d}f}{\mathrm{d}x}(x_0) \approx \frac{f(x_0+\delta x)-f(x_0)}{\delta x}$
- **Round-off errors**, due to representation of quantities with a finite number of digits only; e.g., $\pi, \sqrt{2}, e$

Crucial questions:

- How much error is present in my model?
- How much error is incurred by my numerical solution procedure?
- Is this acceptable in terms of my initial goals?

Outline

Significant digits

Concept of significant digits: *Those that can be used with confidence*

- **Speed estimation:**
- **Mileage estimation:**

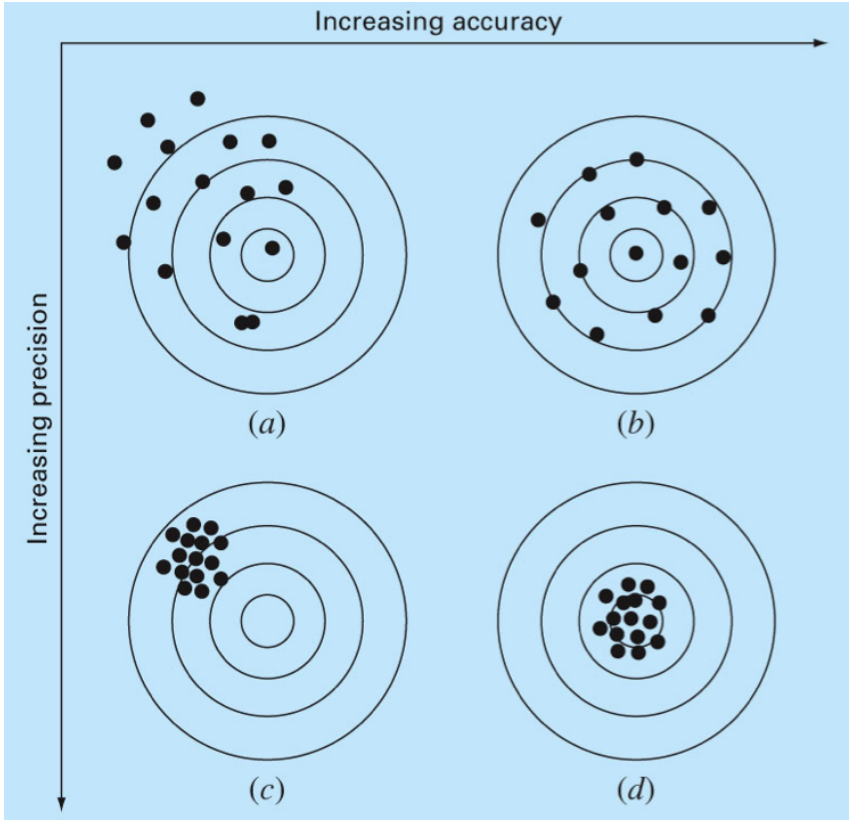
Unambiguous representation:

- Specify significant digits using scientific notation
- E.g., $1.23 \times 10^4 \rightarrow$ 3 signif. digits;
  $1.2300 \times 10^4 \rightarrow$ 5 significant digits

Error characterization

Two useful definitions:

- Bias or inaccuracy: systematic deviation from the exact value
- Variance or imprecision: related to the magnitude of the scatter



A good numerical method should be both accurate and precise!

Error characterization

When the true value is available:

- True absolute error: $\varepsilon_t^{\text{abs}} = \text{true value} - \text{approximation}$
- True relative error: $\varepsilon_t^{\text{rel}} = \dfrac{\text{true value} - \text{approximation}}{\text{true value}} \times 100\%$

When the true value is not available: ← typical in practice!

- Approximate relative error: $\varepsilon_a^{\text{rel}} = \dfrac{\text{approximate error}}{\text{approximate value}} \times 100\%$

Error characterization in iterative methods:

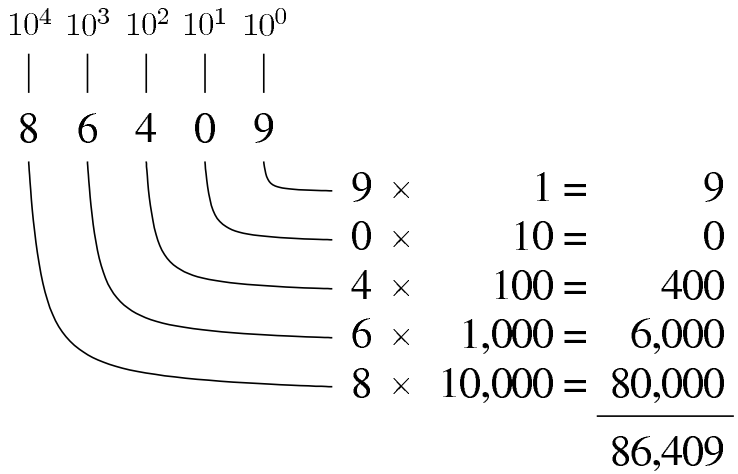
$$\varepsilon_a^{\text{rel}} = \frac{k+1^{\text{st}} \text{ approximate error} - k^{\text{th}} \text{ approximate error}}{k+1^{\text{st}} \text{ approximate error}} \times 100\%$$

Useful to define a stopping criterion: $|\varepsilon_a^{\text{rel}}| <$ user-defined tolerance

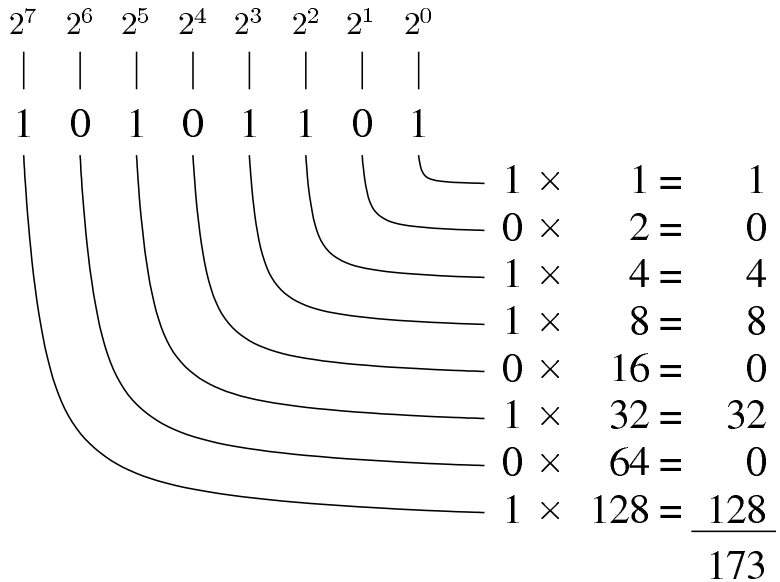
Number systems

- Round-off errors related to the way numbers are stored in a computer
- Numbers are stored in *words*, i.e. strings of binary digits – *bits* in short

Decimal or base-10 system:



Binary or base-2 system:



- Other number systems: octal (base-8), hexadecimal (base-16)

Workshop

Represent the decimal integer 148 in the binary number system

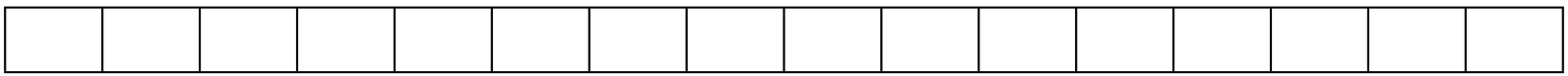
Computer representation of integers

Signed magnitude approach:

- Most straightforward approach
- 1st bit of each word indicates the sign: 0 $\rightarrow$ positive; 1 $\rightarrow$ negative

Workshop

Represent the decimal integer -148 on a 16-bit machine with the signed magnitude approach



- Two possible representations for the integer 0
- Range of integers:
  - 16-bit machine: $(-32767)_{10} \rightarrow (32767)_{10}$
  - 32-bit machine: DIY: in MATLAB [*maxint*] or Python [*np.iinfo*]

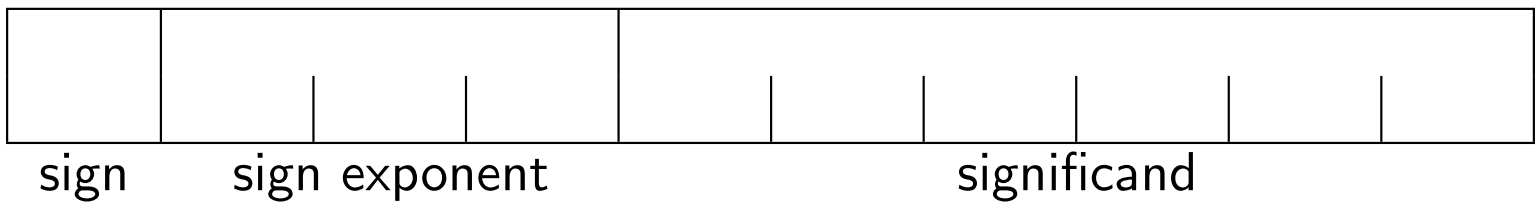
Unsigned (positive) integers:

- Unsigned integers: uint8, uint16, uint32

Computer representation of floating-point numbers

Floating-point form: $m \times b^e$

- m, mantissa (aka significand: better term)
- b, base
- e, exponent



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

- Removes "useless" zeros; e.g., 0.02941 normalized as 0.2941×10^{-1}

DIY example: Chapra and Canale, Ch.3

Consider the representation of a floating-point number on words of 8 bits, with (i) sign coded on the bit 1, (ii) signed exponent coded on bits 2-4, and (iii) mantissa magnitude coded on bits 5-8.

- What are the smallest, 2nd smallest, and 3rd smallest positive floating-point numbers?
- What is the largest positive floating-point number?

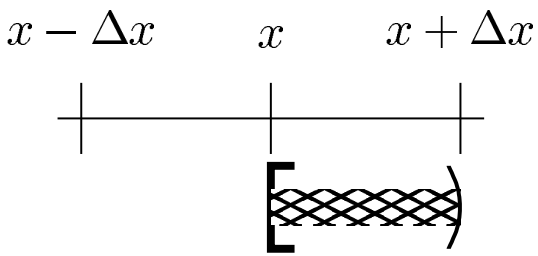
Computer representation of floating-point numbers

Facts:

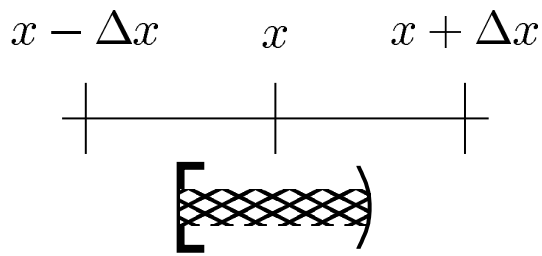
1. Only a limited range of quantities can be represented:
   - overflow: attempt to access numbers outside the acceptable range
   - underflow "hole": between zero and the first positive number

2. Only a finite number of quantities can be represented
   - Retain only significant figures

Chopping: store as the quantity at the lower bound of the interval



Rounding: store as the nearest allowable number



- This is why you should never test equality of 2 floating-point numbers!

Computer representation of floating-point numbers

Facts:

3. Intervals between number increase as the numbers grow in magnitude
   - Maximal interval-to-value ratio: $\frac{|\Delta x|}{|x|} \leq \epsilon$
   - Machine precision: $\epsilon = b^{1-t}$ (t, significant digits in significand)

Workshop

What is the machine precision for 8-bit words with significand magnitude coded on 4 bits?

Standard IEEE formats and precision:

- Single precision:
  - range: $10^{-38} - 10^{38}$, $\epsilon = \approx 10^{-7}$
- Double precision:
  - range: $10^{-308} - 10^{308}$, $\epsilon = \approx 10^{-16}$
- Quadruple precision:
  - range: $10^{-4932} - 10^{4932}$, $\epsilon = \approx 10^{-19}$

```
epsilon = 1
DO
  IF (epsilon+1 <= 1)
    EXIT
  epsilon = epsilon/2
ENDDO
epsilon = 2*epsilon
```

Computer representation of special numbers

1. `inf` and `-inf`:
   - represents $+\infty$ and $-\infty$
   - arise with division by zero: `4.2 / 0.0`
   - MATLAB: `inf` and `-inf`
   - Python: `np.inf` and `-np.inf`
   - uses special reserved values of exponent to represent it
2. `NaN`:
   - "not-a-number"
   - often to represent missing values
   - intended to signal undefined operations
   - e.g. `0.0 / 0.0` or `0.0 * Inf`
   - Python only: square root of a negative number
   - Interesting result: `nan == nan` is false: why?
3. +0 and −0 (not available in MATLAB)
   - both -0 and $+0$ exist
   - MATLAB hides this from the user
   - Python: `0.0 / -2.0` has a result of `-0.0`

Computer demo

`NaN`, `Inf`, `-0.0`, a demo of overflow,
and machine precision: `(1.0 + (e + e)) != (1.0 + e + e)`

Round-off errors and common arithmetic operations

Addition:

- Loss of precision occurs when adding number of different magnitudes
- Cumulative effect can be very large

Workshop

Consider a hypothetical computer with a 4-digit significand and the 1-digit exponent (in the decimal system). What is the result of adding 0.1557×10^1 and 0.4381×10^{-1}? Which information is lost?

Subtraction:

- Loss of precision occurs when subtracting nearly equal numbers; a.k.a. subtractive cancellation
- Among the greatest source of round-off error in numerical methods

Workshop

What is the result of subtracting 0.7641×10^3 from 0.7642×10^3 on the same hypothetical computer? How many significant digits are lost?

Final words

The three main sources of errors are due to:

1. **The modeller:** model formulation errors
2. **The algorithm:** truncation/discretization errors
3. **The computer:** round-off errors

In engineering applications, use double-precision or better to mitigate the effect of round-off error

Reading:

- Chapter 3 in S. C. Chapra, and R. P. Canale, *"Numerical Methods for Engineers"*, McGraw Hill, 5th/6th Edition
- Wikipedia article on `Floating point` is well-written
- What every computer scientist should know about floating-point arithmetic