Approximations and round-off errors

Introduction to Numerical Problem Solving, Spring 2017

Sakari Lukkarinen

Helsinki Metropolia University of Applied Sciences

Content

- Introduction
- Significant figures
- Accuracy, precision and error
- Error definitions
- Tolerance and stop criteria
- Error estimates for iterative methods
- Types of mathematical problem
- Trade-offs and characteristics of numerical methods
- Summary

Introduction

Numerical methods involves always approximations and thus leads to errors:

- *Round-off error* is due to fact that computers represents only quantities with a finite number of digits.
- *Truncation error* comes from the approximations the numerical methods employ.

In [230]:
```
from decimal import *
getcontext().prec = 4
a = Decimal(1.234567)
print(2*a)
```

```
2.469
```

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

In [232]:
```
x = 0.5
y = 1 + x + x**2/2 + x**3/(1*2*3)
print(exp(0.5), y)
```

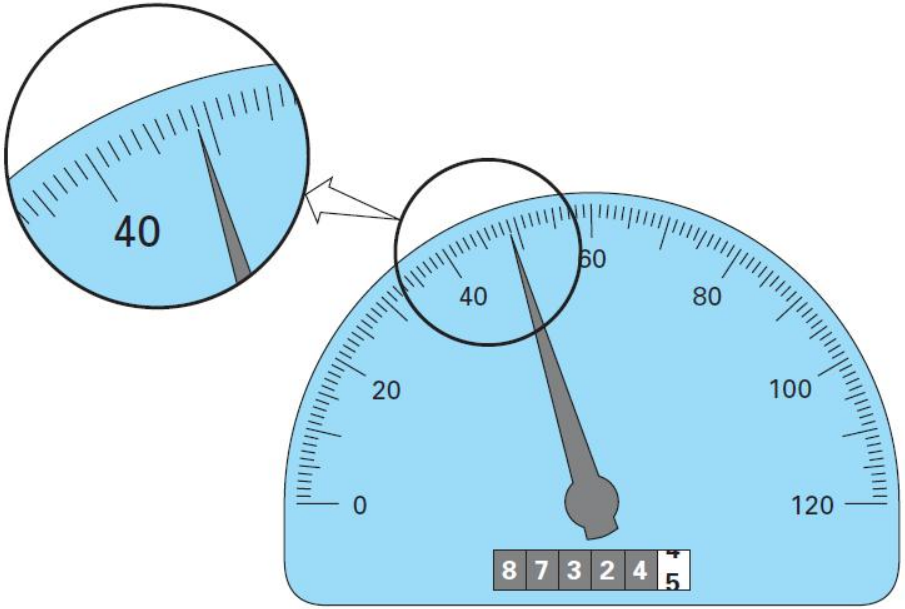
```
1.6487212707 1.6458333333333333
```

Significant figures

Can we say that the speed is 48.8642138 km/h?

Would it be better to say it is between 48 and 49 km/h?

Or roughly 50 km/h?



Trailing zeros

The numbers

0.00001845, 0.0001845, and 0.001845

All have four significant figures.

The number

45 300

May have three, four or five significant figures. This uncertainty can be resolved with scientific notation

4.53×10^4, 4.530×10^4, 4.5300×10^4

Implications

- Numerical methods yields approximate results. è We must develop *criteria* to specify how confident we are in our approximate results.

- Because computers retain only a *finite number of significant figures*, we can never represent some numbers, like $\pi, e, \sqrt{2}$, exactly. This leads to *round-off error*.

Accuracy, Precision and Error

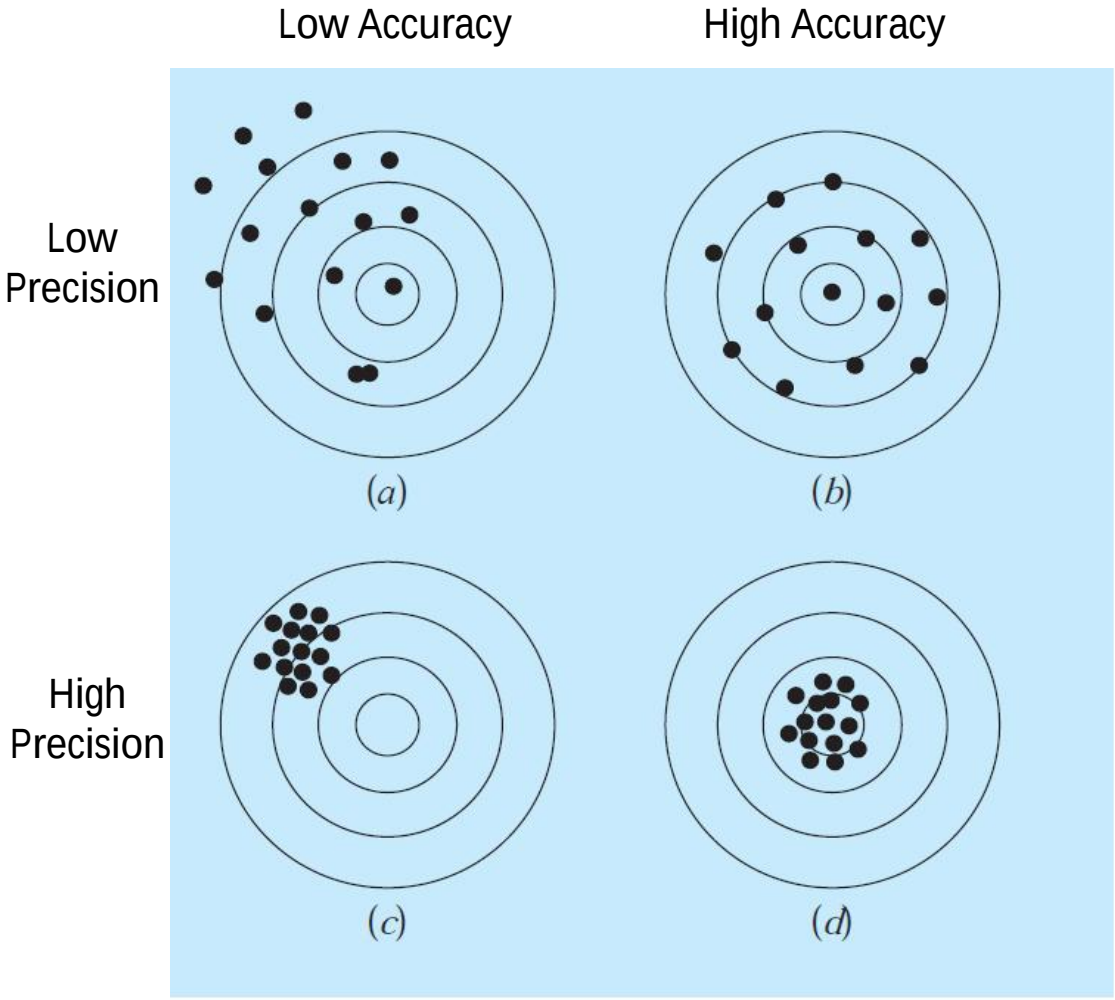
Accuracy refers to how closely a computed or measured value agrees with the true value.

Precision refers to how closely individual computed or measured values agree with each other.

Inaccuracy (or *bias*) is defined as systematic deviation from the truth.

Imprecision (or *uncertainty*) refers to the magnitude of the scatter.

Collective term *Error* represents both inaccuracy and uncertainty.



Error definitions

$$true\_value = approximation + true\_error$$

Numerical true error is then

$$E_t = true\_value - approximation$$

True percent relative error (or normalized error)

$$\epsilon_t = \frac{true\_error}{true\_value} \cdot 100\%$$

Example

You have task of measuring the lengths of a bridge and a rivet. You measured 9999 and 9 cm. If true values are 10,000 and 10 cm, compute the true error and the true percent relative error.

True error

For bridge $E_t = 10{,}000 - 9999 = 1\ cm$

For rivet $E_t = 10 - 9 = 1\ cm$

Relative error

For bridge $\epsilon_t = \frac{1}{10{,}000} \cdot 100\% = 0.01\%$

For rivet $\epsilon_t = \frac{1}{10} \cdot 100\% = 10\%$

Rivets in bridge

Approximation error

If we don’t know the true value, then

$$\epsilon_a = \frac{approximate\_error}{approximation} \cdot 100\%$$

In iterative approaches

$$\epsilon_a = \frac{current\_approximation - previous\_approximation}{current\_approximation} \cdot 100\%$$

Tolerance and stop criteria

Repeat computation until

$$|\epsilon_a| < \epsilon_s$$

ϵ_s is a prespecified *tolerance* (in percent)

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

we can be sure the result is correct *at least n* significant figures.

Error estimates for iterative methods

How many terms of the series do we need in order that $e^{0.5}$ = 1.648721 ... is correct to at least three significant figures?

Stop criteria: n = 3, $\epsilon_s = (0.5 \times 10^{2-3})\% = 0.05\%$

Functions can be presented by infinite series. For example, exponential function can be computed using

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

Example – Iterate the value of exp(x)

First term approximation

$$e^x \approx 1$$

Second term approximation

$$e^x \approx 1 + x$$

For $x = 0.5$,

$$e^{0.5} \approx 1 + 0.5 = 1.5$$

A true percent relative error

$$\epsilon_t = \frac{1.648721 - 1.5}{1.648725} \cdot 100\% = 9.02\%$$

Approximate estimate error

$$\epsilon_a = \frac{1.5 - 1}{1.5} \cdot 100\% = 33.3\%$$

Continue until $\epsilon_a < \epsilon_s$

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

Terms	Result	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

Python code for an iterative solution of exp(x)

```
from scipy.misc import factorial
from numpy import abs
def iterExp(val, es, maxiter):
    iter = 1
    sol = val
    ea = 100
    while True:
        sol_old = sol
        sol = sol + (val**iter)/factorial(iter)
        iter = iter + 1
        if sol != 0:
            ea = abs((sol - sol_old)/sol)*100
        if ea <= es or iter >= maxiter:
            break
    return sol, ea, iter
```

```
val, ea, iter = iterExp(1, 1e-6, 30)
print(val, ea, iter)

>> 2.7182818262 9.21615564152e-07 12

trueval = exp(1)
print(trueval)

>> 2.71828182846

et = (trueval - val)/trueval*100
print(et)

>> 8.31610676352e-08
```

Round-off error

Computers keep only a certain number of significant figures in memory. In addition, computers use a base-2 representation, they cannot precisely represent certain 10-base numbers. Round-off error comes from omission of significant figures.

sqrt(2) ==> 1.4142135623730951

27/17 ==> 1.588235294117647

How many digits (significant figures) do these examples have?

Computer representation of numbers

The fundamental information representation unit is *word*, that consists of a string of *bi*nary digi*ts* or *bits.*

2-base integer number

$$00001111_2 = 0 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$= 8 + 4 + 2 + 1 = 15$$

10-base integer number

$$0015_{10} = 0 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 = 10 + 5 = 15$$

16-base integer number

$$000F_{16} = 0 \cdot 16^3 + 0 \cdot 10^2 + 0 \cdot 16^1 + 15 \cdot 10^0 = 15$$

Signed 16-bit integer

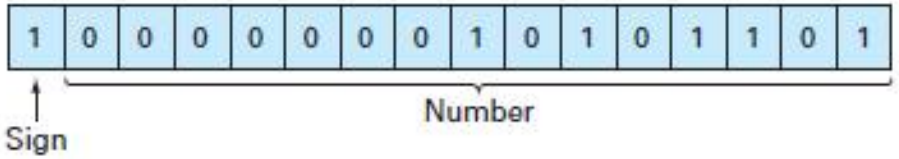
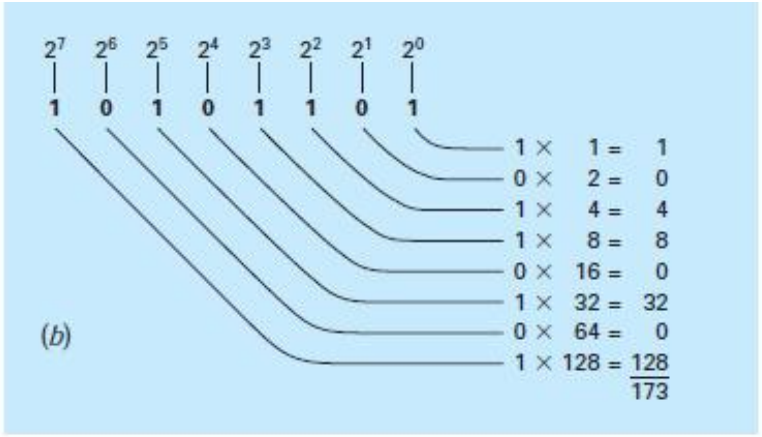


FIGURE 3.6
The representation of the decimal integer −173 on a 16-bit computer using the signed magnitude method.



Floating point representation

$$x = sign \cdot m \cdot 2^{exponent}$$

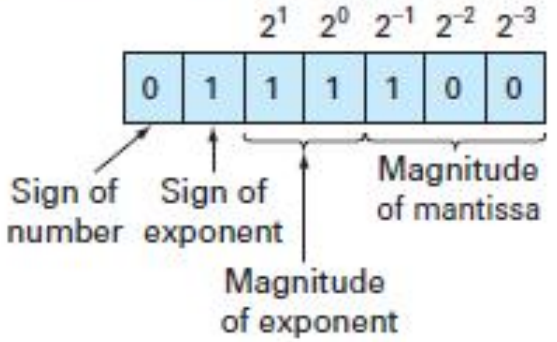
FIGURE 3.7
The manner in which a floating-point number is stored in a word.


```
print(np.finfo(np.float64))

Machine parameters for float64
---------------------------------------------------------------
precision= 15   resolution= 1.0000000000000001e-15
machep=   -52   eps=        2.2204460492503131e-16
negep =   -53   epsneg=     1.1102230246251565e-16
minexp= -1022   tiny=       2.2250738585072014e-308
maxexp=  1024   max=        1.7976931348623157e+308
nexp  =    11   min=        -max
---------------------------------------------------------------
```

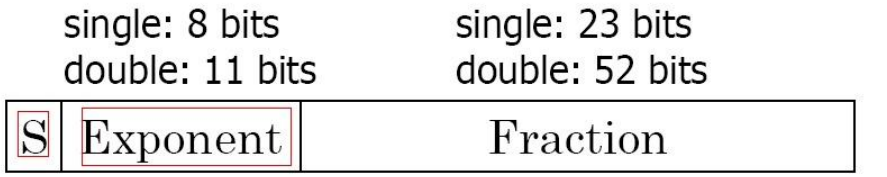
Example - 7-bit positive floating point number

FIGURE 3.8
The smallest possible positive floating-point number from Example 3.5.



IEEE Floating point format (single, double)

IEEE Floating-Point Format



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent}-\text{Bias})}$$

- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single precision: Bias = 127;
  - Double precision: Bias = 1203

Floating point representation consequences

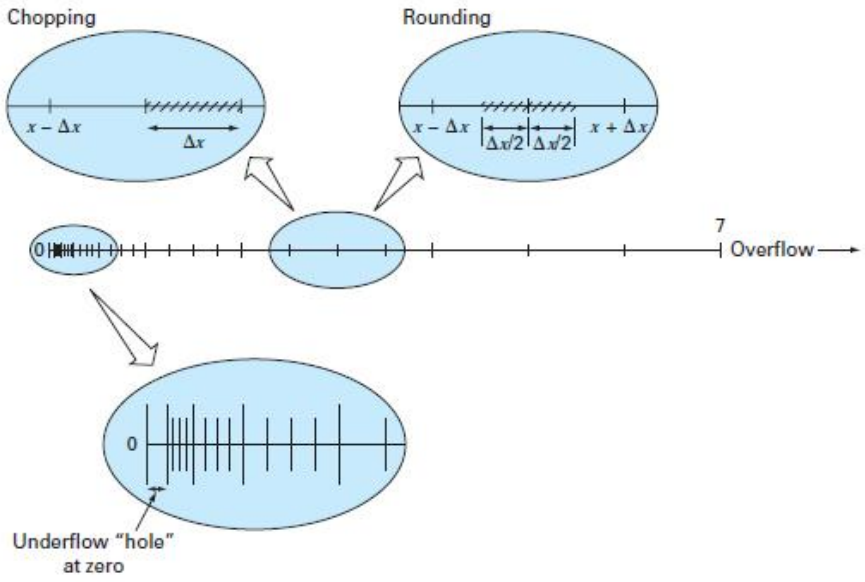
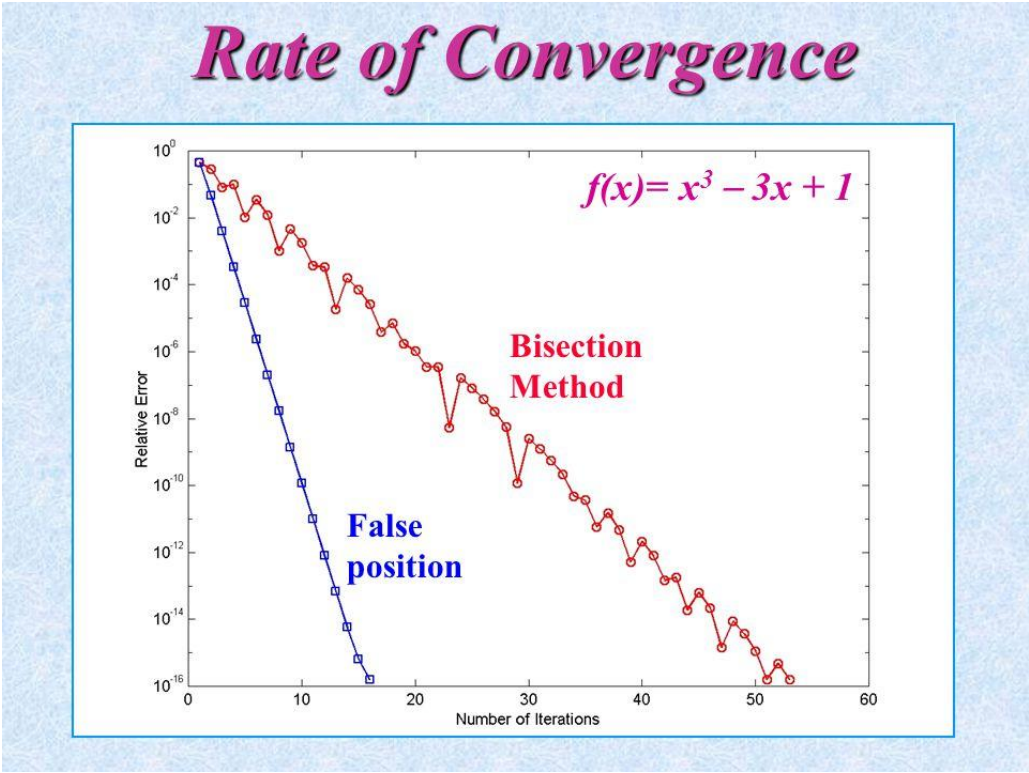


FIGURE 3.9
The hypothetical number system developed in Example 3.5. Each value is indicated by a tick mark. Only the positive numbers are shown. An identical set would also extend in the negative direction.

1. There is *limited range* of quantities that may be represented.
   => Overflow and underflow errors
2. There are only a *finite number of quantities* that can be presented within the range.
   => Truncation (chopping) error
3. The interval between the numbers Δx increases as the numbers grow in magnitude.
   => Rounding error

Characteristics of the numerical method

- Number of initial guesses or starting points
- Rate of convergence
- Stability
- Accuracy and precision
- Breadth of application
- Special requirements
- Programming effort required



http://slideplayer.com/slide/4033924/

Summary

Typically numerical methods *iteratively approximate* the true value. Due to *rounding and truncation* the numerical solutions contain inaccuracies and uncertainties which are collectively called *errors*.

Quite often the *true value* is not known and the error is approximated based on current and previous iterative solutions. The iteration is stopped when the *stop criteria* is achieved.

The numerical methods can be characterized on several ways. Typical characteristics used for comparison of the methods are: the *rate of convergence*, *stability*, *accuracy* and *precision*. In addition, the methods typically require some *initial guesses* for the solution, may have *limitations* in applications and may have some *special requirements*.

Reference

Chapra & Canale. 2010. Numerical Methods for Engineers. 6th Edition. McGraw-Hill. Ch 3. Approximations and Round-Off Errors.

Exercise #1

Is 0.1 + 0.1 + 0.1 equal to 0.3? E.g. what comes out

```
0.1 + 0.1 + 0.1 == 0.3
```

Why this is not true?

How does the computer interpret internally the value 0.1?

How would you then test, if a floating point expression is equal to another expression? (like: is 0.1 + 0.1 + 0.1 equal to 0.3?)

Exercise #2

What is the smallest value that differ from 1.0?

e.g, what is the value of eps, when 1.0 + eps > 1.0?

Does the smallest difference (eps) change, if you change the point of study around 2.0 or 0.0?

e.g. when 2.0 + eps > 2.0?

What is the relationship of x and eps, e.g how does the eps varies when the x increases or decreases (study when x + eps > x)?