Chapter 2 Errors in Numerical Methods

Numerical methods are mathematical techniques used for solving mathematical problems that cannot be solved or are difficult to solve analytically.

Solutions to a math problem can be classified into two types: 1) **Analytical solution**: an exact answer in the form of a mathematical expression in terms of the variables associated with the problem. 2) **Numerical solution**: an approximate numerical value (a number) for the solution.

For a problem to be solved numerically, you may choose several numerical methods which differ in accuracy, time of calculation. Numerical methods are mostly implemented in a computer program (such as MATLAB, C++), we need to know how to represent number on a computer.

2.1 Computer Representation of Numbers

Computers store and process data in binary form.
Binary number: 0 or 1.

Convert Binary number 1001.011 to Decimal form: 1001.011=

Base 10	Conversion	Base 2
1		
2		
4		
8		
9		
25		

Example 2.1 Convert number 25 to binary with MATLAB.

1 Bit: one binary digit (0 or 1). **1 Byte:** Group of 8 bits.
MAX BYTE NUMBER = 1111 1111 =

2.1.1 Integer Number

Computer uses 2 bytes (16 bits), and the 1st bit for sign.

1	1	0	0	0	1	1	0	0	0	0	0	1	1	0	1

Upper limit = $2^{14} + 2^{13} + \ldots + 2^{2} + 2^{1} + 2^{0} = 2^{15} - 1 = 32,767$

Integer range: [-32768 to 32767]

2.1.2 Floating-point Number

Float-point number can be represented by **<u>scientific notation:</u>**
-279.456 =

Binary equivalent of scientific notation:

How does computer store floating-point number? Sign, Biased exponent, Mantissa, i.e. mantissa in binary form, exponent added by bias and in binary form.

sign	biased exponent	mantissa

<u>Single Precision 32-bit</u>

Sign 1 bit	exponent+ bias (127) 8 bits	Mantissa 23 bits

Range:

<u>Double Precision 64-bit</u>

Sign 1 bit	exponent + bias (1023) 11 bits	Mantissa 52 bits

Range:

2.2 Errors in Numerical Solutions

Since numerical solutions are approximated results, we have to specify how different the approximated results are from the true values, i.e. how large the error is.

2.2.1 Error Estimation

The difference between the true value and the approximated value is **<u>error</u>**. Error can be estimated in three ways:

1). **<u>True Error</u>**: The difference between the true solution value and the approximated (numerical) solution value,

$$E_t = \text{true value} - \text{approximated value}$$

2). **<u>True Relative Error</u>**: The percentage of the numerical error over the true value,

$$\varepsilon_t = \left|\frac{\text{true value - approximated value}}{\text{true value}}\right| \times 100\%$$

3). **<u>Estimated Relative Error</u>**: For some problem, the true solution is not known, calculations for a numerical solution are executed in an iterative manner until a desired accuracy is achieved, then estimated relative error is used as a standard to check the solution. The percentage of the difference between the current approximation and preceding approximation over preceding approximation is defined as approximation error,

$$\varepsilon_a = \left|\frac{\text{current approximation - preceding approximation}}{\text{current approximation}}\right| \times 100\%$$

Example 2.2 Determine the true relative error and estimated relative error from approximating of $\boldsymbol{e^{0.5}}$ by using the series $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$ up to 6th term. And write MATLAB code to display the all the true relative errors for each approximation.

true value: >> format long ; exp(0.5)

>> ans =

1.648721 . . .

1st term estimate:

2nd term estimate:

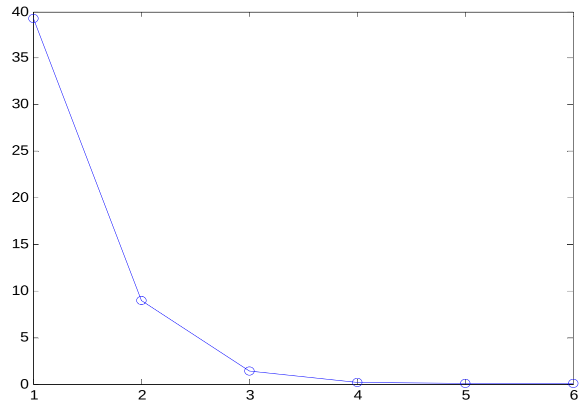
True relative error**:**

Estimated relative error:

Repeat for approximation to 3rd, 4th…term, we can get

Terms	**Results**	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

MATLAB code for plotting the true relative error VS terms kept:



2.3 Error Types

2.3.1 Round-off Error

Specific quantities such as π, or $\sqrt{2}$ cannot be expressed exactly by a limited number of digits.

$$\pi = 3.141592653589793238462643\ldots$$

But computers can retain only a finite number of bits, thus chopping or rounding should be applied to the annoying long number.

Example 2.3 Display π in short, long and long scientific format with MATLAB.

```
>> format short ; pi
>> ans = 3.1416
```
(short data has 5 digits)

```
>> format long ; pi
>> ans = 3.141592653589793
```
(long data has 16 digits)

```
>> format long e ; pi
>> ans = 3.141592653589793e+000
```
(long e data has 16 digits)

Try also exp(1), sqrt(2) by yourself.

Error resulted from omission of the remaining significant figures is called <u>**round-off error**</u>.

<u>**Precision**</u>: All computations in MATLAB are done in double precision by default.

True value $\pi = 3.141592653589793\boxed{238462643\ldots}$

MATLAB pi = 3.141592653589793e+000

The digits in the box have been rounded-off under MATALB environment.

2.3.2 Truncation Errors

Truncation Error: result from using numerical method (approximation) in place of an exact mathematical procedure to find the solution.

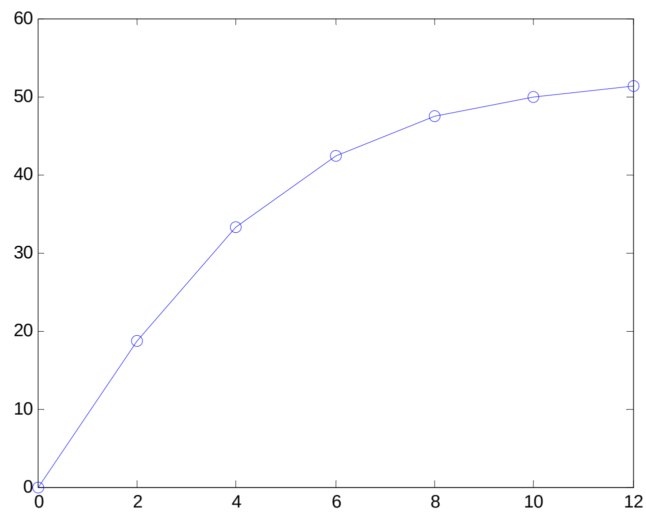
Example 2.4 The velocity with respect to time of bungee jumper is given in the 1st-order differential equation as below. Compute velocity of a free fall bungee jumper with a mass of 70 kg. Use a drag coefficient of 0.25 kg/m.

$$\frac{dv}{dt} = g - \frac{C_d}{m} v^2$$

where v = vertical velocity (m/s), t = time (s), g = gravity acceleration (@ 9.81 m/s^2) c_d = drag coefficient (kg/m), m = jumper's mass (kg).

Analytical Method:

MATLAB code:



Numerical Method:

Rate of change of velocity can be approximated by

$$\frac{dv}{dt} = \frac{\Delta v}{\Delta t} = \frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i}$$

Substitute $dv/dt = g - (c_d/m)v^2$ into above to give

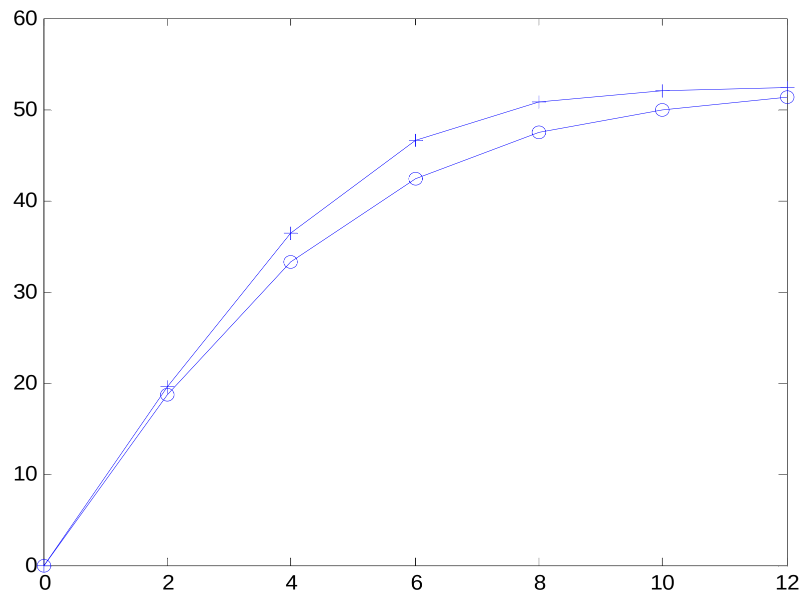
Rearrange equation to yield

Employ a step size $\Delta t = 2$ sec, @ start $t_i = 0$ s, $t_{i+1} = 2$ s, $v(0) = 0$ m/s:

Next step $t_i = 2$ s, $t_{i+1} = 4$ s, $v(2) = 19.62$ m/s:

MATLAB code:

t(s)	v(m/s)
0	0
2	19.62
4	36.49
6	46.60
8	50.71
10	51.96
12	52.30



From the above example, we find that

2.4 Function Approximation with Taylor Series

Taylor series expansion has been taught in your previous math course as:

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

Where $h = x_{i+1} - x_i$, the reminder term R_n accounts for all the terms from n+1 to infinity, and

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1} \qquad \text{where } x_i < \xi < x_{i+1}$$

If the Taylor series is truncated after term n+1, then *f(x+h)* is approximated by

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

The n+1th order truncation error is

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

Where ξ is not known but lies somewhere between x_i and x_{i+1}.

If we let $x_{i+1} = x$, $x_i = 0$, then h = x_{i+1} - x_i = x, thus function *f(x)* is approximated by Taylor series as

$$f(x) \cong f(0) + f^{'}(0)x + \frac{f^{''}(0)}{2!}x^2 + \frac{f^{'''}(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{n!}x^n$$

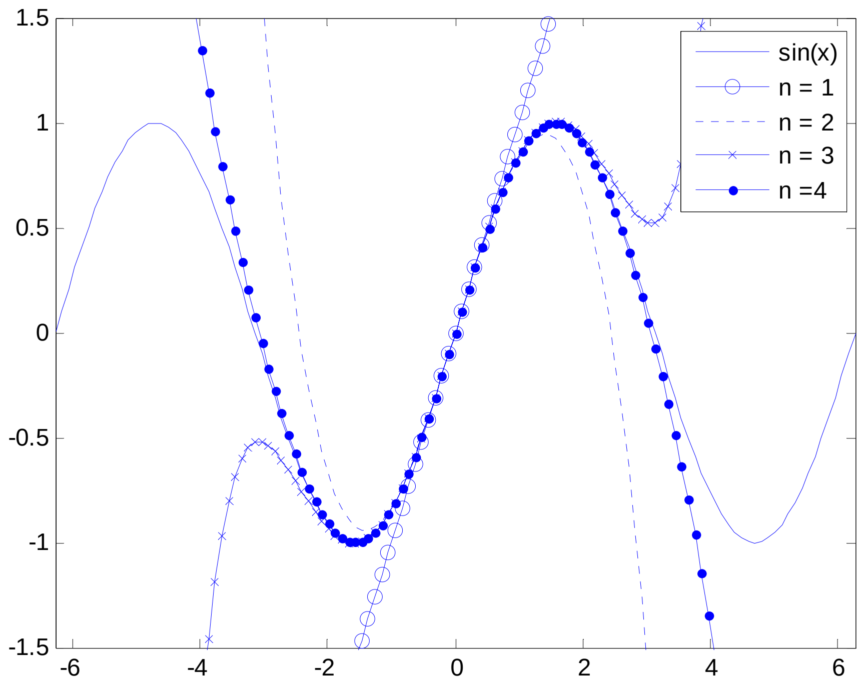
Example 2.5 Determine the Taylor series of exponential function $f(x)=e^x$.

Example 2.6 Plot the Taylor series approximation of *f(x)*=sin*(x)* from n=1 to n=4 in MATLAB.

The Taylor series of sin(x) can be found as

MATLAB code:

```
>> x = -2*pi:pi/30:2*pi;
>> y = sin(x);plot(x,y)
>> axis([-2*pi 2*pi -1.5 1.5])
>> hold on
>> y1 = x; plot(x,y1,'-o')
>> y2 = x-x.^3/6; plot(x,y2,':')
>> y3 = x-x.^3/6+x.^5/120; plot(x,y3,'-x')
>> y4 = x-x.^3/6+x.^5/120-x.^7/120/42; plot(x,y4,'.-')
>> legend('sin(x)','n = 1','n = 2','n = 3','n =4')
```



$\boxed{\textbf{Example 2.7}}$ Use Taylor series expansions with $n = 0$ to 6 to approximate $f(x) = \cos(x)$ at $x_{i+1} = \pi/3$ on the basis of the value of $f(x)$ and its derivatives at $x_i = \pi/4$.

Step size: $h =$ **True value:** $\cos(\pi/3) = 0.5$

Zero-order approximation: $f(\pi/3) \approx$

First-order approximation:

$f(\pi/3) \approx$

Second-order approximation:

$f(\pi/3) \approx$

The process can be continued and the results listed as in

Order n	$f(\pi/3)$	$\lvert\varepsilon_t\rvert$ (%)
0	0.707106781	41.4
1	0.521986659	4.40
2	0.497754491	0.449
3	0.499869147	2.62×10^{-2}
4	0.500007551	1.51×10^{-3}
5	0.500000304	6.08×10^{-5}
6	0.499999988	2.44×10^{-6}