
MATH2089

Numerical Methods

Lecture 1

Part 1 Computing with real numbers

Dr Victoria Timchenko

School of Mechanical and Manufacturing
Engineering (J17)

Part1 Overview

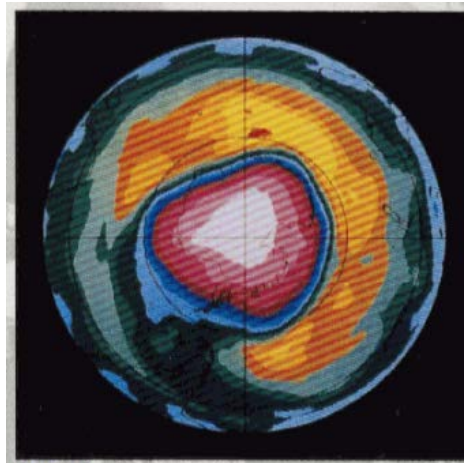
- What is MATH2089 about?
 - What are Numerical Methods?
 - How we approach “Problem solving”
 - Digital representation of numbers
 - Floating Point Arithmetic
 - Error analysis
-

Engineering Grand Challenges



Weather Balloon

Prediction of Weather, Climate and Global Change

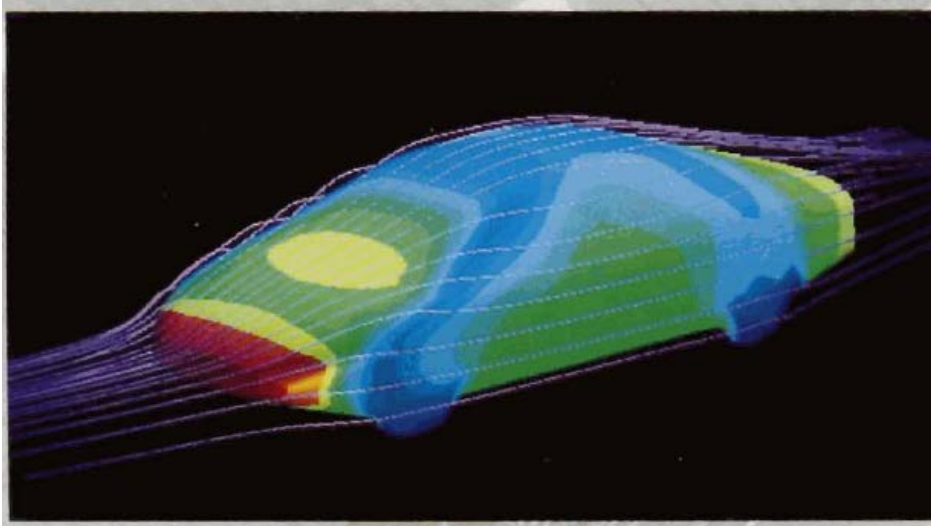


Ozone Model

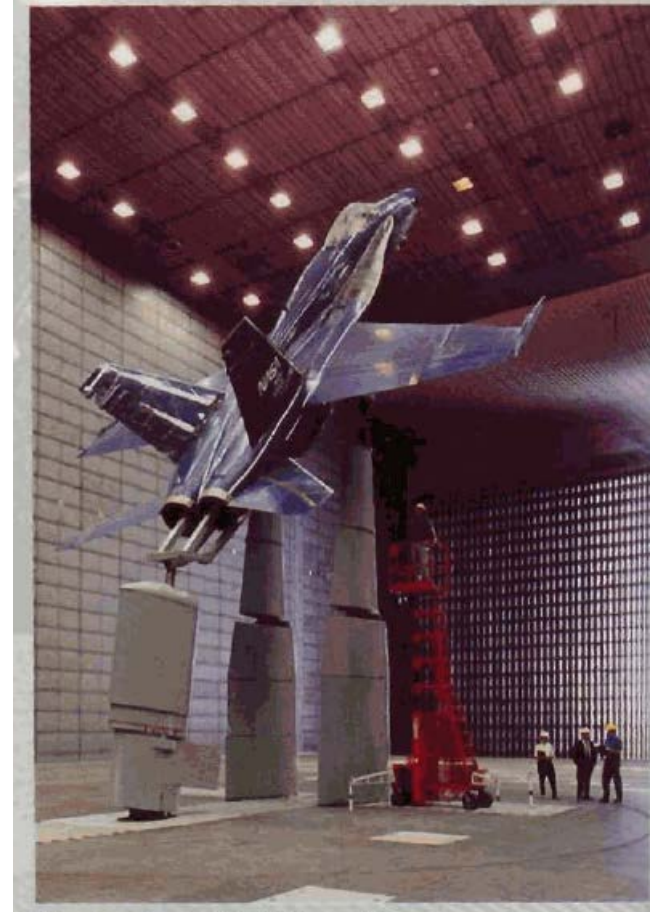


Tornado Machine

Engineering Grand Challenges



**Improvements in
Vehicle Performance**



Wind Tunnel

Objectives of the Course

- **Introduce the basics of numerical analysis**
 - **Develop skills to apply numerical methods for solving practical engineering problems**
-

What are numerical methods for engineers?

- Numerical methods have been used to solve mathematical expressions of engineering and scientific problems for at least 4000 years...
 - Such methods apply numerical approximation in order to convert continuous mathematical problems into systems of discrete equations that can be solved with sufficient accuracy by computer and used to make engineering decisions.
 - This course will provide you with an introduction to several of those numerical methods which you may then find opportunity to practice later.
-

Expected Learning Outcomes

In the end of the course you will be able to:

- Quantify absolute and relative errors;
- Solve nonlinear equations using bisection method, fixed point iteration, Newton-Raphson and secant methods;
- Solve systems of linear equations by Gaussian elimination, LU factorization and iterative methods;
- Perform linear least-squares regression and nonlinear regression as well as numerical differentiation and integration;
- Solve ordinary differential equations and partial differential equations;
- Apply these techniques using Matlab to practical problems in engineering.

Syllabus

Week	Topic	Relevant reading
1	Data representation, error analysis, introduction to MATLAB	1.3-1.6 & class notes
2	Applied MATLAB programming.	class notes
3	Nonlinear equations: bisection method, fixed point iteration, Newton-Raphson and secant methods	2.3 – 2.6, 2.14
4-5	Systems of linear equations: elimination methods, LU factorization, Iterative methods, special linear systems	3.3 - 3.19
6	Interpolation and polynomial approximation, curve fitting	5.5 - 5.6, 5.8 - 5.10
7	Numerical differentiation	7.1 – 7.10
8	Numerical integration	8.1 - 8.12
9	Euler method, Predictor-corrector methods	9.5-9.6, 9.10
10	Runge-Kutta method. Boundary value problems	9.7, 10.5-10.6
11	Parabolic equations. Methods of solutions	11.5
12	Elliptic and hyperbolic equations. Methods of solution	11.4-11.5, 11.9

Assessments

Numerical Methods

ASSESSMENT COMPONENT	DETAILS	MARKS	DUE DATE
Matlab computer Test	Lab Test administrated during labs periods	12	Week 6
Tutorial Quiz	Quiz (30-35 minutes long) administered during tutorials	13	Week 7
Matlab computer laboratory participation	Satisfactory participation in laboratory classes	15	Throughout Semester
Final examination	During exam period. The formal exam scripts will not be returned.	60	June 2018
Total		100	

Labs/Tuts MATH2089

Week2: Lab work Matlab fundamentals (3)

Week3: Tutorial work on data representation, error analysis

Week4: Matlab laboratory work on nonlinear equations (3)

Week5: Tutorial work on nonlinear equations

Week6: 30-35 min Matlab test/ Matlab laboratory work on systems of equations

Week7: 30-35 min tutorial Quizz on nonlinear equations and systems of equations.

Week8: Laboratory work on curve fitting and polynomial approximation (3)

Week9: Tutorial work on numerical differentiation and integration

Week10: Laboratory work on ODE (Euler & Heun methods) (3)

Week11: Tutorial work on ODE

Week12: Laboratory work on PDE (3)

Week13: Tutorial work on PDE & review.

5 labs each 3 =15 marks in total assessment

Numerical and Analytical Methods

$$ax^2 + bx + c = 0$$

Quadratic equation

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

analytical solution:
general, can be used for any a, b, c

$$x = -(bx + c) / ax$$

can be solved iteratively:
gives approximate solution,
does not always work!
Try with a=1, b=-3 and c=2

Why use Numerical Methods?

$$ax + b = 0$$

Can be solved analytically

$$ax^2 + bx + c = 0$$

$$ax^2 + b \sin x + c = 0$$

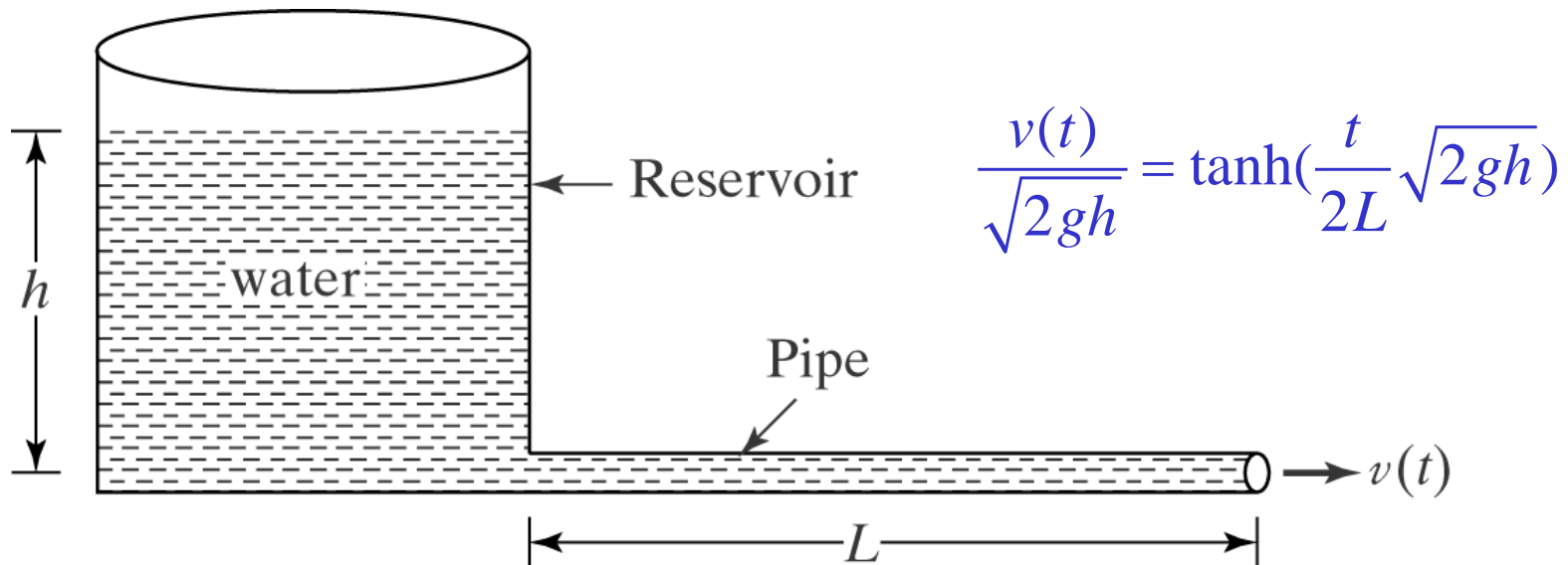
Cannot be solved analytically

$$ax^{2.1} + bx + c = 0$$

**Real engineering problems are complex
and described by non-linear equations –
no analytical solutions,
Have to be solved by numerical methods!**

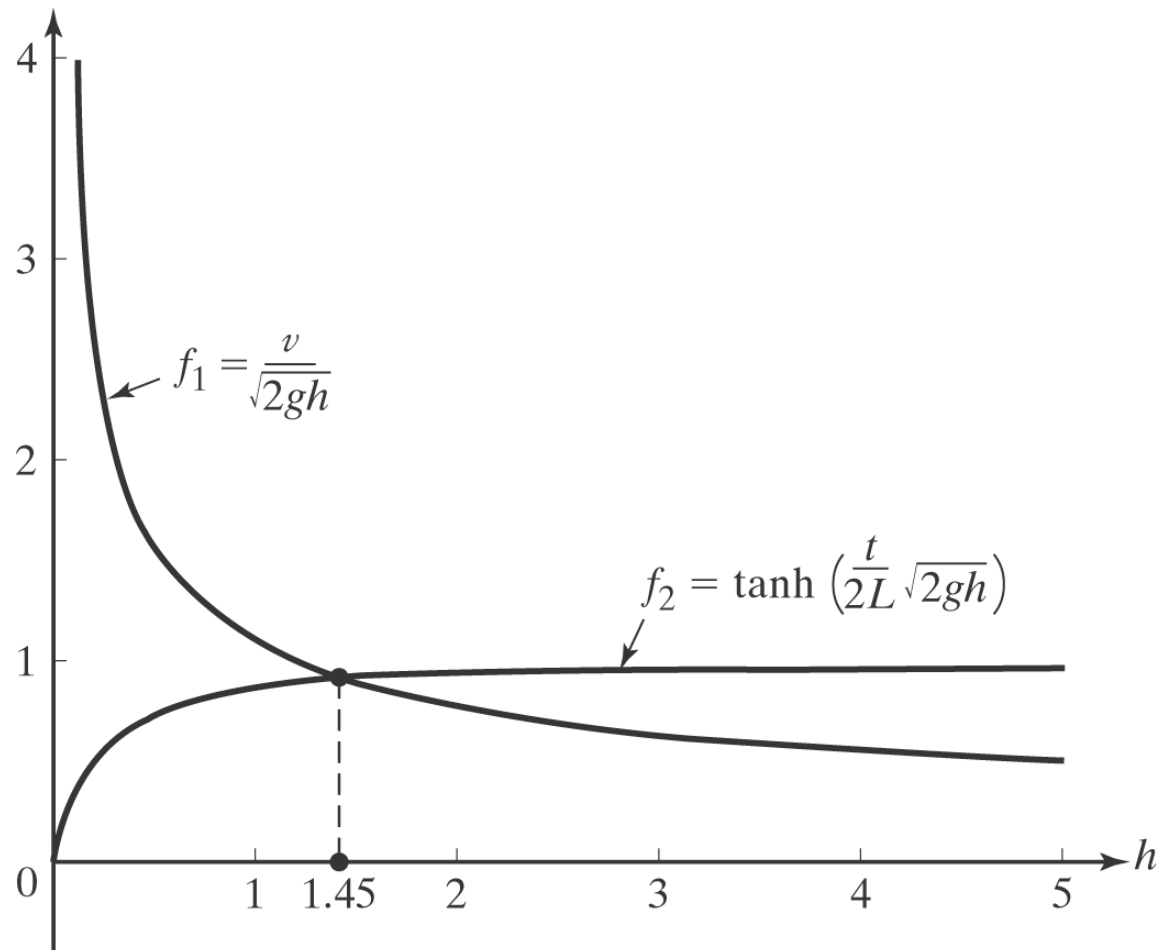
When are Numerical Methods used?

Example: Discharge of water from reservoir



Find the value of h necessary for achieving a velocity of $v = 5$ m/s at time $t = 3$ s when $L = 5$ m

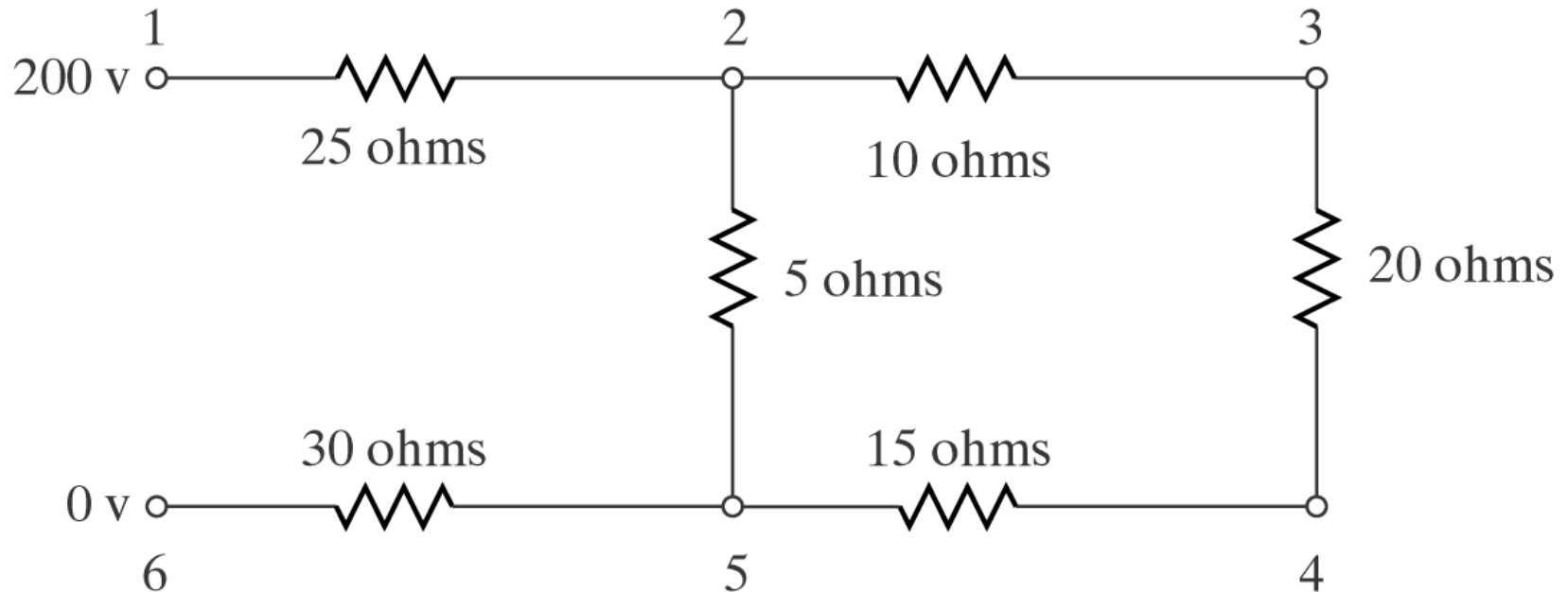
When are Numerical Methods used?



Solution of transcendental equation

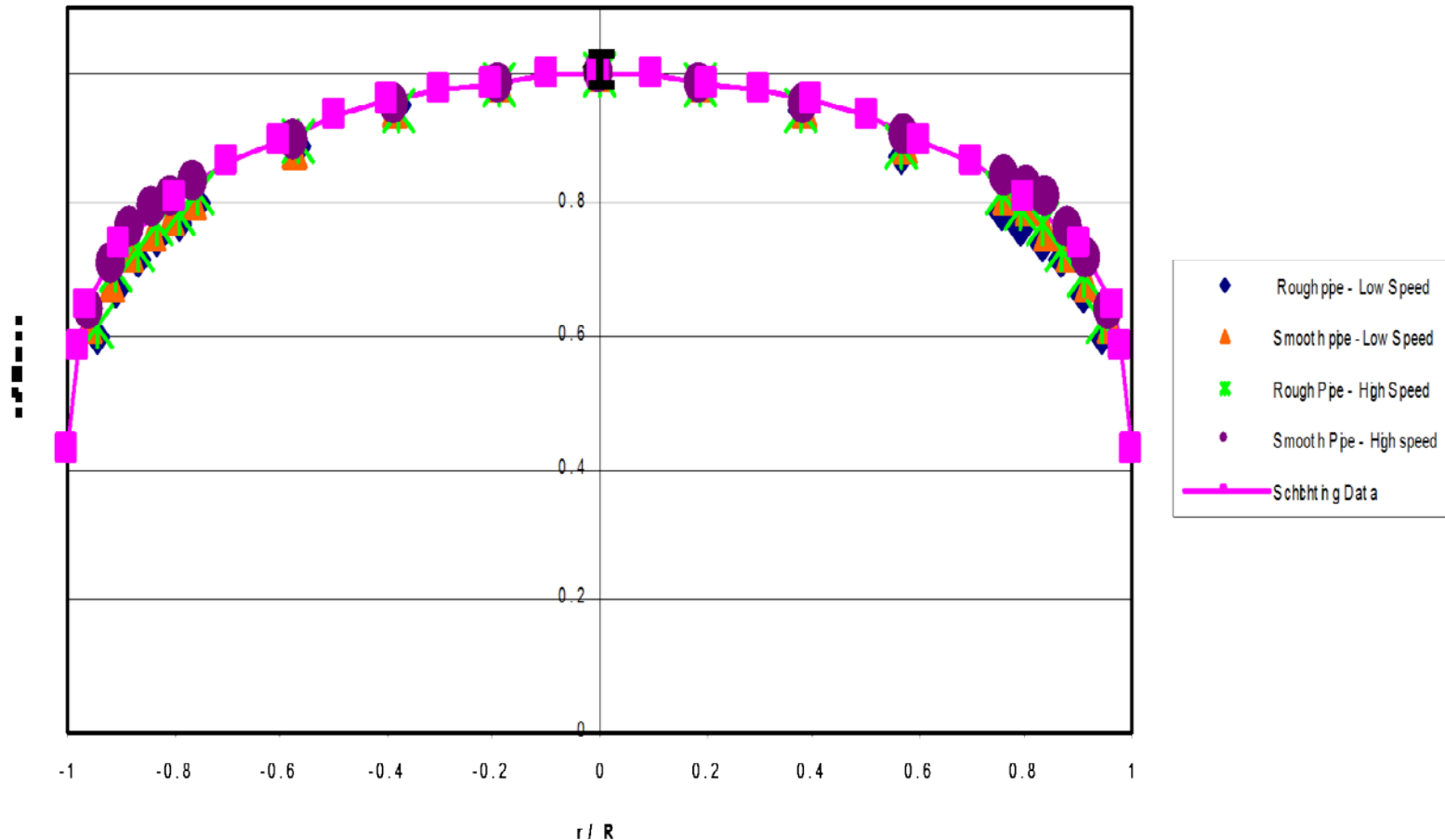
When are numerical methods used?

Example: Electrical Circuit Analysis



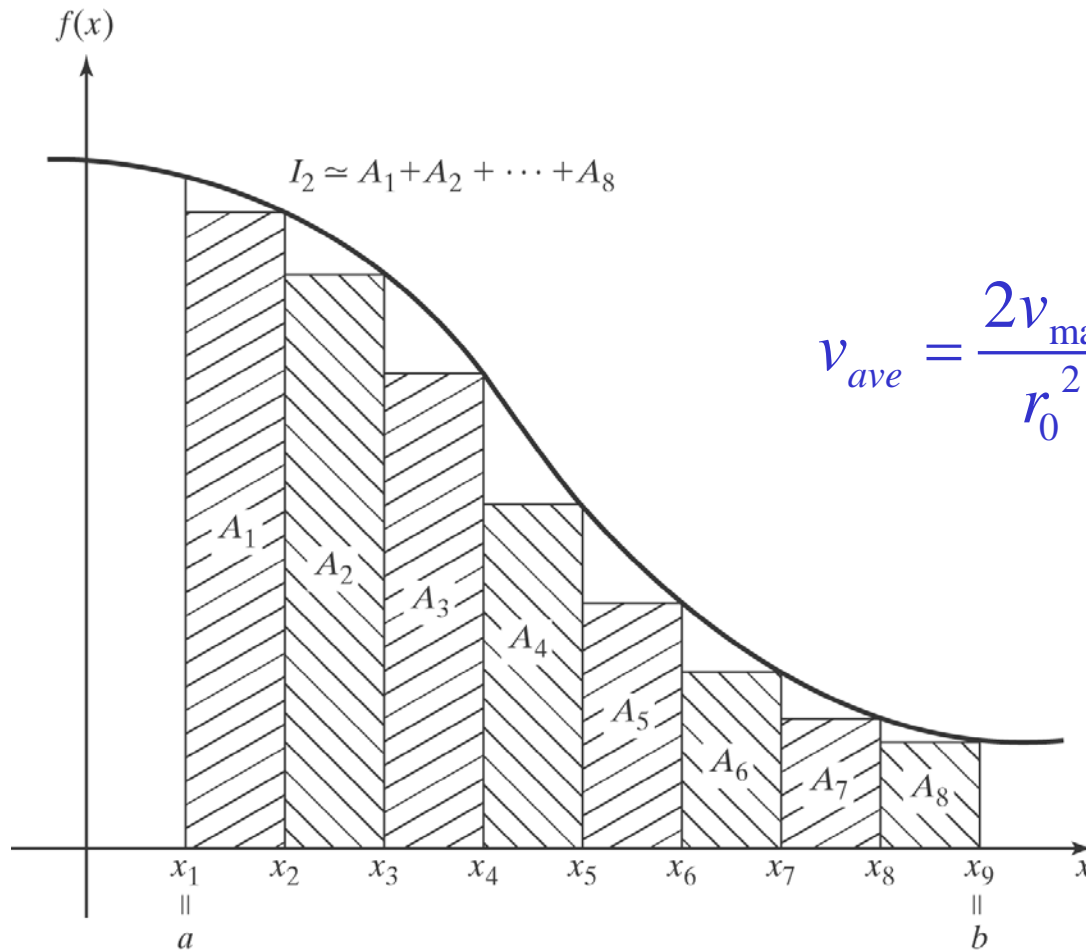
Solution of system of linear equations

When are Numerical Methods used?



Benchmark data for velocity profile (Schlichting, 1968)

When are Numerical Methods used?



$$v_{ave} = \frac{2v_{\max}}{r_0^2} \int_0^{r_0} r \left(1 - \frac{r}{r_0}\right)^{1/n} dr$$

Numerical evaluation of integral I_2

Mathematical Modeling and Problem Solving

1. ***Problem definition:*** State the problem clearly, including any simplifying assumptions
 2. ***Mathematical model:*** Develop a mathematical statement of the problem in a form that can be solved for a numerical answer
 3. ***Problem solving tools (methods):*** Solve the equation(s) using graphics, statistics, computer, analytical methods, *numerical methods*, etc.
 4. ***Results and discussion:*** Present and interpret the numerical results. This is the hardest part of solving problems and must be learned on the job.
 5. ***Conclusion:*** Based on previously discussed results arrive at a decision
-

Example: Problem Solving

A parachutist of mass 68 kg jumps out of a stationary hot air balloon. Compute velocity prior to opening the parachute. The drag coefficient $c = 12.5 \text{ kg/s}$ and the gravitational constant $g = 9.8 \text{ m/s}^2$.



"It's not so bad...At least the parachute opened!..."

Example (cont.)

Problem definition

- Determine the terminal velocity of a free-falling body near the earth's surface
- Air resistance is linearly proportional to velocity, $F_U = cv$
- Initial condition of $v = 0$ at $t = 0$
- v is dependent variable, t is independent variable

Example (cont.)


Mathematical model

➤ Newton's second law: $\sum F = ma$

$$\frac{dv}{dt} = \frac{\sum F}{m}$$

$$\sum F = F_D - F_U = mg - cv$$

$$\frac{dv}{dt} = g - \frac{c}{m}v$$

$$F_U = cv$$


Initial condition $v(t = 0) = 0$

Example (cont.)

Method to solve the problem $\frac{dv}{dt} = g - \frac{c}{m}v$

a) Analytical method:

$$\int \frac{dv}{g - \frac{c}{m}v} = \int dt$$

where A is an integrating constant

$$\frac{m}{c} \ln \left(g - \frac{c}{m}v \right) = t + A$$

Using initial condition $v(t=0) = 0 \longrightarrow A = -\frac{m}{c} \ln g$

Analytical solution: $v(t) = \frac{gm}{c} \left(1 - e^{-(c/m)t} \right)$ or $v(t) = 53.31 \left(1 - e^{-0.1838t} \right)$

Example (cont.)

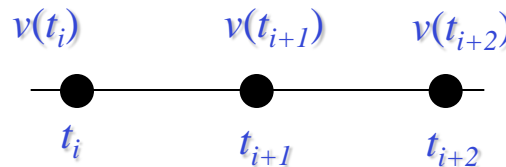
b) Numerical method :

The time rate change of velocity can be approximated as:

$$\frac{v(t_{i+1}) - v(t_i)}{t_{i+1} - t_i} = g - \frac{c}{m} v(t_i)$$

Numerical solution:

$$\underbrace{v(t_{i+1})}_{\text{New value}} = \underbrace{v(t_i)}_{\text{Old value}} + \left[g - \frac{c}{m} v(t_i) \right] \underbrace{(t_{i+1} - t_i)}_{\text{step size } \Delta t}$$



$i=0$ corresponds to $t=0$

$$v(t_{i+1}) = v(t_i) + [9.8 - 0.1838v(t_i)](t_{i+1} - t_i)$$

Example (cont.)

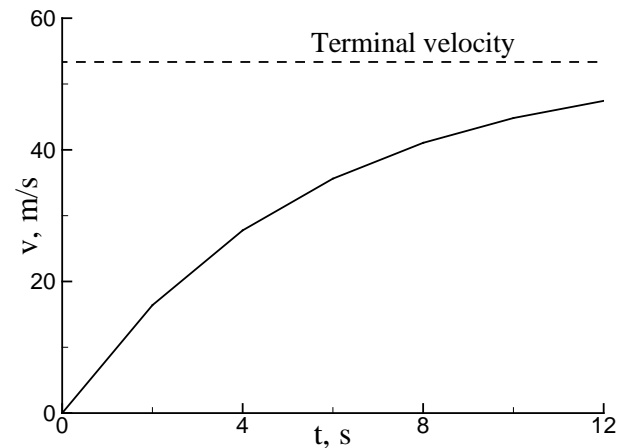
Results and discussion

a) Analytical solution:

- The parachutist accelerates rapidly. A velocity of 44.83 m/s (161.4 km/h) is attained after 10 s.
- After a sufficient long time, the terminal velocity of 53.31 m/s (191.9 km/h) is reached. This shows that the force of gravity will be in balance with the air resistance, $F_D = F_U$

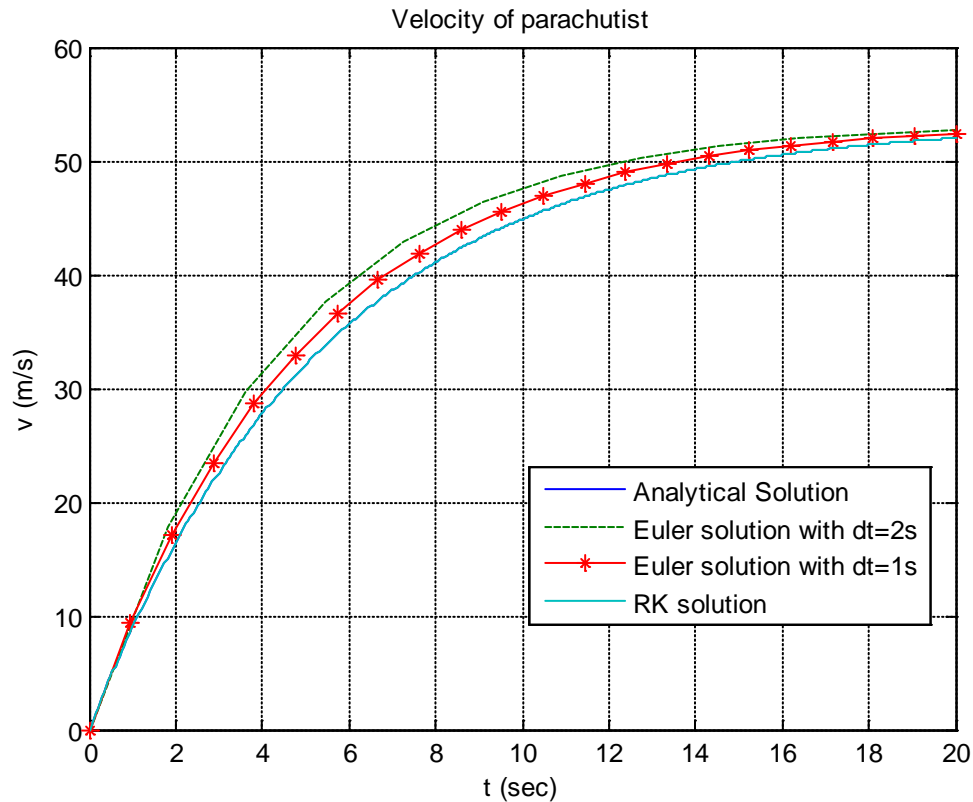
Or $mg = cv_{\text{terminal}}$

t, s	$v, m/s$
0	0.00
2	16.40
4	27.76
6	35.62
8	41.06
10	44.83
12	47.44
∞	53.31



Example: (cont.)

b) Numerical solution:



Example: (cont.)

Conclusion

Based on the obtained results the numerical solution obtained using the RK method shows the closest agreement with analytical results and therefore is the most accurate.

Data Representation

In a digital computer, all information is stored in binary form. A **bit** is a binary digit (i.e., a zero or one). A **byte** is larger unit, usually consists of 8 bits. A **word** consists of bytes, for example 32-bit word consists of 4 bytes

Most computers do arithmetic using the binary (base 2) number system. They convert input (which is in base 10 numbers) to base 2, then perform base 2 arithmetic and finally translate the answer into base 10

In decimal form:

$$N = (a_k \times 10^k) + (a_{k-1} \times 10^{k-1}) + \dots + (a_1 \times 10^1) + (a_0 \times 10^0)$$

$$N = a_k a_{k-1} \dots a_2 a_1 a_{0ten}$$

In binary form:

$$(N)_{10} = (a_k \times 2^k) + (a_{k-1} \times 2^{k-1}) + \dots + (a_1 \times 2^1) + (a_0 \times 2^0)$$

$$N = a_k a_{k-1} \dots a_2 a_1 a_{0two}$$

Binary Storage of Integers

- Integers can be **exactly** represented by base 2

Limitations:

- A finite number of bits are used to store each value in computer memory
- Limiting the number of bits limits the size of integer that can be represented

Determine the number in binary system:

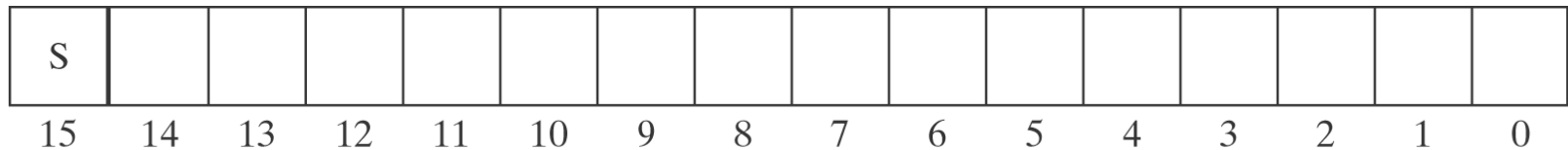
$$(111)_2 \quad ? \quad = 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 = 7$$

$$(1111)_2 \quad ?$$

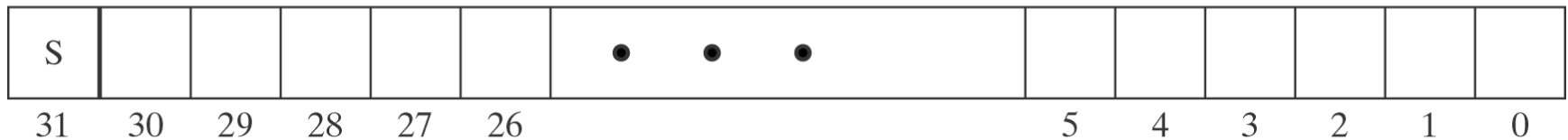
$$(11111)_2 \quad ?$$

Internal Representation of Integers in Memory

Can be pictured as an ordered sequence of storage locations called memory cells



(a) 16-bit data word



(b) 32-bit data word

$$2^{15} - 1 \text{ to } -2^{15} \quad (32767 \text{ to } -32767)$$

$$2^{31} - 1 \text{ to } -2^{31}$$

Floating Point Arithmetic

All computers designed since 1985 use IEEE floating-point arithmetic.

Most nonzero floating-point numbers are normalized. This means they can be expressed as

$$x = \pm(1 + f) \cdot 2^e$$

The quantity f is the fraction or **mantissa** and e is the **exponent**. The fraction satisfies: $0 \leq f < 1$ and uses at most 52 bits.

$$0 \leq 2^{52}f < 2^{52}$$

The exponent e is an integer in the interval

$$-1022 \leq e \leq 1023$$

Floating Point Binary Representation

A double precision (64 bit) floating point real number in memory can be schematically represented as



The finite number of bits in the exponent limits the magnitude or **range** of the floating point numbers.

The finite number of bits in the mantissa limits the number of significant digits or the **precision** of the floating point numbers.

Digital Storage of Floating Point Numbers

The floating point mantissa is expressed in powers of $1/2$

$$\left(\frac{1}{2}\right)^0 = 1 \text{ is not used} \quad \left(\frac{1}{2}\right)^1 = 0.5 \quad \left(\frac{1}{2}\right)^2 = 0.25 \quad \left(\frac{1}{2}\right)^3 = 0.125$$

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{0}{2^6} + \frac{0}{2^7} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{0}{2^{10}} + \frac{0}{2^{11}} + \frac{1}{2^{12}} + \dots$$

The binary mantissa for 0.1 is $(00011\ 0011\ \dots)_2$.

The decimal value of 0.1 cannot be represented by a finite number of binary digits.

Consequences of Finite Storage

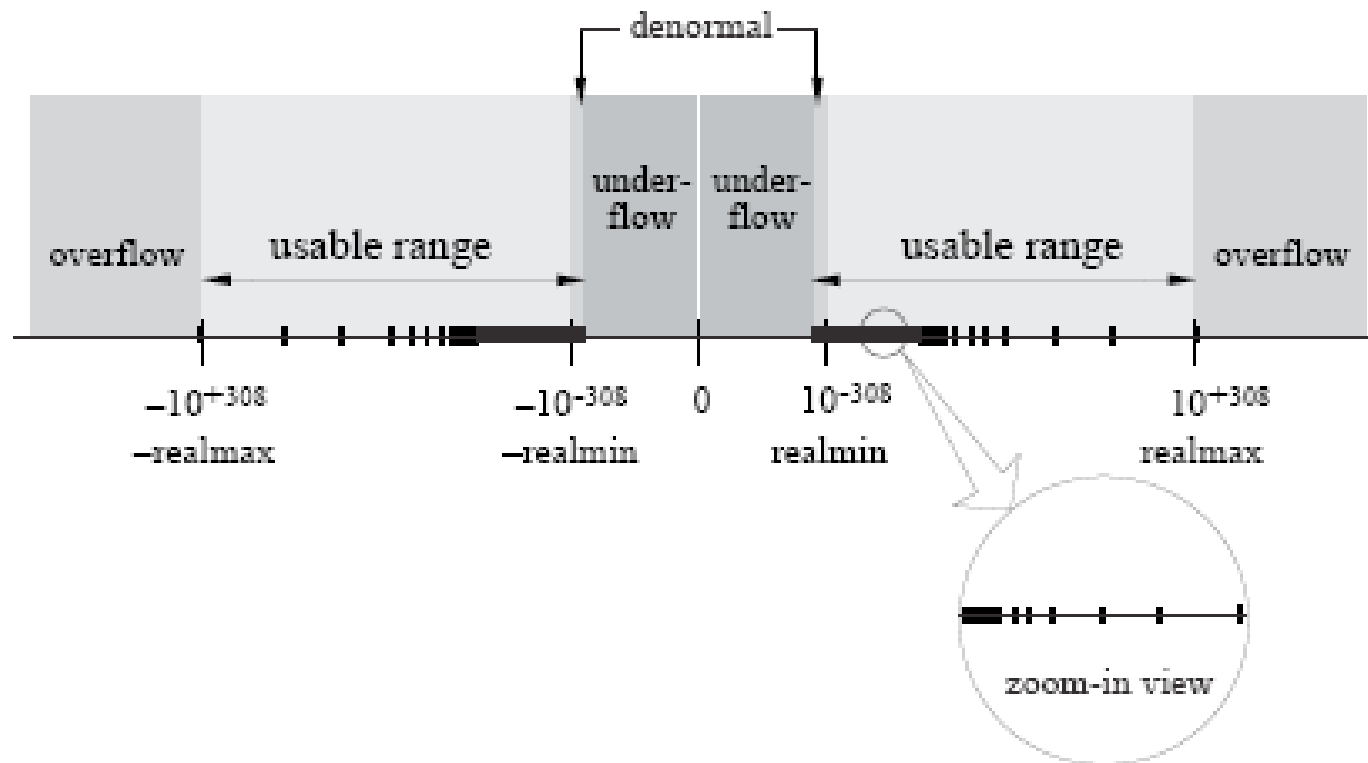
Limiting the number of bits allocated for storage of the exponent \Rightarrow Upper and lower limits on the **range** (or magnitude) of floating point numbers

Limiting the number of bits allocated for storage of the mantissa \Rightarrow Limit on the **precision** (or number of significant digits) for any floating point number.

Most real numbers cannot be stored exactly - they do not exist on the floating point number line

Floating Point Number Line

It is a subset of the real number line



Matlab:	<code>realmin</code>	2^{-1022}	$2.2251\text{e-}308$
	<code>realmax</code>	$(2-\text{eps}) \cdot 2^{1023}$	$1.7977\text{e+}308$

Machine Precision

The magnitude of round-off errors is quantified by machine precision ε_m .

There is a number, $\varepsilon_m > 0$, such that

$$1 + \delta = 1 \quad \text{whenever } |\delta| < \varepsilon_m.$$

In exact arithmetic $1 + \delta = 1$ only when $\delta = 0$, so in exact arithmetic ε_m is identically zero.

Matlab uses double precision (64 bit) arithmetic. The built-in variable **eps** stores the value of ε_m

$$\mathbf{eps} = 2.2204 \times 10^{-16}$$

Underflow, Overflow and Not-a-Number

Any number that is less than the smallest floating point number **underflows** to 0

Any number larger than the largest floating point number is said to overflow and resulted in **Inf**

Any result that is mathematically not defined is represented by **NaN**

Floating Point Arithmetic

Floating Point Arithmetic

Operation	Floating Point Value is . . .
$2.0 + 2.0 = 4$	exact
$9.0 \times 7.0 = 63$	exact
$\frac{12.0}{3.0} = 4$	exact
$\frac{29}{13} = 2.230769230769231$	approximate
$\frac{29}{1300} = 2.230769230769231 \times 10^{-2}$	approximate

Floating Point Arithmetic in MATLAB

```
>>x=29/1300
```

```
x=0.223
```

```
>>y=29-1300*x
```

```
y=3.5527e-015
```

In exact arithmetic the value of y should be zero.
What is going on?

The Patriot Missile Failure

On February 25, 1991, during the Gulf War, an American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. The Scud struck an American Army barracks, killing 28 soldiers and injuring around 100 other people.

It turns out that the cause was an inaccurate calculation of the time since boot due to computer arithmetic errors. Specifically, the time in tenths of second as measured by the system's internal clock was multiplied by 1/10 to produce the time in seconds. This calculation was performed using a 24 bit fixed point register.

24 bit timer stored 0.000110011001100110011001100

Error = 0.0000000000000000000000000011001100 . . . binary

$\approx 9.5 \times 10^{-8}$

**Timer running 100 hours : error = $9.5 \times 10^{-8} \times 100 \times 60 \times 60 \times 10 = 0.34$
secs**

Scud 1, 676 m/sec : error > 0.5 km



Error Analysis

If \tilde{x} is an approximation to x

The absolute error is $E_x = x - \tilde{x}$

The relative error is $R_x = \frac{x - \tilde{x}}{x}$, $x \neq 0$.

A number \tilde{x} is considered to be an approximation to the true value x to d significant digits if d is the largest positive integer for which

$$\left| \frac{x - \tilde{x}}{x} \right| < \frac{1}{2} 10^{-d}$$

Sources of Error

- (1) ***Errors in mathematical modelling:*** the simplifying approximations and assumptions made in representing a physical system by mathematical equation introduce error.
- (2) ***Blunders:*** the programming errors, if undetected, introduce errors to the computed values.
- (3) ***Errors in input:*** due to the errors in data transfer and the uncertainties associated with measurements.
- (4) ***Machine errors:*** the floating point representation of numbers involves rounding and chopping errors, as well as underflow and overflow errors. These errors are introduced at each arithmetic operation during computation.
- (5) ***Truncation errors associated with the mathematical process:*** for example, the approximate evaluation of an infinite series or an integral involving infinity as a limit of integration involve computational errors.

Round-off Error

Since only a finite number of digits are stored in a computer, the actual numbers may undergo chopping or rounding of the last digit.

Effects of round-off usually accumulate slowly, but . . .

Subtracting nearly equal numbers leads to **severe loss of precision**.

A similar **loss of precision** occurs when **two numbers of very different magnitude** are added.

Round-off is inevitable: good algorithms minimize the effect of round-off!

Catastrophic Cancellation Errors

Evaluate $c = a + b$ with

$a = x.xxx \dots \times 10^0$ and

$b = y.yyy \dots \times 10^{-8}$

$$\begin{array}{r} \text{available precision} \\ \overbrace{x.xxx \ xxxx \ xxxx \ xxxx} \\ + \quad 0.000 \ 0000 \ yyyy \ yyyy \ yyyy \ yyyy \\ \hline = \quad x.xxx \ xxxx \ zzzz \ zzzz \ \underbrace{yyyy \ yyyy}_{\text{lost digits}} \end{array}$$

The most significant digits of a are retained, but the least significant digits of b are lost because of the mismatch in magnitude of a and b

Catastrophic Cancellation Errors (con't)

Consider $c = a - b$ with

$$a = x.\text{xxxxxxxxxxxx}1\text{ssssss}$$

$$b = x.\text{xxxxxxxxxxxx}0\text{ttttt}$$

where x , y , s and t are decimal digits.

The digits $\text{sss} \dots$ and $\text{ttt} \dots$ are lost when a and b are stored in double-precision, floating point format

Roots of Quadratic Equation

The roots of the quadratic equation: $ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

When is this formula susceptible to catastrophic cancellation?

Find a mathematically equivalent formula that removes this difficulty

Taylor Series

For a continuous function $f(x)$ defined on the interval $x \in [a, b]$ we define the n^{th} order Taylor Series approximation $P_n(x)$

$$P_n(x) = f(x_0) + (x - x_0) \left. \frac{df}{dx} \right|_{x=x_0} + \frac{(x - x_0)^2}{2!} \left. \frac{d^2 f}{dx^2} \right|_{x=x_0} + \dots + \frac{(x - x_0)^n}{n!} \left. \frac{d^n f}{dx^n} \right|_{x=x_0}$$

Then there exists ξ with $x_0 \leq \xi \leq x$ such that

$$f(x) = P_n(x) + R_n(x) \quad \text{where} \quad R_n(x) = \frac{(x - x_0)^{(n+1)}}{(n+1)!} \left. \frac{d^{(n+1)} f}{dx^{(n+1)}} \right|_{x=\xi}$$

Big “O” Notation – Same Order of Magnitude

Big “O” notation

$$f(x) = P_n(x) + \mathcal{O}\left(\frac{(x - x_0)^{(n+1)}}{(n + 1)!}\right)$$

or, for $x - x_0 = h$ we say

$$f(x) = P_n(x) + \mathcal{O}\left(h^{(n+1)}\right)$$

Truncation Error

Truncation error is the discrepancy introduced by the use of an approximate expression in place of an exact mathematical expression or formula. For example consider the Taylor series expansion of the function $\ln(1+x)$

$$\begin{aligned} y(x) = \ln(1+x) &= \sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{i} x^i = \\ &= x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4 + \frac{1}{5}x^5 - \frac{1}{6}x^6 + \dots \end{aligned}$$

Let the function $y(x)$ be approximated by the first four terms of the Taylor's series expansion. The resulting discrepancy between the exact function $y(x)$ and the approximate function is called the **truncation error**

Truncation Error- Example

Find the truncation error in approximating the function $y(x) = \ln(1+x)$ by

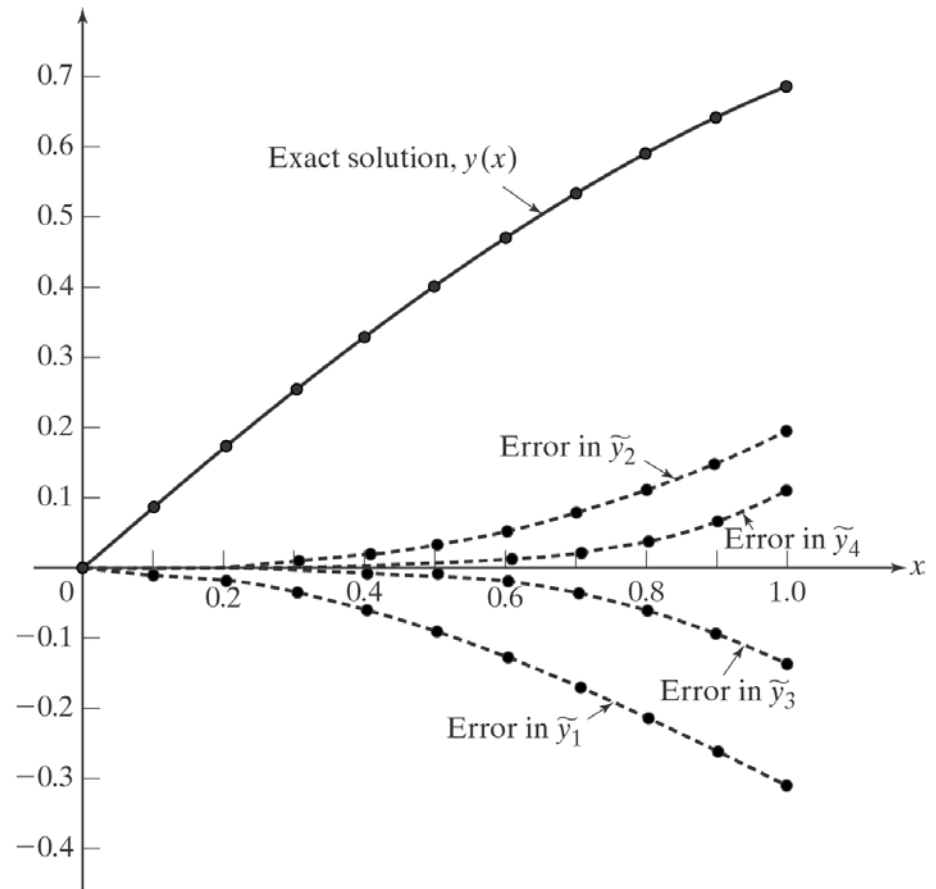
$$\tilde{y}_1(x) = x$$

$$\tilde{y}_2(x) = x - \frac{1}{2}x^2$$

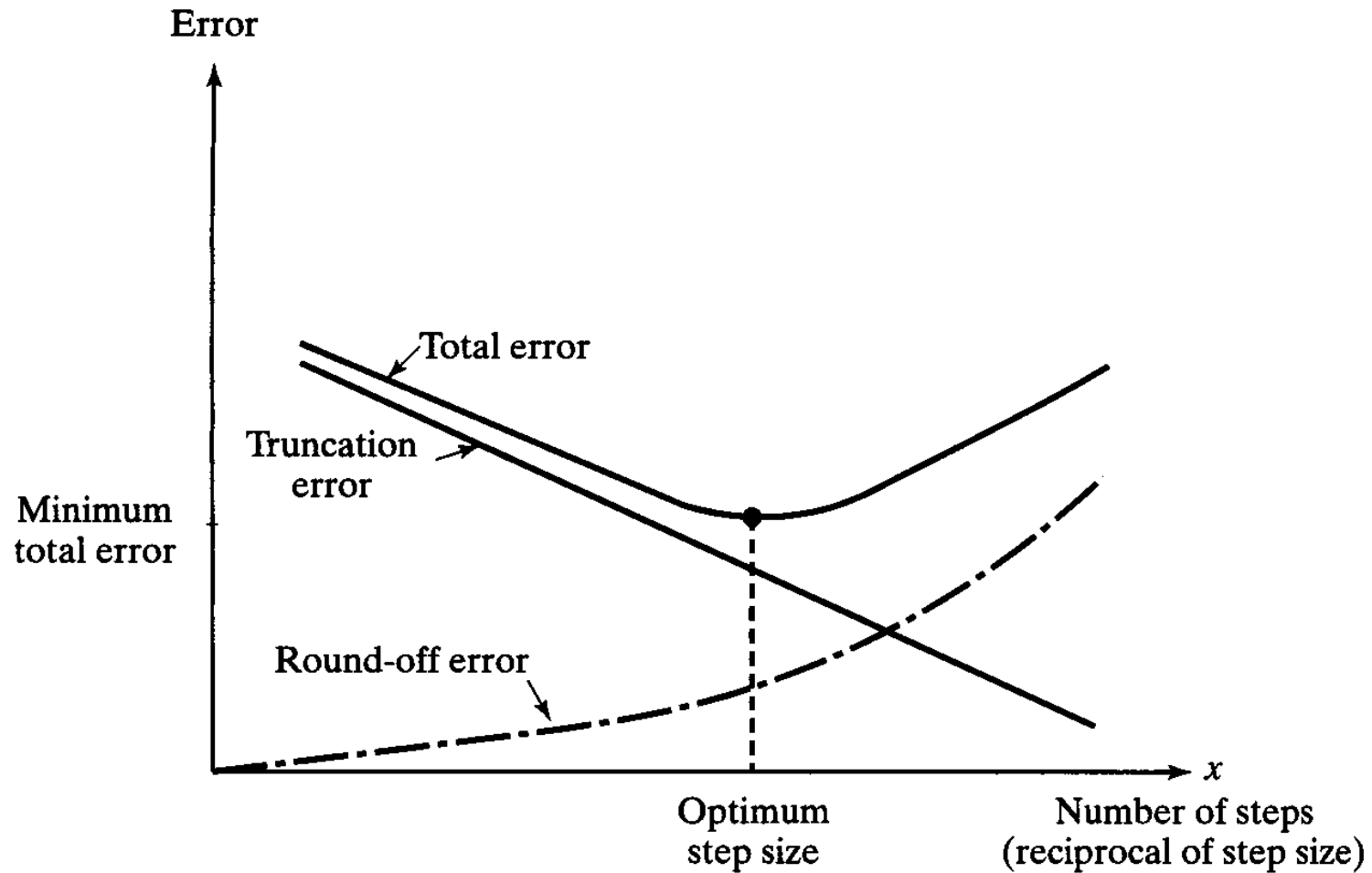
$$\tilde{y}_3(x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3$$

$$\tilde{y}_4(x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \frac{1}{4}x^4$$

Over the range $0 < x < 1$



Total Numerical Error



Practical Considerations

1. x has a relative error of 10^{-p} - means that x has approximately p correct decimal digits
2. Never give an answer to more accuracy than your input data
3. What error is acceptable in your engineering computation?
10% or relative error of 0.1
6 significant decimal digits \Leftrightarrow relative error $\leq 0.5 \times 10^{-6}$

Practical Considerations (con't)

4. Avoid subtracting two nearly equal numbers.
 5. When adding and subtracting numbers, sort the numbers and work with the smallest numbers first
 6. Use Taylor series to analyze total numerical errors.
 7. Try to estimate the accuracy of your numerical results. Check if the results satisfy some condition or equation.
 8. Employ sensitivity analysis and compare the results with a different step size, method, model parameter or input values.
-

Part 2 Starting with MATLAB

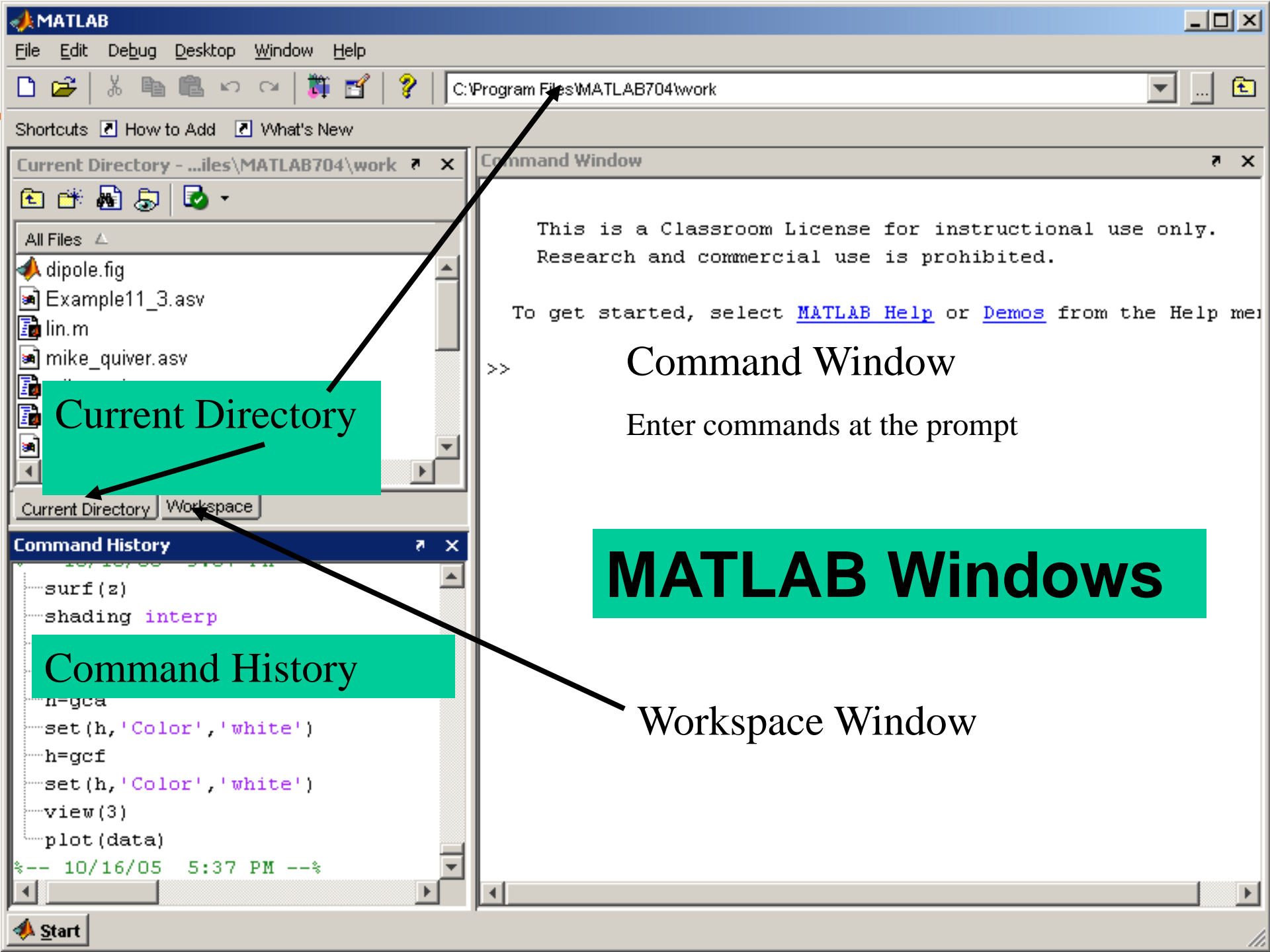
Matlab stands for **Matrix laboratory**

Some Advantages:

- No need to define the class and size of a variable before using it
- No special commands and/or procedures are required for running programs with size over 655,360 bytes
- Fast and convenient for matrix operations.
- Lots of powerful functions and toolboxes for calculation and presentation
- Very comprehensive online help

Some Disadvantages:

- Looping is relatively slow in Matlab when compared with FORTRAN, Pascal and C
- Matlab programs must be run in Matlab environment! Matlab alone can NOT produce any standalone executable (exe) program
- This problem may be solved by the Matlab Compiler Toolbox! This toolbox allows users to translate simple Matlab m-files to C (or C++) source code files. Then, one can use existing C compiler to produce the corresponding executable (exe) files



Current Directory

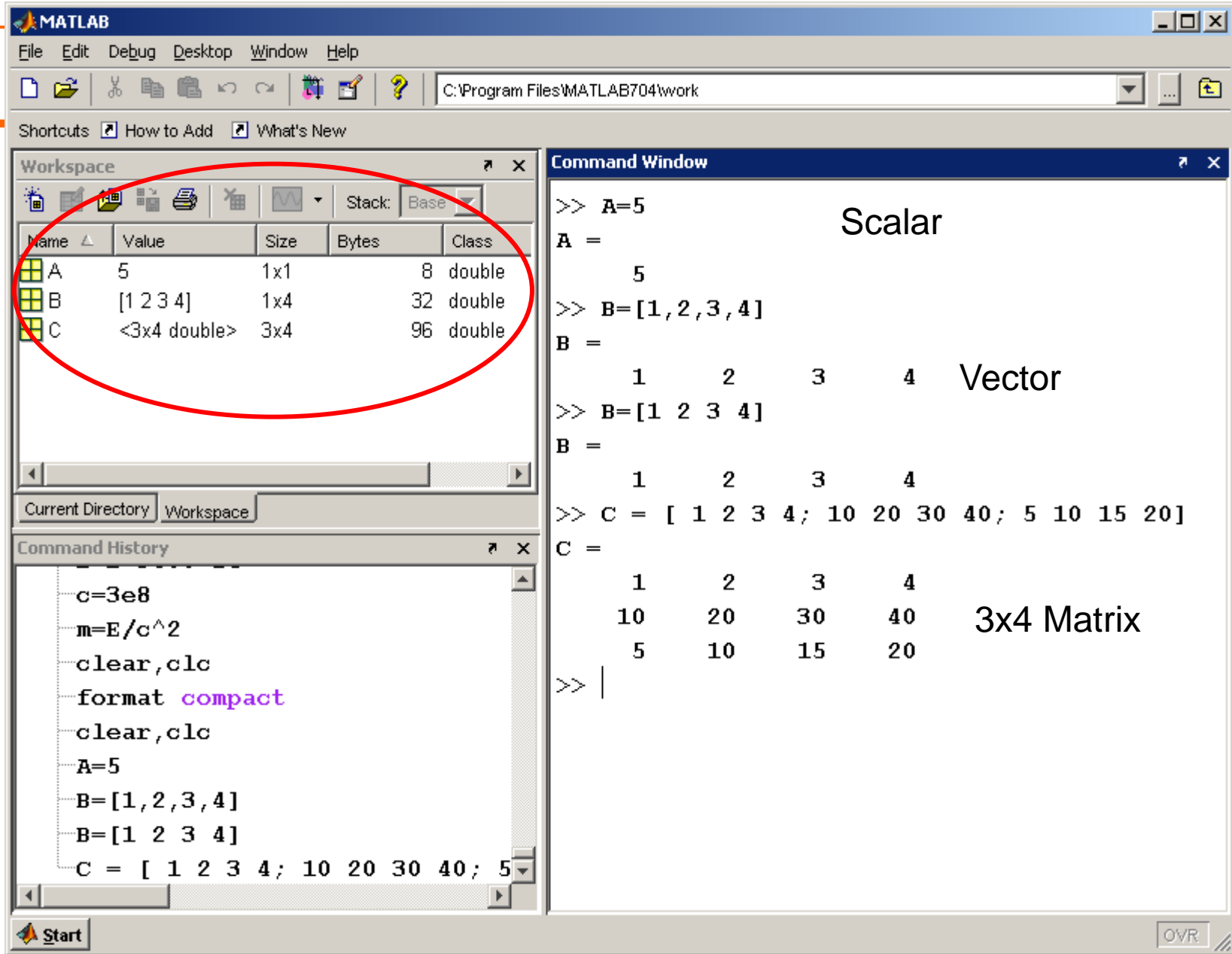
Command History

Command Window

Enter commands at the prompt

MATLAB Windows

Workspace Window



When you define variables in the command window, they are listed in the workspace window

The image displays the MATLAB software interface with several windows open. The **Workspace** window shows variables A, B, C, and ans. The **Documents** window shows a grid for editing variable C. The **Command History** window shows the commands entered. The **Array Editor** window shows the details of variable C.

Workspace Window:

Name	Value	Size	Bytes	Class
A	5	1x1	8	double
B	[1 2 3 4]	1x4	32	double
C	<3x4 double>	3x4	96	double
ans	-1	1x1	8	double

Documents Window (C):

	1	2	3	4	5	6	7	8
1	1	2	3	4				
2	10	20	30	40				
3	5	10	15	20				
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

Command History Window:

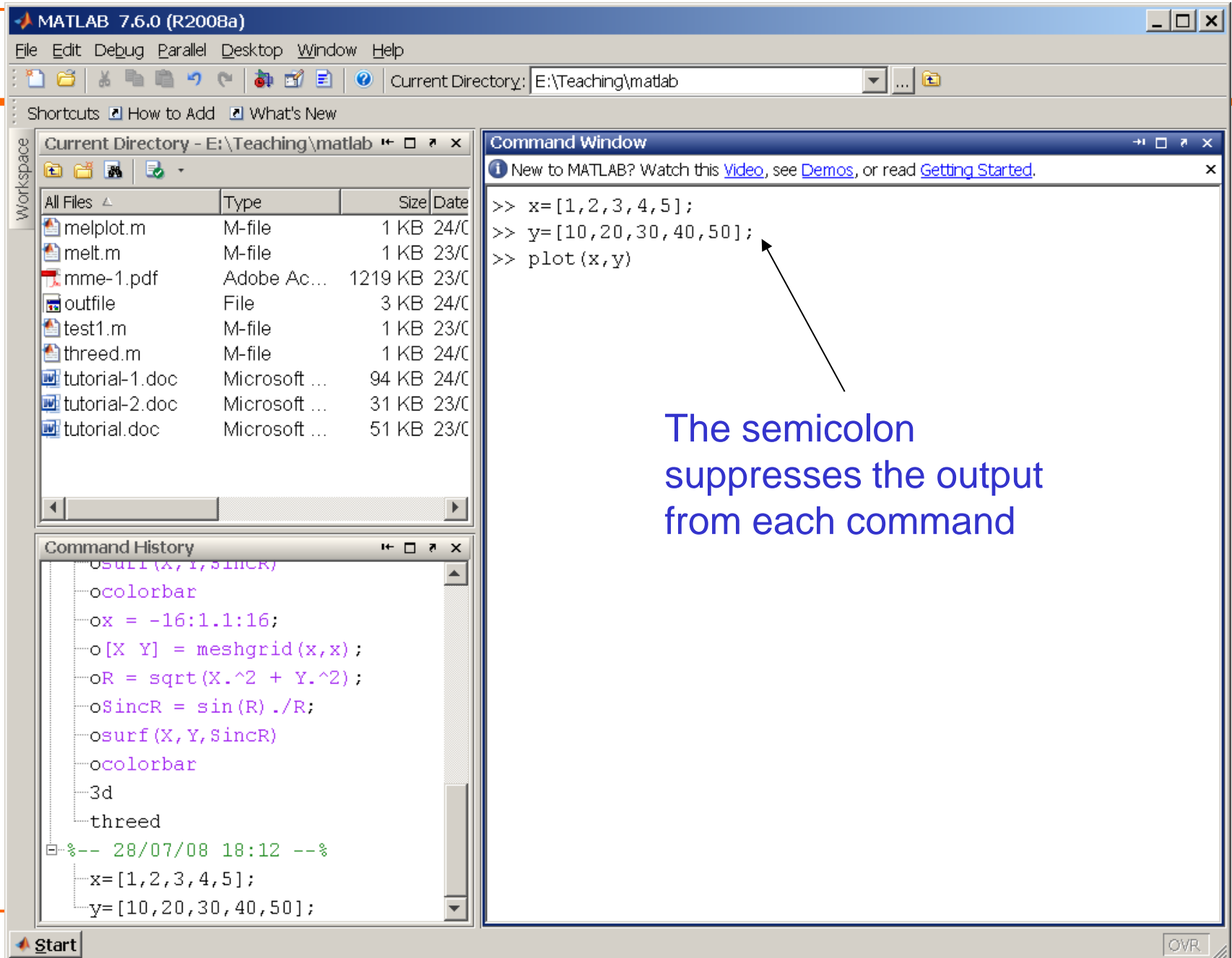
```
%-- 12/29/03 2:16 PM --%
5^2
cos(pi)
clc
5^2
cos(pi)
A=5
B=[1,2,3,4]
C = [ 1 2 3 4; 10 20 30 40; 5
whos
```

Array Editor Window (C):

	Size	Bytes	Class
A	1x1	8	double array
B	1x4	32	double array
C	3x4	96	double array
ans	1x1	8	double array

Grand total is 18 elements using 144 bytes

If you double click on any variable in the workspace window MATLAB launches a **document** window containing the **array editor**. You can edit variables in the array editor



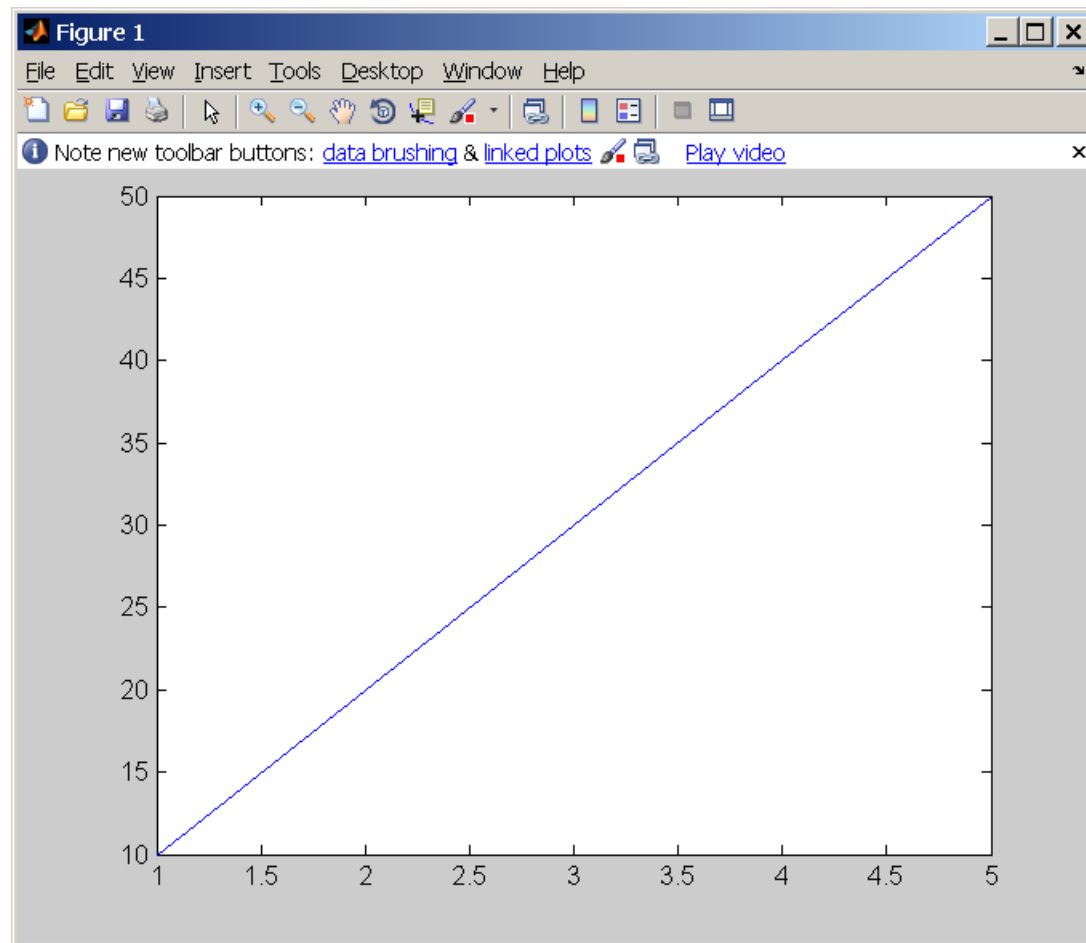


Figure Window

Editing Window

- This window allows you to type and save a series of commands without executing them
 - There are several ways to open an editing window
 - ❑ From the file menu
 - ❑ With the new file icon
-

