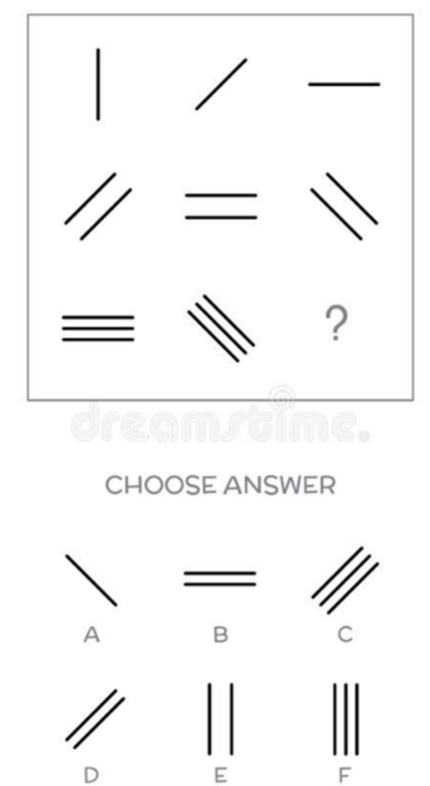
Or go to www.pollev.com/jsands601

Which image comes next?



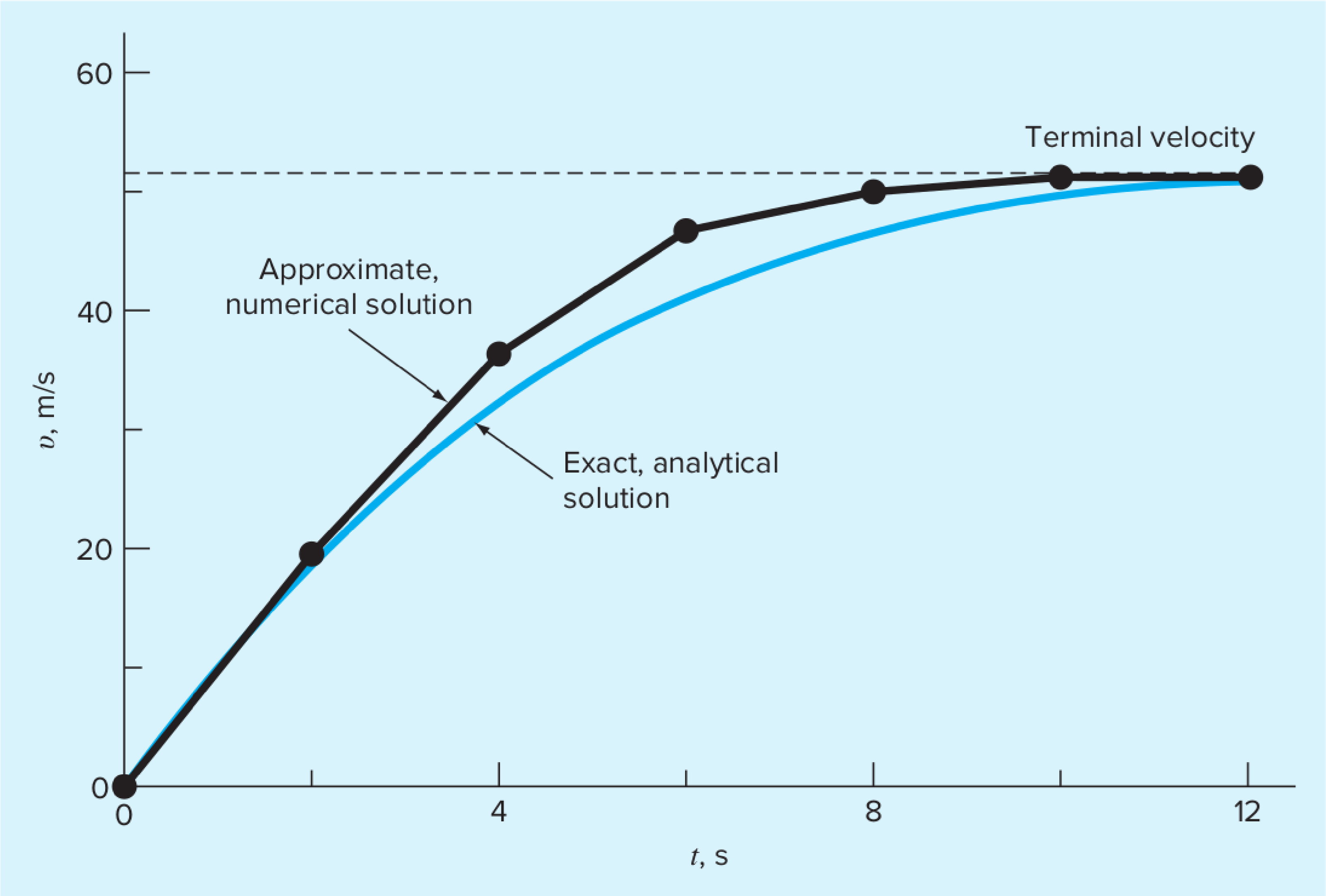
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

1.1 Numerical Approximations

- In general it is better to obtain analytic expressions (closed form solutions) to problems so that the only error introduced is through engineering tolerances.
- However as the scope of problems we try to address increases in size & complexity we find that we cannot obtain analytic solutions.
- In these cases we resort to numerical methods which approximate integrals, differential equations, help with data analysis and visualisation.
- The benefit is that we can solve a much larger range of problems than we can analytically.

- A typical trait of numerical approximation compared with an analytic solution.



1.2 Modelling & Data Analysis

- We will be using two main types of numerical methods.

Mathematical Modelling

Creating equations (often differential) that describe physical situations

Approximating the solutions to those equations

Simulating test environments

Identifying key parameters that produce the wanted change in the system behaviour

Data Analysis

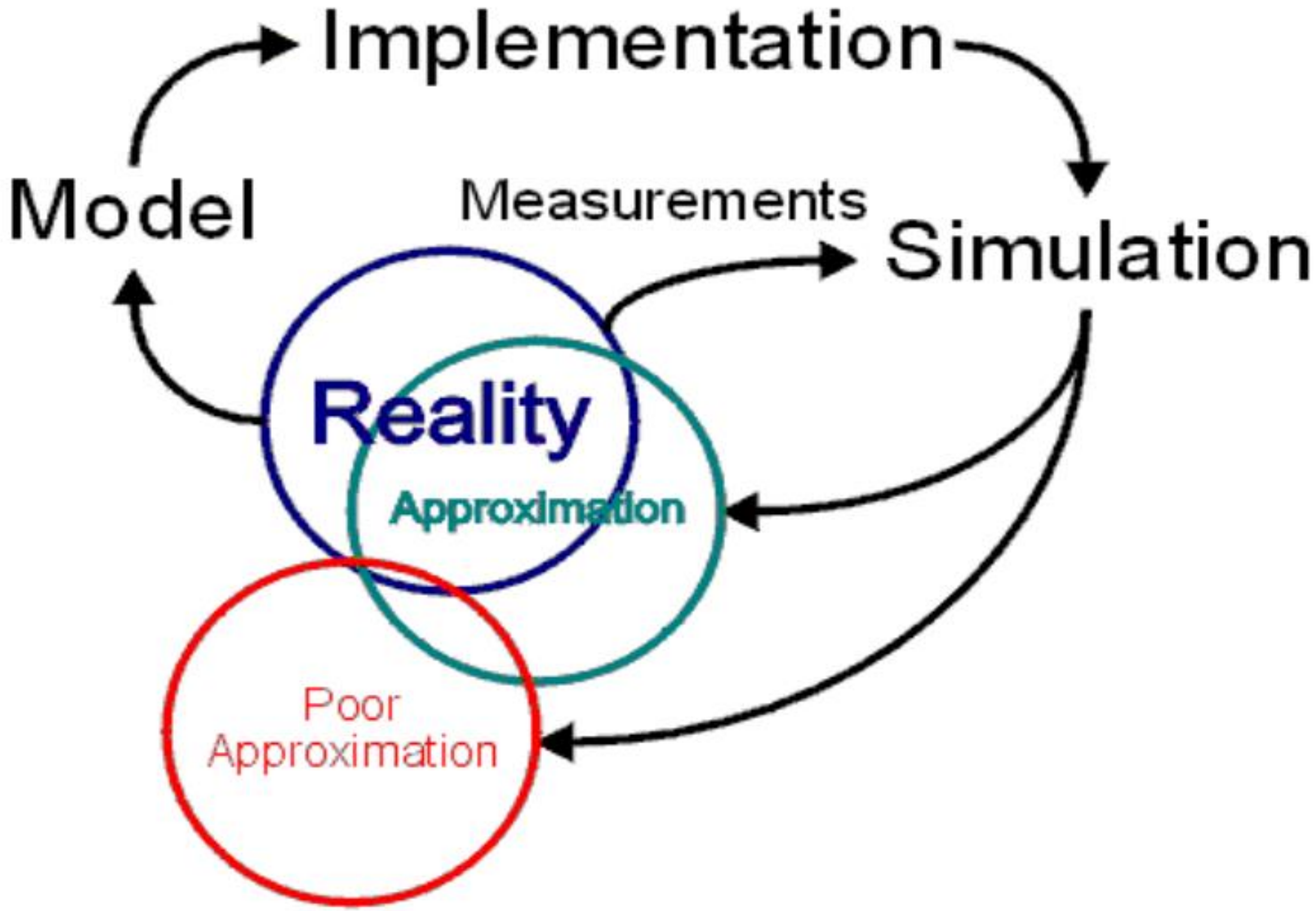
Reading and writing files that record observations from experiments

Organising and transforming the data

Plotting, fitting and calculating statistics to discover trends in the data

Creating graphics to display the results

Modelling Process



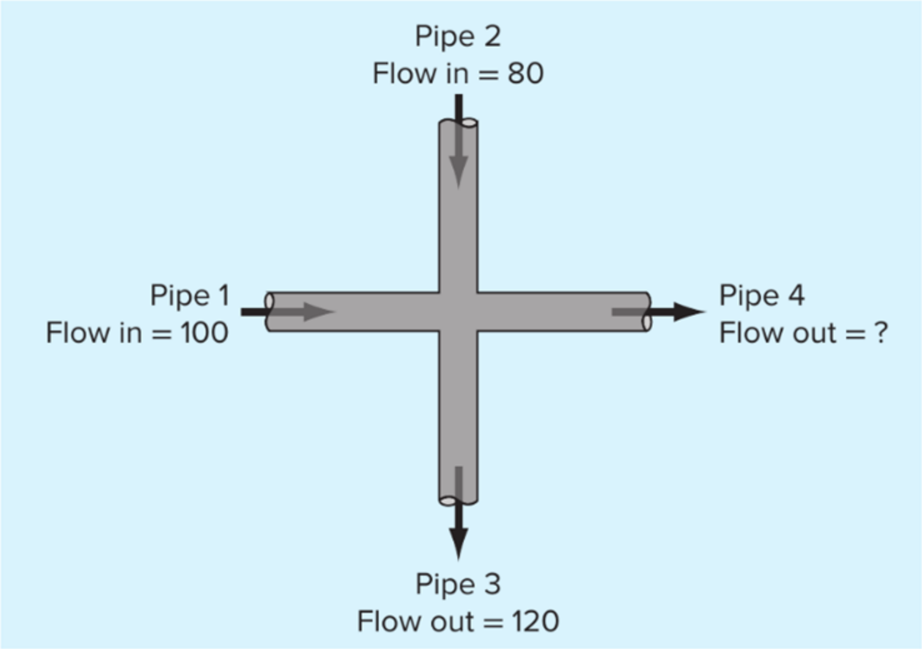
1.3 Conservation Laws

- Most models we will look at involve some form of **conservation law** which can be described as:

$$\boxed{\text{Change} = \text{increases} - \text{decreases}}$$

- If the **net change** of a system is **0** then we say it is in dynamic equilibrium (stuff in = stuff out).

What is the flow out?



20

40

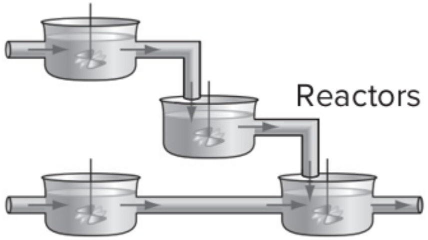
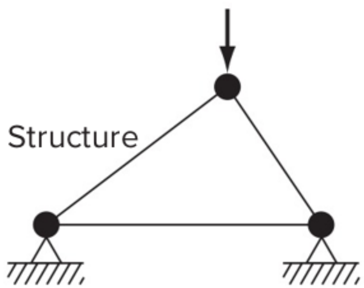
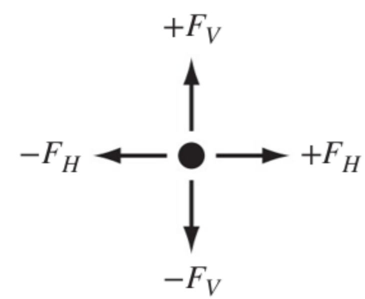
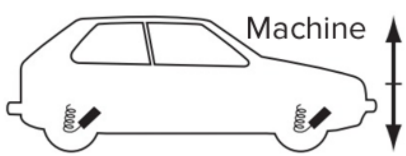
60

80

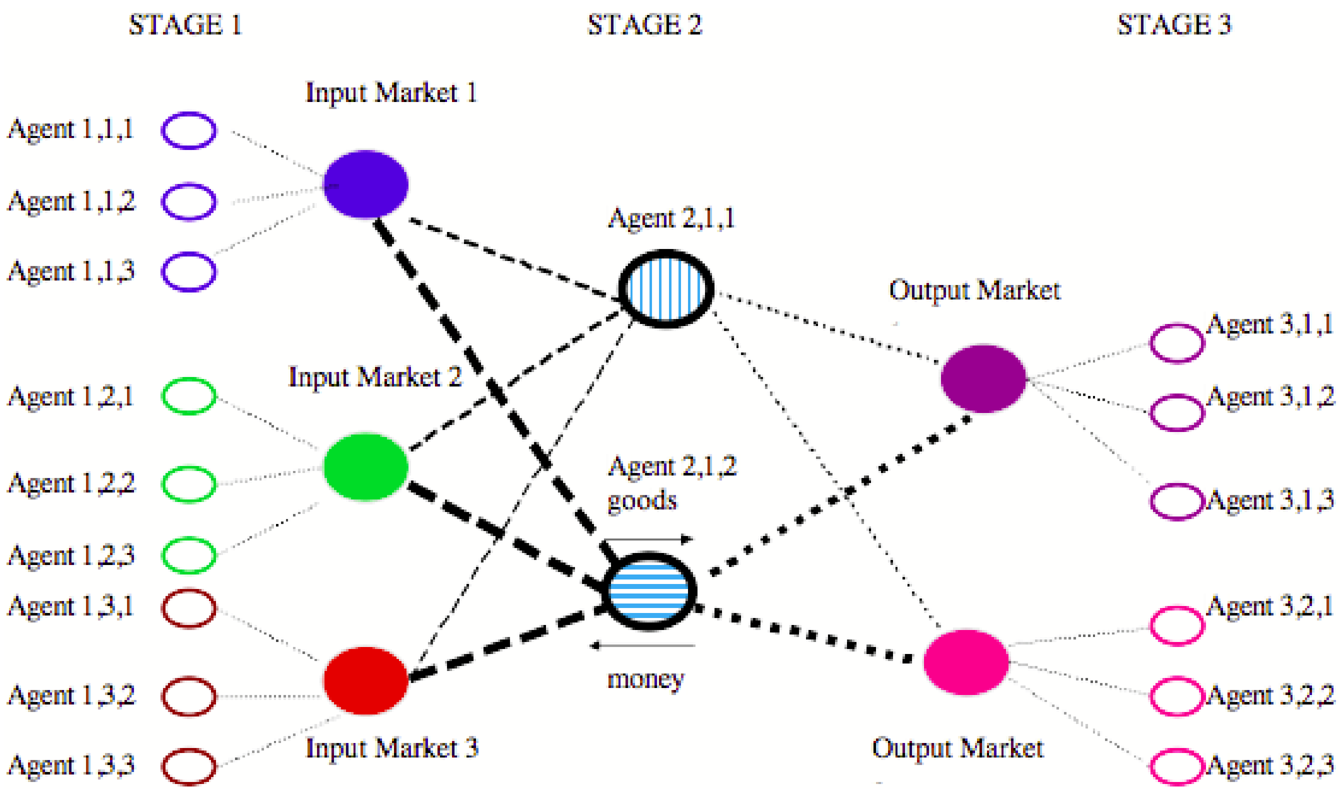
100

120

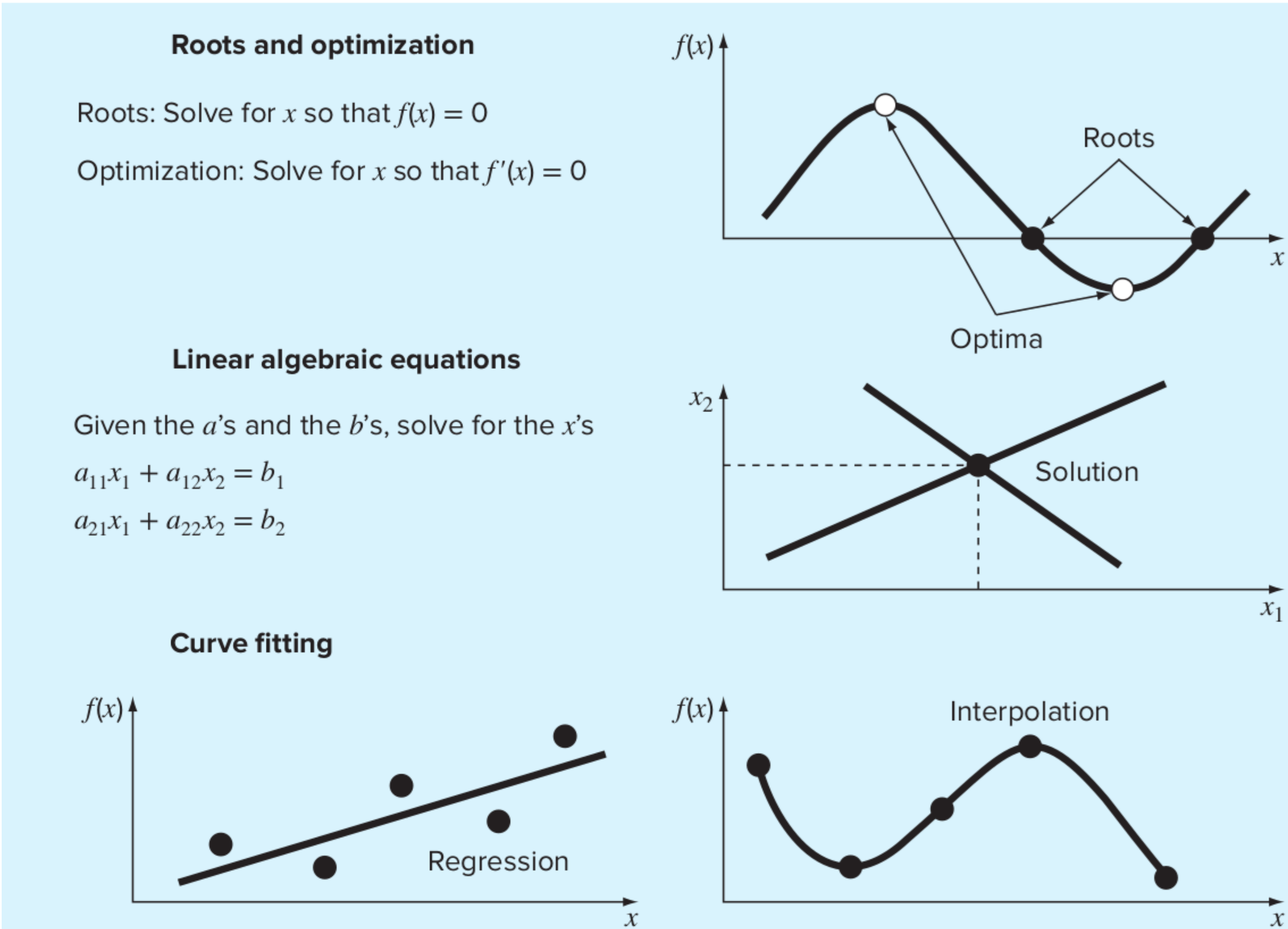
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Field	Device	Organizing Principle	Mathematical Expression
Chemical engineering		Conservation of mass	Mass balance: Input  Output Over a unit of time period Δmass = inputs − outputs
Civil engineering		Conservation of momentum	Force balance:  At each node Σ horizontal forces $(F_H) = 0$ Σ vertical forces $(F_V) = 0$
Mechanical engineering		Conservation of momentum	Force balance:  $m\dfrac{d^2x}{dt^2}$ = downward force − upward force

- Conservation of flow in supply chain management.



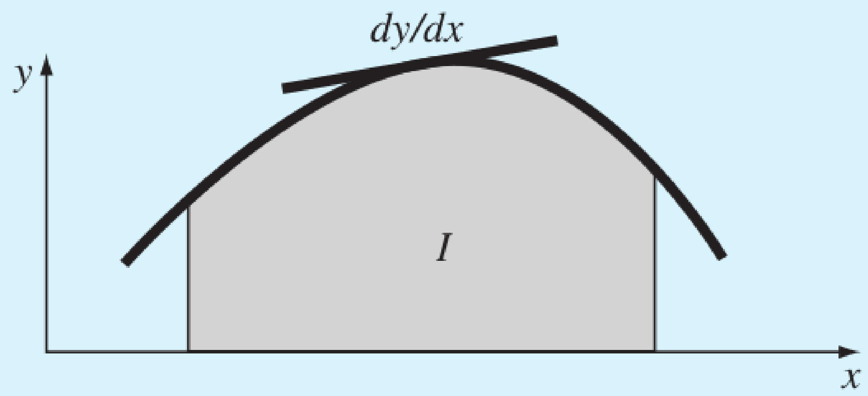
1.4 Types of Numerical Method



Integration and differentiation

Integration: Find the area under the curve

Differentiation: Find the slope of the curve



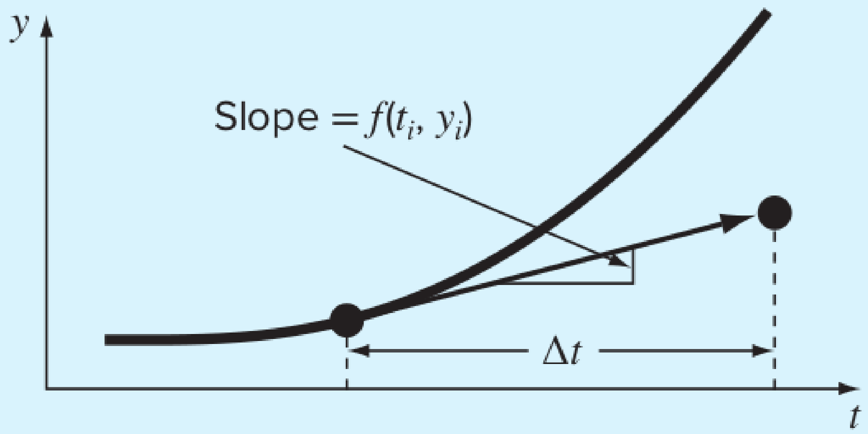
Differential equations

Given

$$\frac{dy}{dt} \approx \frac{\Delta y}{\Delta t} = f(t, y)$$

solve for y as a function of t

$$y_{i+1} = y_i + f(t_i, y_i)\Delta t$$



1.5 Implementation of Numerical Methods

- We first derive a numerical method using mathematics to allow us to solve a complex problem.

- We must then implement it somehow. In modern times computing power has grown significantly and so the methods covered here can be utilised by them.

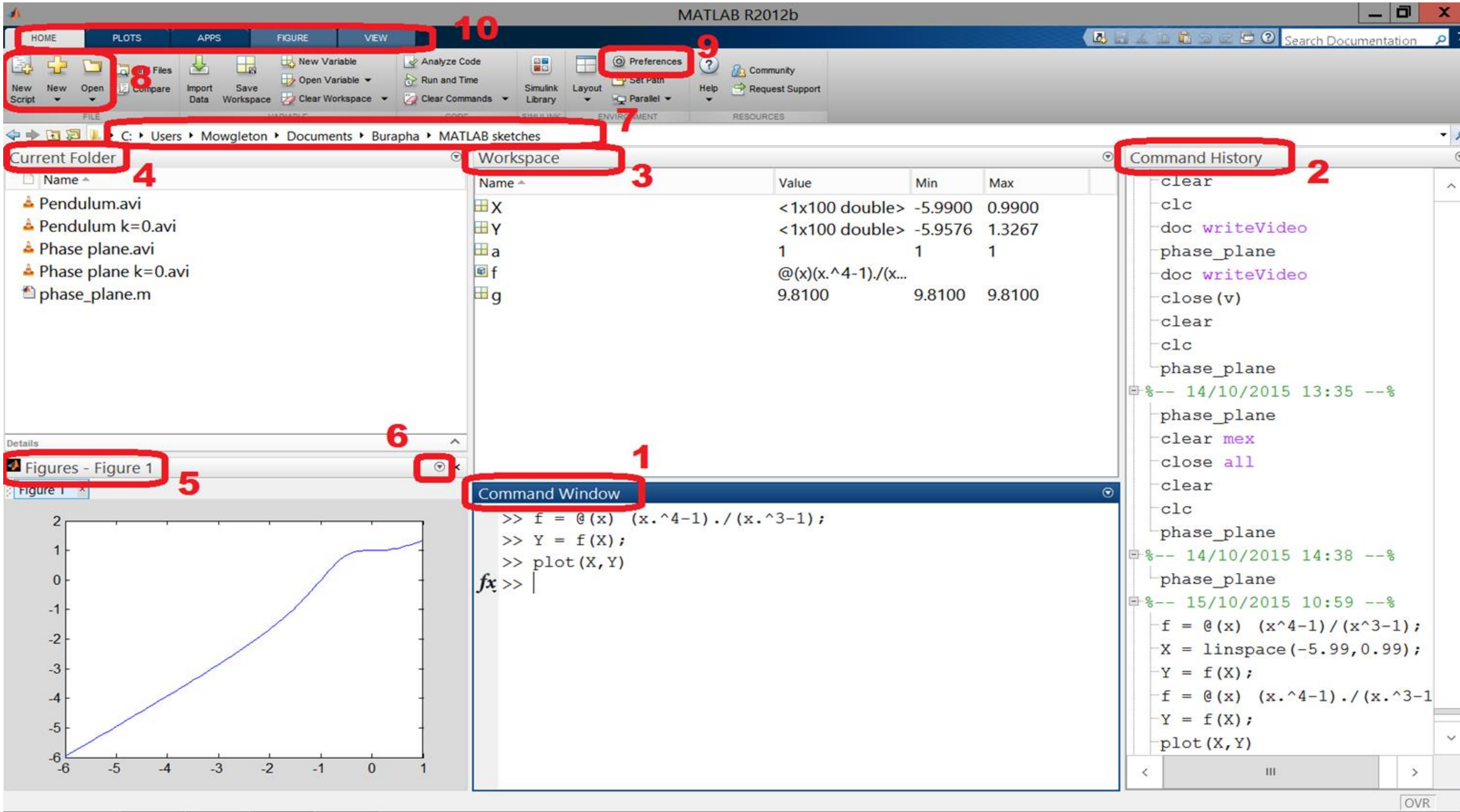
- We must write **algorithms** in a language the computer understands to carry out our orders.

1.6 Matlab Fundamentals

- Matlab is a contraction of "**Mat**rix **Lab**oratory" since its primary data structure is a matrix type.
- Matlab is both an interpreted, high level programming language (more on this later) and an interactive computation environment.
- In addition to the basic programming routines that it comes with, there are additional packages (**toolboxes**) that can be installed to enhance functionality.
- Example toolboxes are Simulink (control systems), Optimisation, Signal Processing, Neural Networks etc.

User Interface

Component	Description
1 – Command window	Interactive prompt, accepts commands, displays output
2 – Command history	Lists previously used commands
3 – Workspace	Displays variables in current session
4 – Current folder	Displays files in current working directory
5 – Figure window	Displays figures
6 – Window actions	Menu of window actions
7 – Working directory	Displays path of current working directory
8 – File shortcuts	Shortcuts to opening/creating new files
9 – Preferences	Shortcut to MATLAB preferences
10 – Ribbon tab	Tabs displaying action buttons for various components



- Your interface can be **rearranged** to suit your needs.
- You can **dock or undock** any of the windows.
- If you can't see a certain window make sure it is checked in the **layout menu** (next to preferences).

1 - Command Window

- This is the place for entering basic commands one at a time.

- You will see the command prompt:

>> ← **Enter commands after this symbol**

- Matlab will also return the output of any command here.

>> 45 – 23 ← **Press “Enter” to execute a command**

ans =

22

>> ← **Ready for next command**

- The answer to any calculation will be stored as a variable called "ans" in the **workspace**.

- You can suppress the output by putting a semicolon at the end of a command.

```
>> x = 14 + 29;
>>
```

← **Semicolon**

- The answer is not displayed but it is still calculated.

- The variable, *x*, now appears in the workspace.

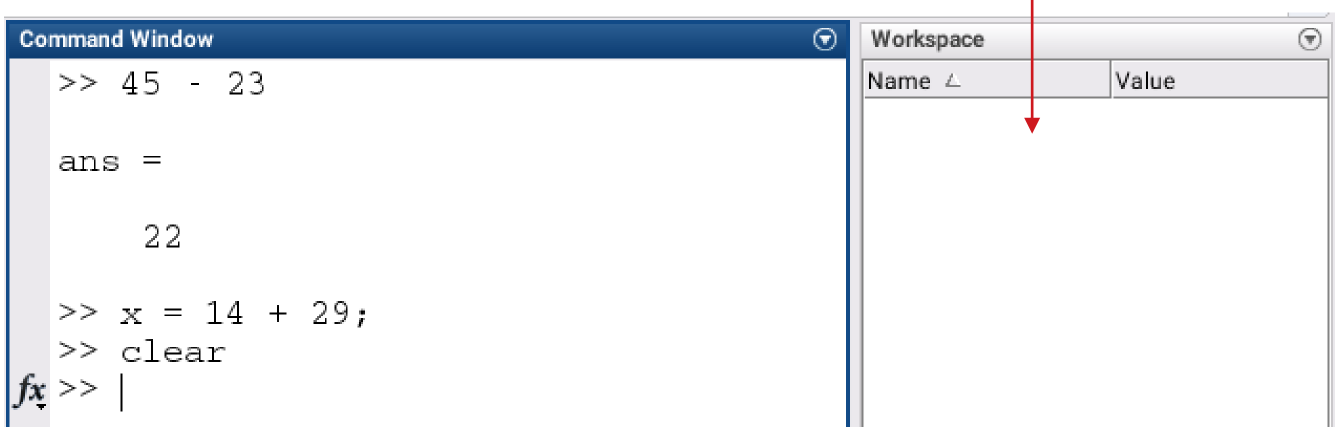
- We can delete all variables.

  >> clear

- Or just clear the specified variable.

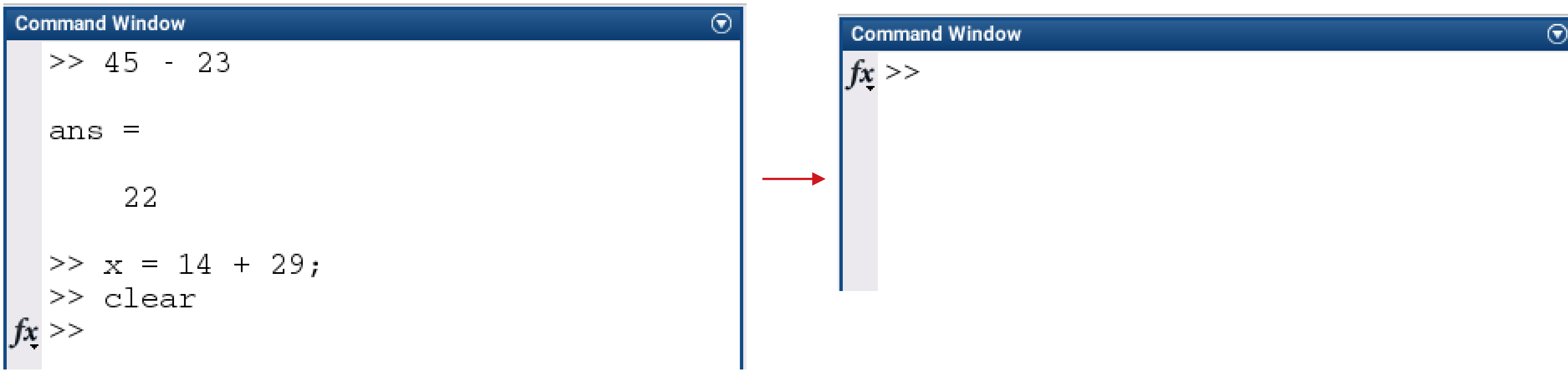
Variables have been cleared

- >> clear **x**



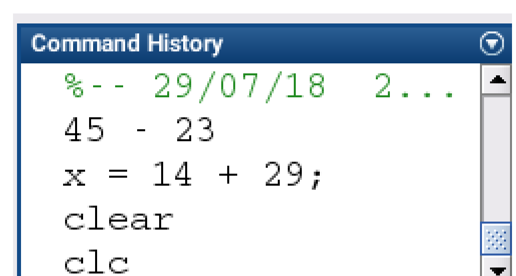
- If the command window has too many commands then we can **clear the command window**.

>> clc



- Don’t worry though all the commands are stored in the **command history** for easy access later.

- You can re-enter old commands by double clicking or by drag and drop.



- You can scroll through previously entered commands by pressing the "**up**" cursor key.

- Pressing "**Esc**" clears the command line.

- You can enter multiple commands on the same line separated by a comma **","** or semicolon **";"** if you want to suppress the output.

```
>> a = 4,A = 6;x = 1;

a =

     4
```

- Note that variable names are **case-sensitive** **(**$a \neq A$**)**.

- Re-assigning a variable will change its value, no need to clear every time.

- Complex numbers use the imaginary unit which is pre-assigned to “**i**”.

```
>> x = 2+i*4

x =
   2.0000 + 4.0000i
```

- There are also special mathematical numbers available such as “**pi**”.

```
>> pi

ans =
    3.1416
```

- You can change the format of the output.

```
>> format long
```

```
>> pi

ans =
   3.14159265358979
```

- Options for the format command:

type	**Result**	**Example**
`short`	Scaled fixed-point format with 5 digits	3.1416
`long`	Scaled fixed-point format with 15 digits for double and 7 digits for single	3.14159265358979
`short e`	Floating-point format with 5 digits	3.1416e+000
`long e`	Floating-point format with 15 digits for double and 7 digits for single	3.141592653589793e+000
`short g`	Best of fixed- or floating-point format with 5 digits	3.1416
`long g`	Best of fixed- or floating-point format with 15 digits for double and 7 digits for single	3.14159265358979
`short eng`	Engineering format with at least 5 digits and a power that is a multiple of 3	3.1416e+000
`long eng`	Engineering format with exactly 16 significant digits and a power that is a multiple of 3	3.14159265358979e+000
`bank`	Fixed dollars and cents	3.14

- The better you get with Matlab the more you will search and understand the **documentation**.

- To find out more about a Matlab function, type "**doc**" or "**help**" followed by the function, or simply "doc" to open the documentation browser.

Command Window

```
>> help sin
 sin    Sine of argument in radians.
    sin(X) is the sine of the elements

    See also asin, sind.

    Reference page for sin
    Other functions named sin
```

Command Window

```
>> doc sin
>>
```

sin

Sine of argument in radians

collapse all in page

Syntax

```
Y = sin(X)
```

Description

`Y = sin(X)` returns the sine of the elements of X. The `sin` function operates element-wise on arrays. The function accepts both real and complex inputs. For real values of X in the interval [-Inf, Inf], `sin` returns real values in the interval [-1 ,1]. For complex values of X, `sin` returns complex values. All angles are in radians.

example

Examples

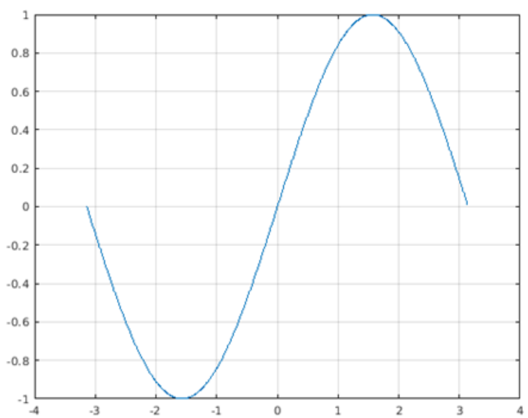
collapse all

Plot Sine Function

Plot the sine function over the domain $-\pi \le x \le \pi$.

Open Live Script

```
x = -pi:0.01:pi;
plot(x,sin(x)), grid on
```



1.7 Arrays, Vectors & Matrices

- Arrays are a collection of values with indexed locations.
- **1D arrays** are known as **vectors**, **2D arrays** are known as **matrices**, higher dimensions are generally *n*-dimensional arrays (multidimensional arrays).
- Use **square brackets** to enter vectors with entries separated by **spaces** or **commas**.

```
>> a = [1 2 3 4 5]

a =
     1     2     3     4     5
```

```
>> a = [1, 2, 3, 4, 5]

a =
   1   2   3   4   5
```

- **Semicolon's** can be used to **end lines** in arrays so a column matrix can be written as:

```
>> b = [2;4;6;8;10]
```

- There is also a **transpose** operator " ' ".

```
>> b = [2 4 6 8 10]'
```

```
b =

     2
     4
     6
     8
    10
```

- Both produce the same result.

- We can create a matrix:

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```

- Matlab can **concatenate** (join together) arrays to make new arrays.

```
>> A = [ [1 2 3]' [4 5 6]' [7 8 9]' ]

A =

     1     4     7
     2     5     8
     3     6     9
```

```
>> A = [ [1 2 3] [4 5 6] [7 8 9] ]

A =

     1     2     3     4     5     6     7     8     9
```

- Long lines can be split using an **ellipsis**.

```
>> y = [1 2 3 ...
4 5 6]

y =

     1     2     3     4     5     6
```

- Be careful that the **dimensions** of your arrays **match** up otherwise Matlab will give you an **error**.

```
>> A = [ [1 2 3] [4 5 6]' [7 8 9] ]
Dimensions of arrays being concatenated are not consistent.
```

- The "**who**" command tells you what variables you have stored and the "**whos**" command gives more detail.

```
>> who

Your variables are:
A      a      ans   b      x
```

```
>> whos

  Name      Size                    Bytes  Class

  A         3x3                        72  double array
  a         1x5                        40  double array
  ans       1x1                         8  double array
  b         5x1                        40  double array
  x         1x1                        16  double array (complex)

  Grand total is 21 elements using 176 bytes
```

- We access the array value we want using its **index**.

- The colon operator “ **:** ” signifies a full line or column.

```
b =
     2
     4
     6
     8
    10
```

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =
     1     2     3
     4     5     6
     7     8     9
```

```
>> b(4)

ans =
     8
```

```
>> b(2:4)

ans =

     4
     6
     8
```

```
>> A(2,3)

ans =
     6
```

```
>> A(:,3)

ans =

     3
     6
     9
```

```
>> A(2,:)

ans =

     4     5     6
```

- The last entry in a vector can be accessed using the keyword "**end**".

```
>> b
b =
        2       4       6       8      10
>> b(end)
ans =
    10
```

- You can **add an entry**:

```
>> b(end+1) = 12
b =
     2     4     6     8    10    12
```

- You can also remove entries with the **empty set** " **[]** ".

```
>> b(3) = []
b =
     2     4     8    10    12
```

Creating Vectors with a Colon

- Specify **start** and **end** points:

```
>> t = 1:5

t =
     1     2     3     4     5
```

- You can specify **step size** too:

```
>> t = 1:0.5:3

t =
    1.0000    1.5000    2.0000    2.5000    3.0000

>> t = 10:-1:5

t =
    10     9     8     7     6     5
```

Creating Vectors Using Linspace

- The **linspace** function takes start, end and number of elements:

```
>> linspace(0,1,6)

ans =
         0    0.2000    0.4000    0.6000    0.8000    1.0000
```

- If the last input is not specified the default is 100.

How to get "f"?

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

A(2,3)

A(3,2)

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How to get the 3rd row ("g h i")?

$$A = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

A(3,:)

A(:,3)

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

How to get "j k l"?

$$B = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}$$

B(3,:)

B(2:4,3)

B(3,2:end)

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Character Strings

- **Characters** are symbols such as letters.
- A **string** is a sequence of characters.
- In Matlab character arrays are enclosed by **single quotation marks**.

```
>> f = 'Miles ';
>> s = 'Davis';
```

- We can **concatenate** character arrays too.

```
>> x = [f s]

x =
Miles Davis
```

- Useful character array functions:

Function	Description
`n=length(s)`	Number of characters, n, in a string, s.
`b=strcmp(s1,s2)`	Compares two strings, s1 and s2; if equal returns true (`b = 1`). If not equal, returns false (`b = 0`).
`n=str2num(s)`	Converts a string, s, to a number, n.
`s=num2str(n)`	Converts a number, n, to a string, s.
`s2=strrep(s1,c1,c2)`	Replaces characters in a string with different characters.
`i=strfind(s1,s2)`	Returns the starting indices of any occurrences of the string s2 in the string s1.
`S=upper(s)`	Converts a string to upper case.
`s=lower(S)`	Converts a string to lower case.

EXAMPLE 3

```
>> x1 = ‘Canada’; x2 = ‘Mexico’; x3 = ‘USA’; x4 = ‘2010’; x5 = ‘810’

>> strcmp(x1,x2)

ans =

0
```

0 means false

```
>> strcmp(x2,’Mexico’)

ans =

1
```

1 means true

1.8 Mathematical Operations

^	Exponentiation
−	Negation
* /	Multiplication and division
\	Left division[2]
+ −	Addition and subtraction

Raise to the power

Used in linear algebra

- Follow BIDMAS rules.

```
>> y = -4 ^ 2

y =
   -16
```

```
>> y = (-4) ^ 2

y =
    16
```

Vector-Matrix Multiplication

- The default multiplication in Matlab is matrix multiplication.

$$\begin{pmatrix} \boxed{1 \quad 2} \\ 3 \quad 4 \end{pmatrix} \begin{pmatrix} \boxed{\begin{matrix} 1 \\ 3 \end{matrix}} & \begin{matrix} 2 \\ 4 \end{matrix} \end{pmatrix} = \begin{pmatrix} \boxed{1 \times 1 + 3 \times 2} & 2 \times 1 + 4 \times 2 \\ 1 \times 3 + 3 \times 4 & 2 \times 3 + 4 \times 4 \end{pmatrix}$$

$$A = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

```
>> a = [1 2 3];
```

$>> b = [4\ 5]'$ $\longrightarrow$ >> A * b

ans =
78
170

```
>> A * a
Error using  *
Incorrect dimensions for matrix multiplication.
```

Element-wise Multiplication

- We can choose to multiply or divide element by element using the dot operator " .* " or " ./ ".

```
>> x = [1 2 3];
>> y = [2 4 6];
>> x .* y

ans =

     2     8    18
```

```
>> y ./ x

ans =

     2     2     2
```

```
>> x*y
Error using  *
Incorrect dimensions for matrix multiplication.
```

1.9 Common Functions

- Matlab has many built-in functions that make our life easier.

- If you want to find out if Matlab already has a function that can do what you want you can search the documentation or online.

EXAMPLE 4 The **length** and **size** functions are used frequently.

```
>> x
x =
     1     2     3
>> length(x)
ans =
     3
```

```
>> A
A =
     1     2     3
     4     5     6
     7     8     9
>> size(A)
ans =
     3     3
```

Function	Description
abs()	Absolute value of a number
max(), min()	Maximum and minimum of array
length()	Length of vector
size()	Size of array
numel()	Number of elements in array
sort()	Sorts elements of vector
sum()	Sums elements of vector
round()	Rounds to nearest integer
floor()	Rounds down to nearest integer
ceil()	Rounds up to nearest integer
real(), imag()	Real and imaginary parts of number
mod()	Modulo operation for two numbers
rand()	Create random array of given dimensions
plot()	Plots vectors of the same length
dot(), cross()	Dot and cross products of vectors