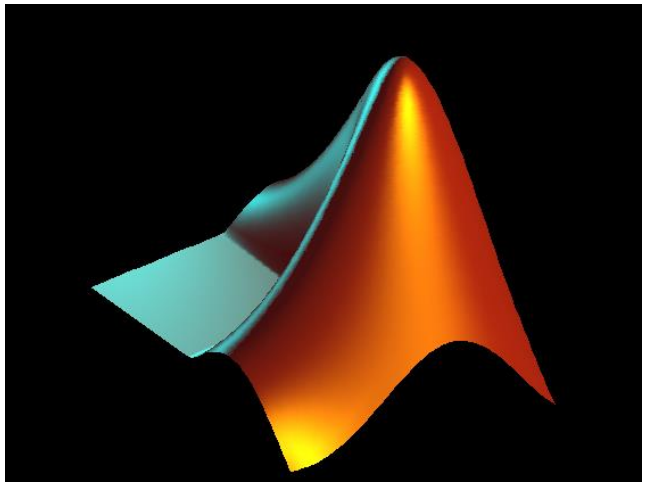
Computational Mathematics with MATLAB
Topic 2



Getting started with MATLAB

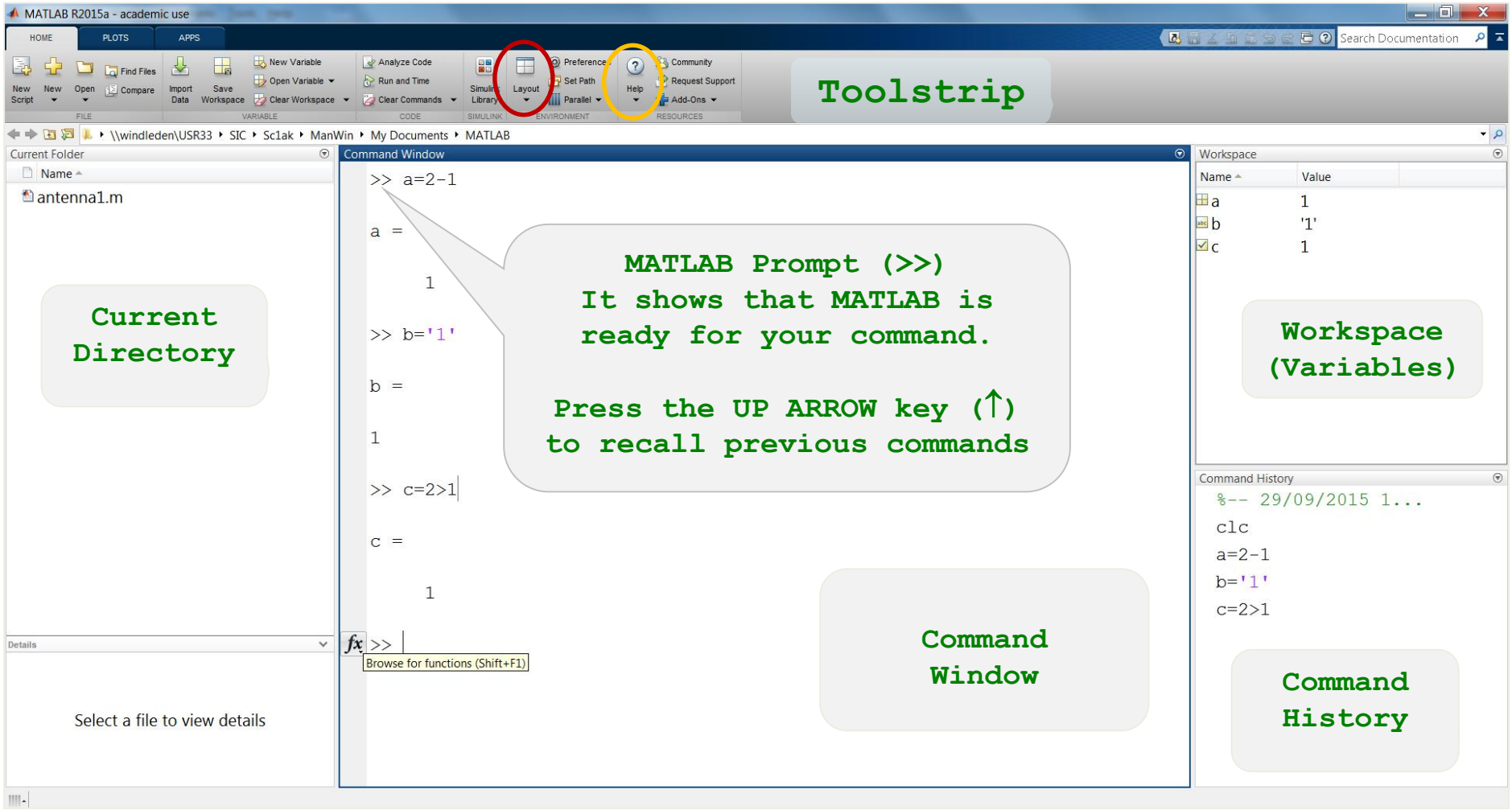
Variables, functions, statements and operators

Outline

- MATLAB user interface
- Essential commands
- Evaluating mathematical expressions
- Function input and output
- How to name MATLAB variables
- Arrays and basic data types, creating arrays
- Numeric arrays
- String arrays
- MATLAB operators
- Logical arrays
- Command Window plotting

MATLAB User Interface

Changing the appearance: go to **Layout** in the **Toolstrip**



A Few Basic Commands & Special Characters

- **clc** clears the *Command Window,* **clf** *clears the Figure Window*
- **clear or clear all** removes all variables from the Workspace
- **;** (semicolon): suppresses the output in the *Command Window* (other uses of the semicolon will discussed later)
- **%** starts a comment (text which is ignored by the compiler)
- **exist** *varname* checks if *varname* is already in use

Getting Help: type **help** *command* or click on *Help* in the *Toolstrip.*

The fastest way to find the answer to a (simple) problem is to type the keywords in a search engine (Google, etc). Reading MATLAB forums and watching Youtube video tutorials is another good way to pick up useful tips.

Using MATLAB as a Calculator

Command Window

```
>> sin(2)+3*cos(2)

ans =

   -0.3391

>> ans

ans =

   -0.3391

>> x=sin(2)+3*cos(2)

x =

   -0.3391

>> x=sind(2)+3*cosd(2)

x =

    3.0331
```

This command calculates the value of the **expression** *sin*(2) + 3*cos*(2) where the **arguments** are in radians

When no **output variable** is specified, MATLAB will create variable **ans** to store the result

Displaying the value currently **assigned to ans**

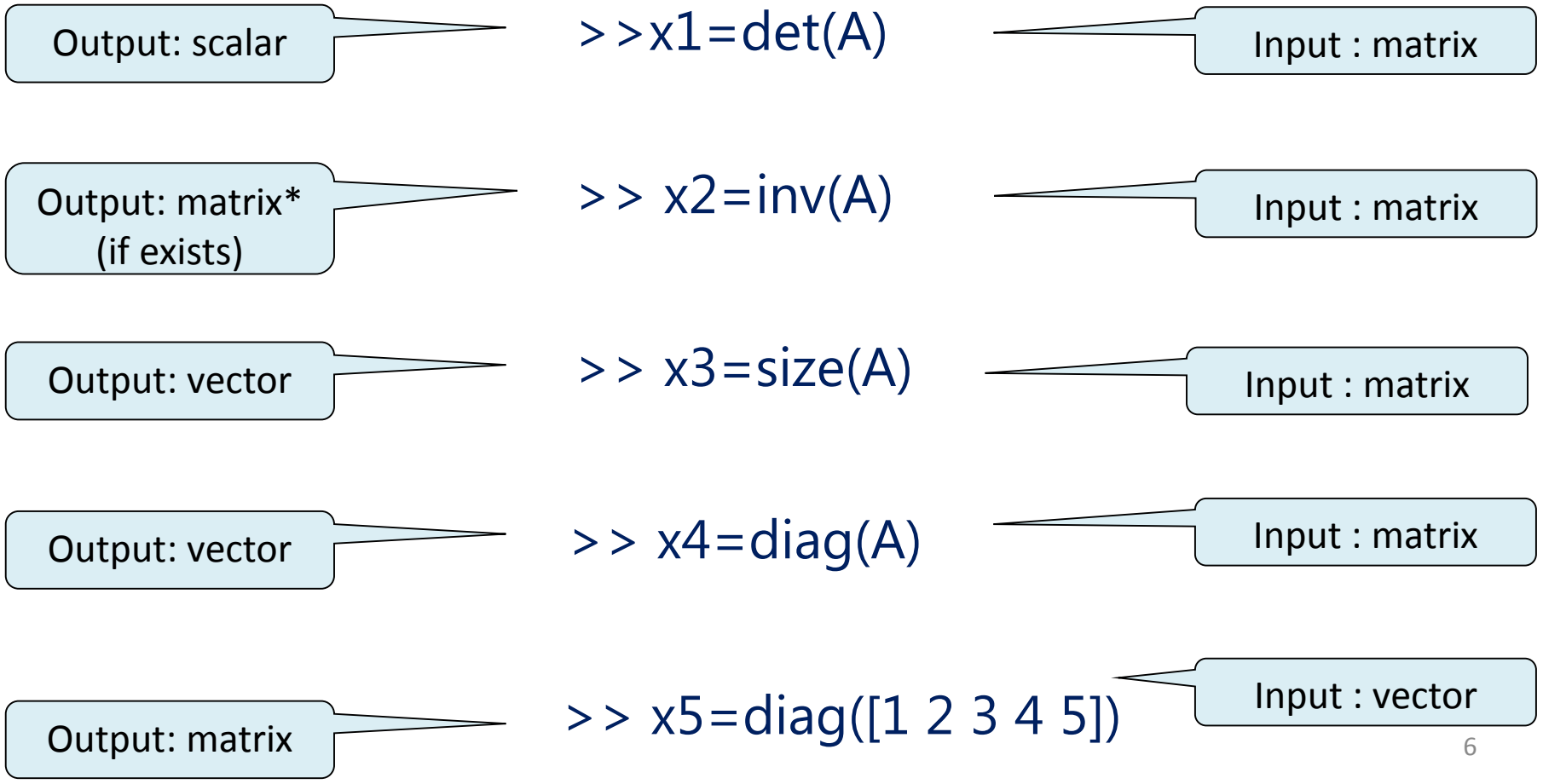
The **definition** of variable x.

Here variable x is **overwritten** with *sind*(2) + 3*cosd*(2) (arguments are now in degrees)

MATLAB Functions – Input and Output

MATLAB functions perform certain tasks, typically by taking some input and returning some output. Functions are called by name.

Type `A=randi(20,3)` and try the following commands.



Examples of Functions with Multiple Inputs

Command Window

```
>>
>> u=[3 -5 4]; v=[2 2 1]; % define 2 3D vectors
>> dot(u,v)     % calculate the scalar (dot) product

ans =

     0

>> w=cross(u,v)   % calculate the vector product

w =

   -13     5    16

>> dot(w,u)       % check if w is perpendicular to u

ans =

     0
```

- The argument (input) of a function must be enclosed in round brackets ()
- Input variables must be separated by commas

How to Name MATLAB Variables

A MATLAB variable has a name, an associated storage location and some stored content (often called value).

A **valid** MATLAB variable or file name

- starts with a letter
- may contain letters, numbers & the **underscore** (_) character
- is between 1-63 characters long
- cannot be a **keyword**, e.g. **if, for, end, read, write**

MATLAB variable/file names are **CASE SENSITIVE**

e.g. `test`, `Test` & `TEST` are all different variables

(it is best to use lowercase letters to avoid confusion)

So, `1test` or `test-1` are NOT valid variable names

`test1` or `test_1` are OK

Do not name variables after built-in functions!

(avoid using **sin, sum, inv, plot** etc.)

Arrays in MATLAB

Arrays are the fundamental data structure in MATLAB.

An **array** is an ordered collection of elements.
Each element of an array can be accessed by a unique **reference** (using indices or subscripts).

Basic data types (all elements are of the same ***base type***):

- **numeric arrays**
- **character arrays**
- **logical arrays**

Examples of advanced data types: **cell arrays** and **structures** (new users should first learn to work with basic types)

Numeric Arrays

- [1 x 1] arrays or **scalars** (numbers), e.g.

2.14

- [1 x m] arrays or **row vectors**, e.g.

[1 x 8]

2	3	4	1	3	6	7.3	8

- [n x 1] arrays or **column vectors**, e.g.

5
6.3
45
11.45

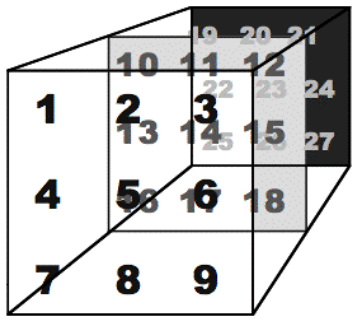
[4 x 1]

- [n x m] arrays or **matrices**, e.g.

4	45	23	-1	89	0
-2	2.3	67	1	65	1.1

[2 x 6]

- Multidimensional arrays, e.g.



[3 x 3 x 3]

Creating Row/Column Vectors

The simplest way to create a vector is to enter the elements 1-by-1

```
>> b = [1 2 3]
```

Elements must be enclosed in **rectangular brackets []**

Elements in a **row vector** must be separated by **commas or spaces**:

```
>> b = [1, 2, 3]
>> b = [1 2 3]
```

Elements in a **column vector** must be separated by **semicolons**:

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

The **transpose operator (')** switches the rows and columns,
so vectors `[1; 2; 3]'` and `[1 2 3]` are the same.

```
>> []
```

← **empty array**

Creating Matrices

Elements in the same row must be separated by spaces/commas and rows must be separated by semicolons.

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

The same result can be obtained by entering

```
>> d = [b; b; b]
```

Matrices can also be generated using built-in matrices (functions). For example, `randi([10,50],3,5)` creates a 3 x 5 matrix filled with integers chosen randomly from the interval [10, 50].

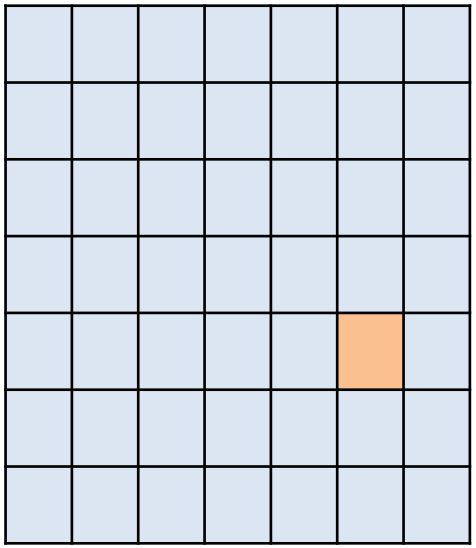
(See also utility matrices at the end)

Indexing matrix/vector elements:

```
matrixname(row, column)
```

Referencing/Indexing Array Elements

Matrix A
[n x m] array

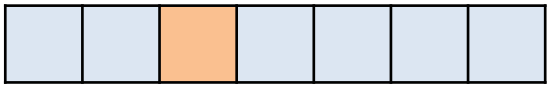


`A(5,6)`

Column Vector v
[n x 1] array

`v(7) or v(7,1)`

Row Vector u
[1 x m] array



`u(3) or u(1,3)`

We will look at ways of indexing subarrays (parts/blocks of an array) later.

Character Arrays (Strings)

A **simple string** is a sequence of characters enclosed in single quotation marks (').

`'Hello'` (string/row vector of length 5, i.e. 1 x 5 array)

`'Good evening!'` (1 x 13 – the *space* counts too!)

`'1234'` (1 x 4)

To include a single quote in a string, use 2 single quotes in a row:

Typing `'I''m lazy!'` creates the string `I'm lazy!`

(1 x 9)

Examples of Character Arrays

Command Window

```
>> v=['a'; 'b'; 'c']

v =

a
b
c

>> M=['a', 'b', 'c'; 'p', 'q', 'r'; 'x', 'y', 'z';]

M =

abc
pqr
xyz

>> M(2,3)

ans =

r
```

v is a 3 x 1 (column) vector of characters

M is a 3 x 3 character matrix

selecting the element in the 2nd row & 3rd column

Operators in MATLAB

Operators are special characters which perform certain actions on their **operands** to return a result. Operators generally act like functions but are used in a different format.

Operator types

- **Arithmetic**
- **Relational**
- **Logical**
- **Assignment operator**
- **Colon operator**

https://uk.mathworks.com/help/matlab/matlab_prog/matlab-operators-and-special-characters.html

Operators in MATLAB

- The **assignment operator =** is used to **define** (declare) variables.

```
x=2
```
(let x be equal to 2)

- **Arithmetic operators** act on numerical variables

```
x+2
```

Several arithmetic operators come in two versions: matrix and elementwise (more on this later)

- **Relational operators** are used to **compare** variables.

```
x==2
```
(is x equal to 2?)

- **Logical operators** are used to compare *logical* variables

(see next section)

Logical Data Type

The logical data type represents **TRUE or FALSE states** using the numbers 1 and 0, respectively. Logical values can be created using **relational operators** and certain MATLAB functions.

Logical values (scalars):

1 – if the proposition/statement is **TRUE**

0 – if the proposition/statement is **FALSE**

Relational Operators		
< less than	> greater than	== equal to
<= less than or equal to	>= greater than or equal to	~= not equal to

Examples of Logical Variables

MATLAB Input	Output (ans)
`>> 2 < 6`	1
`>> 2 <= 6`	1
`>> 2 > 6`	0
`>> 2 >= 6`	0
`>> 2 == 6`	0
`>> 2 ~= 6`	1
`>> isprime(4)`	0
`>> isreal(3)`	1
`>> any([0 0 2 0 0])`	1

Examples of Logical Arrays

Command Window

```
>>
>> x=[ 5 2 56 2]; y=[1 3 12 4]; x>y

ans =

     1     0     1     0

>> M=randi(20,3)

M =

     2     7     9
    17    20     8
    14     1    16

>> isprime(M)

ans =

     1     1     0
     1     0     0
     0     0     0

>>
```

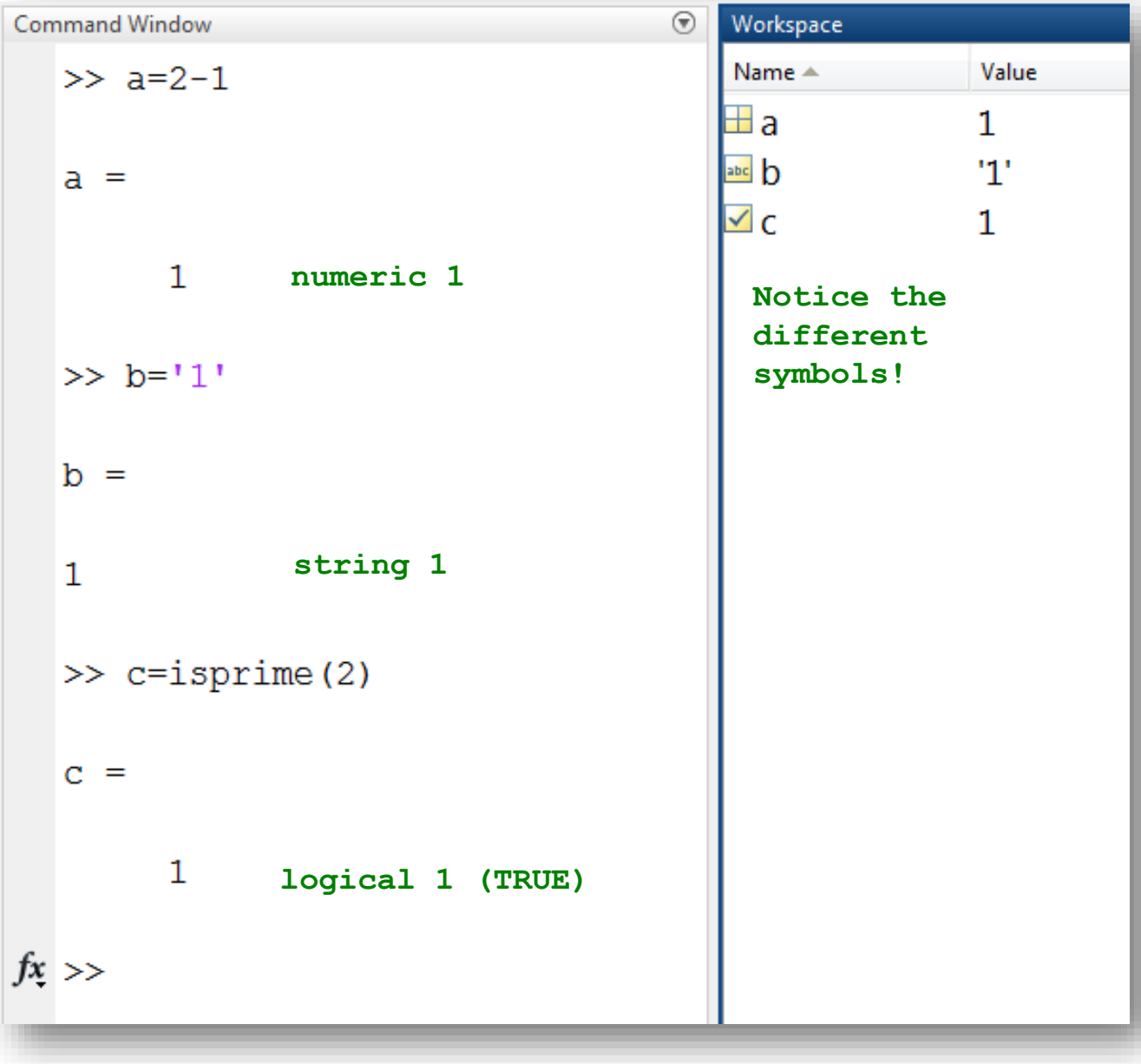
Output: 1 x 4 logical vector
the relationship x>y holds
in the 1st and 3rd positions

M is a 3 x 3 matrix filled with integers chosen randomly from [1,20]

M(1,1), M(1,2) and M((2,1) are primes

Output: 3 x 3 logical matrix
the locations of prime numbers in M are indicated by the 1s

Numeric, String and Logical 1



Arithmetic Operators

There are two types of arithmetic operations in MATLAB. **Matrix operations** follow the rules of linear algebra while **elementwise or array operations** are carried out element-by-element.

Matrix operation	Equivalent Elementwise operation
`+` addition	`N/A (.+ would be same as +)`
`-` subtraction	`N/A (.- would be same as -)`
`*` multiplication	`.*`
`^` exponentiation	`.^`
`/` (not division, see link)	`./ elementwise division`

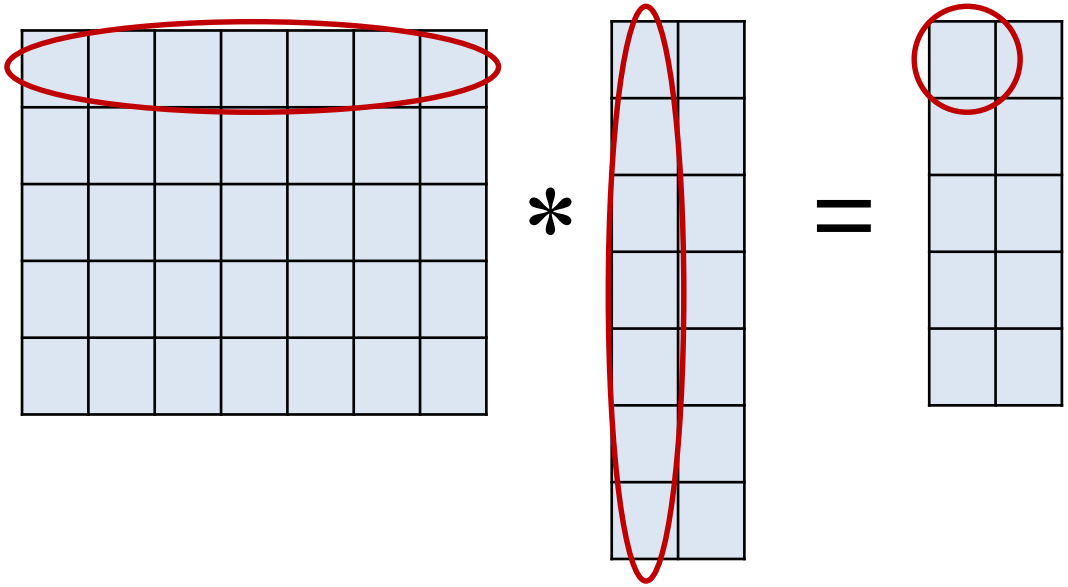
https://uk.mathworks.com/help/matlab/matlab_prog/matlab-operators-and-special-characters.html
https://uk.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html

Use A=randi(10, 3), B=randi(10, 3), and try a few expressions, e.g.

```
A*B, A.*B, A^3, A.^3, A/B, A./B, A.^B, 2^A
```

Matrix vs Elementwise Multiplication

A * B = C (Matrix Multiplication)



[5 x 7] * [7 x 2] → [5 x 2]

Matrices A and B can be multiplied together if

number of columns in A = number of rows in B

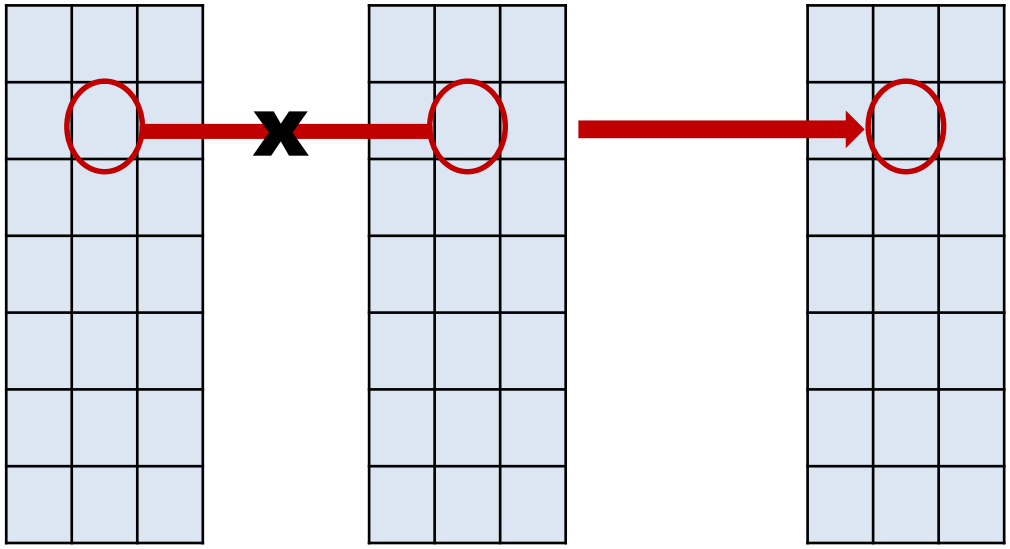
[m x n] * [p x q] → [m x q] if n = p

$$C_{ij} = \sum_{k=1}^{n=p} A_{ik} B_{kj}$$

Matrix vs Elementwise Multiplication

Elementwise multiplication between A and B works only if the 2 arrays have the exact same dimensions (unless A or B is a scalar). The resulting array will have the same dimensions.

A .* B = C



$$C_{ij} = A_{ij} B_{ij}$$

Addition and Subtraction in MATLAB

Matrix addition and subtraction is only meaningful if the matrices involved are of the same size.

However, MATLAB allows a **scalar to be added to an array of any size,** which is forbidden by the rules of matrix algebra.

$$\text{Let } A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

MATLAB interprets the expression `A + 2` as `A + 2*ones(3)` and returns:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + 2 \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \end{bmatrix}$$

Use the utility matrices below to practice functions & arithmetic operations

- **zeros(n,m)**: returns an [n x m] matrix of 0s
- **ones(n,m)**: returns an [n x m] matrix of 1s
- **eye(n,m)**: returns an [n x m] matrix filled with 1s in the main diagonal and 0s elsewhere

  if **A = eye(n,m)** then **A(i,j) = 1** if **i=j** and **A(i,j) = 0** if $i \neq j$

- **rand(n,m)**: returns an [n x m] matrix of random numbers uniformly distributed on the interval [0,1]
- **randn(n,m)**: returns an [n x m] matrix of random numbers following the standard normal distribution N(0,1)
- **diag(v)**: takes a vector v and creates a matrix with the elements of v along the main diagonal and 0s elsewhere.

The commands **zeros(n), eye(n),** etc. create square matrices.

A Few Important Matrix Functions

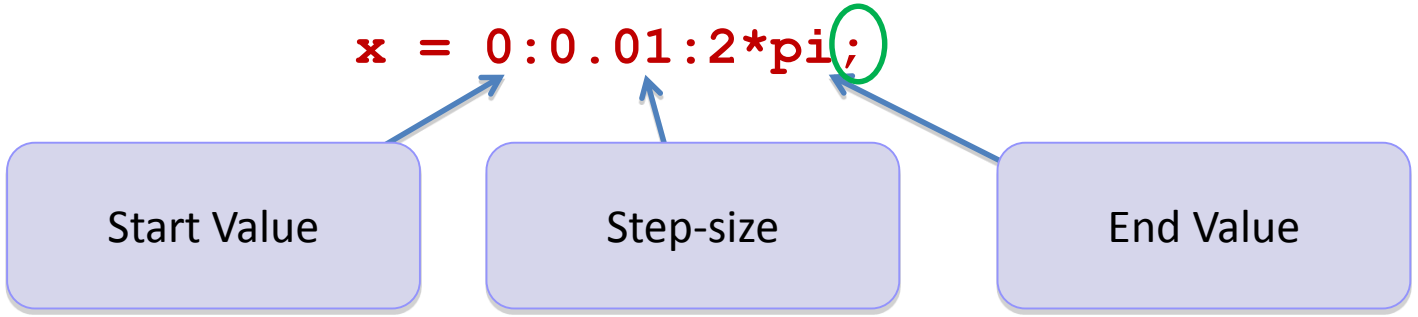
- **size(A)**: returns vector [n, m] (the dimensions of A)
- **det(A)**: returns the determinant of square (n x n) matrix A
- **inv(A)**: returns the inverse of square matrix A (if exists)
- **trace(A)**: returns the sum of the elements in the main diagonal of square matrix A:

$$trace(A) = \sum_{k=1}^{n} a_{kk}$$

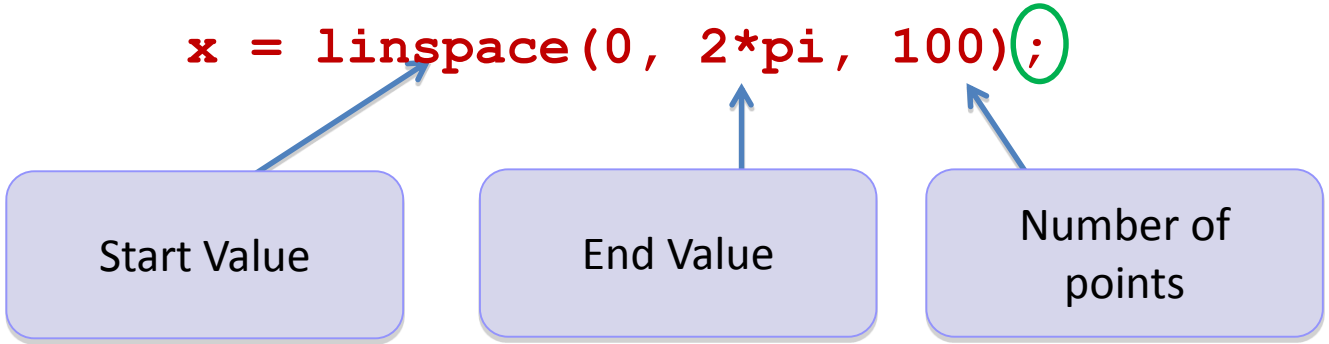
- **diag(A)**: returns a vector which includes the elements in the main diagonal of matrix A
- **flipud(A)**: flips (turns) matrix A upside down
- **fliplr(A)**: flips matrix A from left to right

The Colon Operator

- It is easy to create large, **evenly spaced vectors** using the colon (:) operator (we need such vectors to plot data)



- Alternatively, we can use the **`linspace`** function:

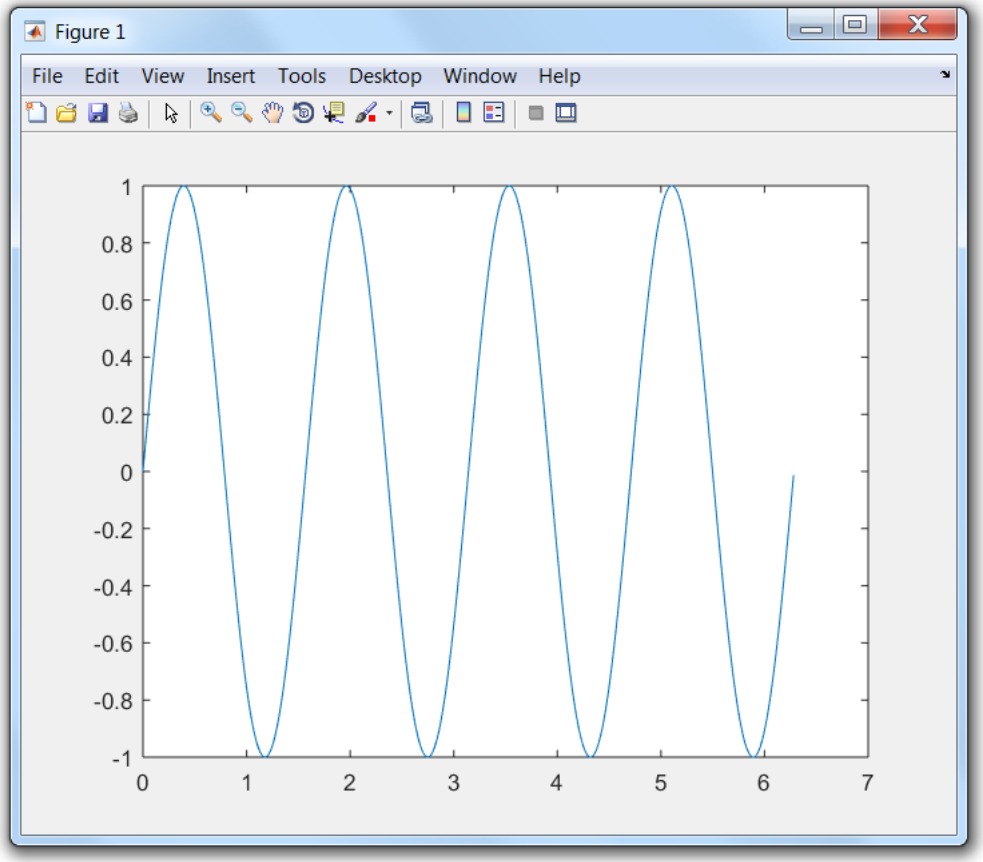


Use the (;) to hide the output!

Basic Plot Example

Command Window

```
>> x=0:0.01:2*pi; y=sin(4*x); plot(x,y)
>>
```



This is MATLAB's default, uncustomised plot output

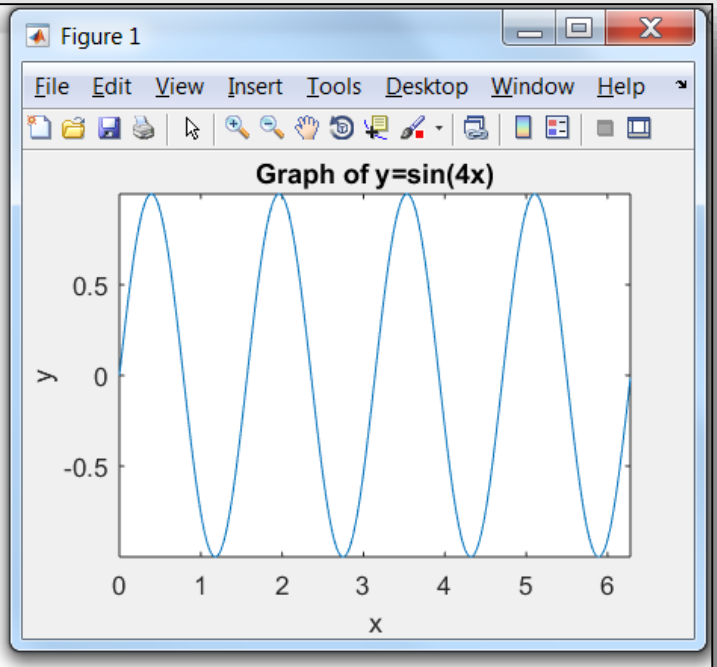
Make sure that vector *x* has a large enough number of elements to ensure a smooth curve

Plot Customisation

Command Window

```
>> x=0:0.01:2*pi; y=sin(4*x); plot(x,y)
>> title('Graph of y=sin(4x)')
>> xlabel('x'), ylabel('y'),
>> axis tight
>>
```

These commands must come after the plot command and can be entered 1-by-1 or all at once.



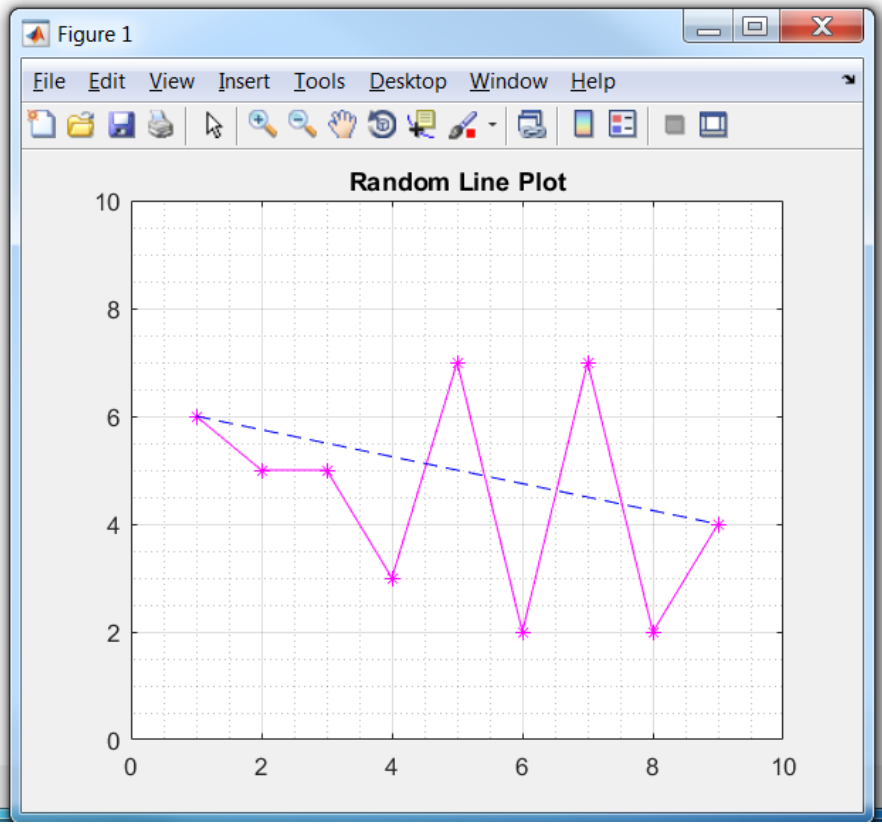
- We can add titles, labels & legends, change the plot area from the command line or from the Figure Window. For more details, visit

http://www.mathworks.co.uk/help/matlab/ref/colorspec.html
http://www.mathworks.co.uk/help/matlab/ref/linespec.html
http://www.mathworks.co.uk/help/matlab/creating_plots/using-high-level-plotting-functions.html

Random Plot Example

Command Window

```
>> x=1:9; y=randi(9,1,9);
>> hold on % to plot more than once in the same window
>> plot(x,y,'-*m')
>> plot([x(1), x(end)],[y(1),y(end)],'--b')
>> axis([0,10,0,10])
>> grid on, grid minor, box on
>> title('Random Line Plot')
>>
```



Each new **`plot`** command closes the previous Figure Window

Use **`hold on`** to keep the Figure Window open (to plot curves on top of each other)

Additional customization: look up **`axis, grid, box`**

Use the keyword **`end`** to select the last element of a vector (x(end), y(end))