University of
BRISTOL

Introduction To Scientific Computing

Basics of MATLAB

Dr. Fintan Healy
Room 1.31, Queens Building
fintan.healy@bristol.ac.uk

Lecture 2

Syntax Basics

Lecture Aims

- Familiarise you with basic MATLAB Syntax
- (Re-)Introduce you to core programming constructs
- Overview of basic plotting functionality

MATLAB

- By now, you should feel relatively *"comfortable"* with MATLAB
- During your studies, you have learnt basic programming concepts through the lens of the Python language
  - but all these concepts translate to MATLAB!
- Remember which language you used depends on the task:

Capability	MATLAB	Python	C++
Scientific Programming	Good	Good	Good
Data Science	Good	Good	Poor
Dashboards	Poor	Excellent	Poor
Plotting	Good	Good-	Poor
Real-time Control-systems	Excellent	Poor	Good
Experiments	Good	Okay	Poor
Documentation	Excellent	Good-	Okay
Debugging	Excellent	Good*	Okay
3rd party Integration	Poor	Good	Okay
Deep learning	Okay	Excellent	Good
Execution time	Okay	Okay	Excellent
Community Contributions	Okay	Excellent	Good

Opinions based on programming requirements for "general engineering" in industry

Basic Types

Data Types

Numeric Types

```
% Numeric Types
a = 1;
b = 1.3;
c = sqrt(pi);
d = 4.5e-3;
e = 8.6e26;

% Type casting
f = single(b);
g = int64(d);

% Complex numbers
h = 1 + 2i;
i = complex(1, 2);
```

Logical Types

```
% Logical Type
a = true;
b = false;

% Logical Operators
c = a & b; % Logical AND
d = a | b; % Logical OR
e = ~a;    % Logical NOT
f = xor(a, b); % Logical XOR
```

Data Types

Numeric Arrays	Logical Arrays

`a = [1,2,3];`
`b = [1+2i, 3+4i, 5+6i];`
`c = [0,0,1i];`
	`a = [true,false,false];`
	`b = [true,0,5];`

- You can create an array of values using square brackets “[ ]”, with values separated by a comma (,) or a space ( )
- You can create arrays of different types, e.g.:
  - – Numbers
  - – Complex numbers
  - – Logicals

Data Types

Character Arrays	Strings
a = 'test';	a = "test";
b = ['t','e','s','t’];	b = ["t", "e", "s", "t"];
a == b; % returns true	a == b; % returns false
c = ['The Answer is ', num2str(42)];	c = ["Mon", "Tue", "Wed", "Thu", "Fri"];
d = 'The Answer is 42’;	

- There are two ways to store text in MATLAB
  - **A Character Array** – use single quotes ‘ ’
  - **A String** – using double quotes “”
- They behave very differently!
  - A **character array** is “an array of single characters” and acts as **an array**!
  - A **String** acts as **one object**, so you can have an array of strings.

University of BRISTOL

Matrix Algebra

Vector and Matrix Definitions

- To define a row vector, we separate entities with spaces or commas

3	1	8

MATLAB

```
a = [3 1 8]
a = [3,1,8]
```

- To define a column vector, we separate entries by semicolons (which means start a new row):

7.3
2.1

MATLAB

```
a = [7.3;2.1]
```

- A matrix is then several rows separated by semicolons:

7	1	8
2	0	5

MATLAB

```
a = [7 1 8;2 0 5]
a = [7,1    8; 2 0,5;]
```

Array Constructors

- Array of zeros

```
a = zeros(1,3);
```

0	0	0

- Array of ones

```
a = ones(2);
b = ones(2,2);
```

1	1
1	1

- Identity Matrix

```
a = eye(3);
```

1	0	0
0	1	0
0	0	1

- Linearly spaced numbers

```
a = 0;
b = 1;
N = 5;
c = linspace(a, b, N);
```

0	0.25	0.5	0.75	1

“an array of N numbers, evenly spaced between a and b”

- Repeated pattern

```
m = 3;
n = 1;
x = [1,2,3];
c = repmat(x, m, n);
```

1	2	3
1	2	3
1	2	3

“repeat matrix ‘x’ , ‘m’ times in a row-wise direction and , ‘n’ times in a column-wise direction”

Array Concatenation

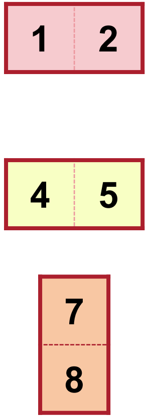
- We can also concatenate (combine) matrices and vectors together.
  - However, dimensions must agree.

MATLAB

```
a = [1 2];

b = [4 5];

c = [7;8];
```



1	2

4	5

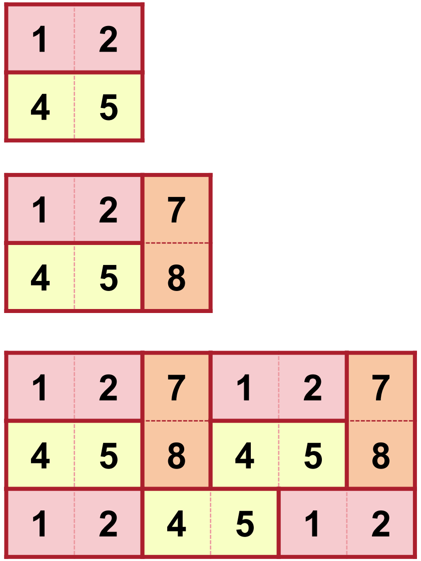
7
8

MATLAB

```
d = [a;b];

e = [d c];

f = [e e;a b a];
```



1	2
4	5

1	2	7
4	5	8

1	2	7	1	2	7
4	5	8	4	5	8
1	2	4	5	1	2

Vector Construction (the ‘:’ Operator)

- The colon operator can create ranges of values with fixed spacing

```
a = 1:5;
% equivalent to
a = [1,2,3,4,5];
```

- The default stepsize is 1. But you can also specify the stepsize by using two colons

```
start = 1;
stop = 7;
step = 2;
x = start:step:stop;
% equivalent to
x = [1,3,5,7];
```

```
x = 5:-2:0;
% equivalent to
x = [5,3,1];
```

Start at ‘a’, take steps of ‘b’. Stop when values are not between ‘a’ and ‘c’

Array Indexing

- You can access elements of an array using parentheses

3
7
5
6

```
a(1) % returns 3
a(3) % returns 5
```

- The keyword ‘end’ can be used to access the last item

5	6	7	8	9

```
a(3) % returns 7
a(end) % returns 9
a(end-1) % returns 8
```

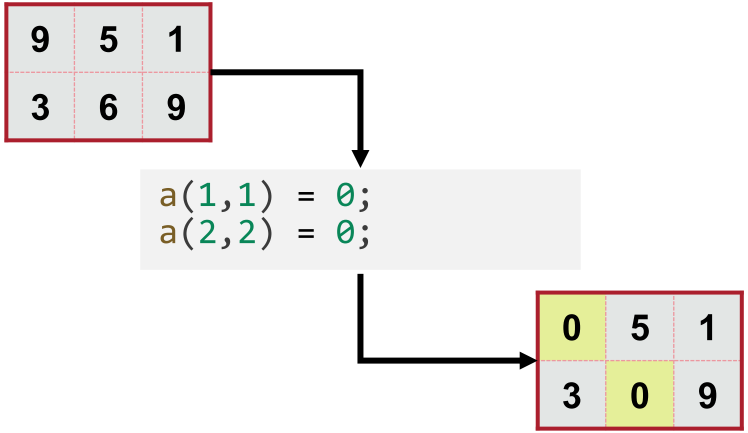
- You can access elements of a matrix using two parameters

7	1	8
3	5	1
2	0	5

row column

```
a(1,2) % returns 1
a(3,3) % returns 5
```

- You can alter elemental values of a matrix in a similar way



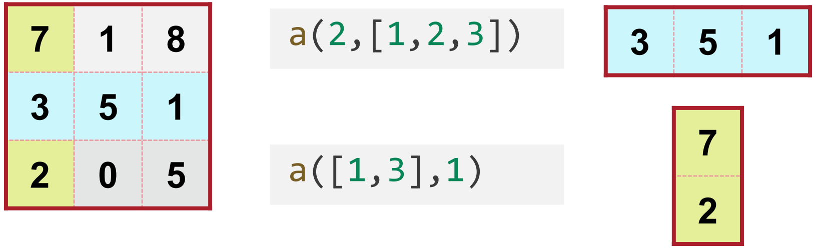
9	5	1
3	6	9

```
a(1,1) = 0;
a(2,2) = 0;
```

0	5	1
3	0	9

Array Indexing

- You can also access slices of an array, by passing arrays into the parameters



7	1	8
3	5	1
2	0	5

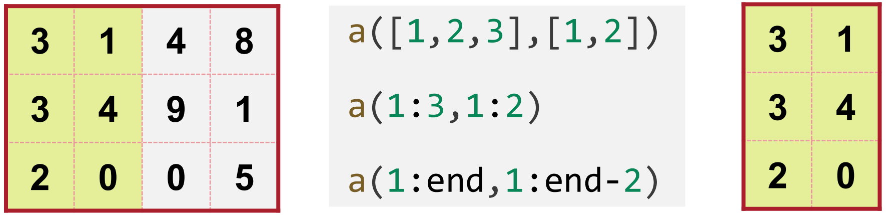
```
a(2,[1,2,3])
```

3	5	1

```
a([1,3],1)
```

7
2

- When taking slices, the colon operator and ‘end’ keyword become very powerful!



3	1	4	8
3	4	9	1
2	0	0	5

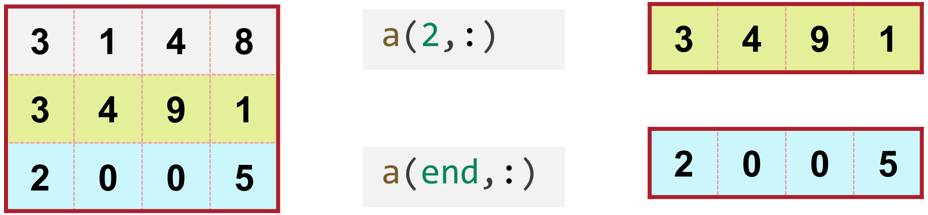
```
a([1,2,3],[1,2])

a(1:3,1:2)

a(1:end,1:end-2)
```

3	1
3	4
2	0

- A lone colon “:” can also be used to access an entire row or column.



3	1	4	8
3	4	9	1
2	0	0	5

```
a(2,:)
```

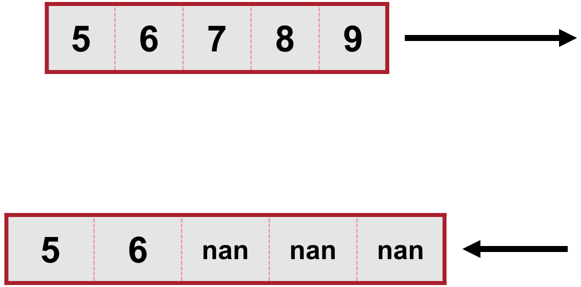
3	4	9	1

```
a(end,:)
```

2	0	0	5

Array Indexing

- Logical arrays can also be passed to arrays as an indexing parameter



5	6	7	8	9

5	6	nan	nan	nan

```
a = [5 6 7 8 9];
b = a>6;  % b = [false false true true true]

% the next three lines do the same thing
a(a>6) = nan;
a(b) = nan;
a([false, false, true, true, true]) = nan;
```

Matrix Manipulation

- MATLAB has a number of built-in functions to perform basic and complicated matrix manipulation.

Operation	Equation	MATLAB
Matrix multiplication	$\boldsymbol{C} = \boldsymbol{AB}$	`C=A*B;`
Transpose	$\boldsymbol{C} = \boldsymbol{A}^T$	`C=A’;`
Inverse	$\boldsymbol{C} = \boldsymbol{A}^{-1}$	`C=inv(A);`
Determinant	$\boldsymbol{C} = \det(\boldsymbol{A})$	`C=det(A);`
Pseudo-inverse	$\boldsymbol{C} = \boldsymbol{A}^{+}$	`C=pinv(A);`

Note

To do matrix multiplication, the number of columns of **A** must equal the number of rows of **B**.

Element-wise Matrix Manipulation

- We can also perform operations on elements of matrices (note, matrices must be the same size in all dimensions for this to work).
- We can add (or subtract) together each element of two matrices:

MATLAB

```
C = A + B;
```

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 + b_1 & a_2 + b_2 \\ a_3 + b_3 & a_4 + b_4 \end{bmatrix}$$

- We can also multiple (or divide) each element of two matrices using .* and ./:

MATLAB

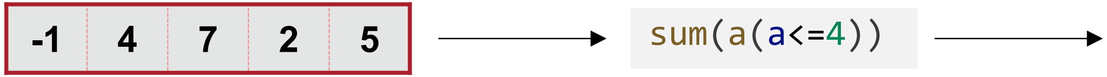
```
C = A .* B;
```

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \odot \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_2 \\ a_3 b_3 & a_4 b_4 \end{bmatrix}$$

Note

This is called the **Hadamard Product**; it is **not** the same as matrix multiplication!

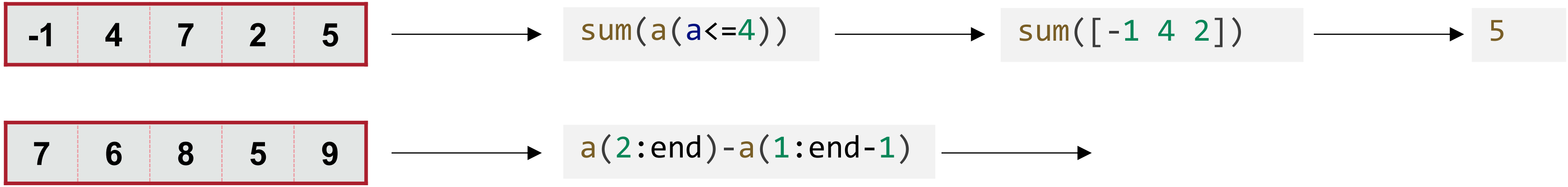
Some More Examples



-1	4	7	2	5

```
sum(a(a<=4))
```

Some More Examples



-1	4	7	2	5

```
sum(a(a<=4))
```

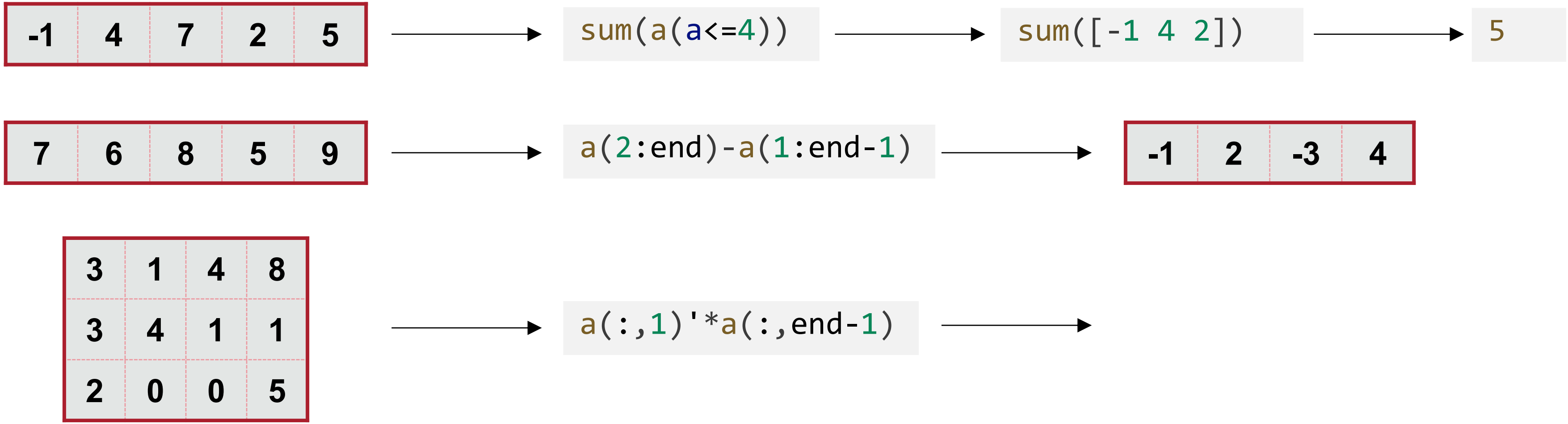
```
sum([-1 4 2])
```

```
5
```

7	6	8	5	9

```
a(2:end)-a(1:end-1)
```

Some More Examples



-1	4	7	2	5

```
sum(a(a<=4))
```

```
sum([-1 4 2])
```

```
5
```

7	6	8	5	9

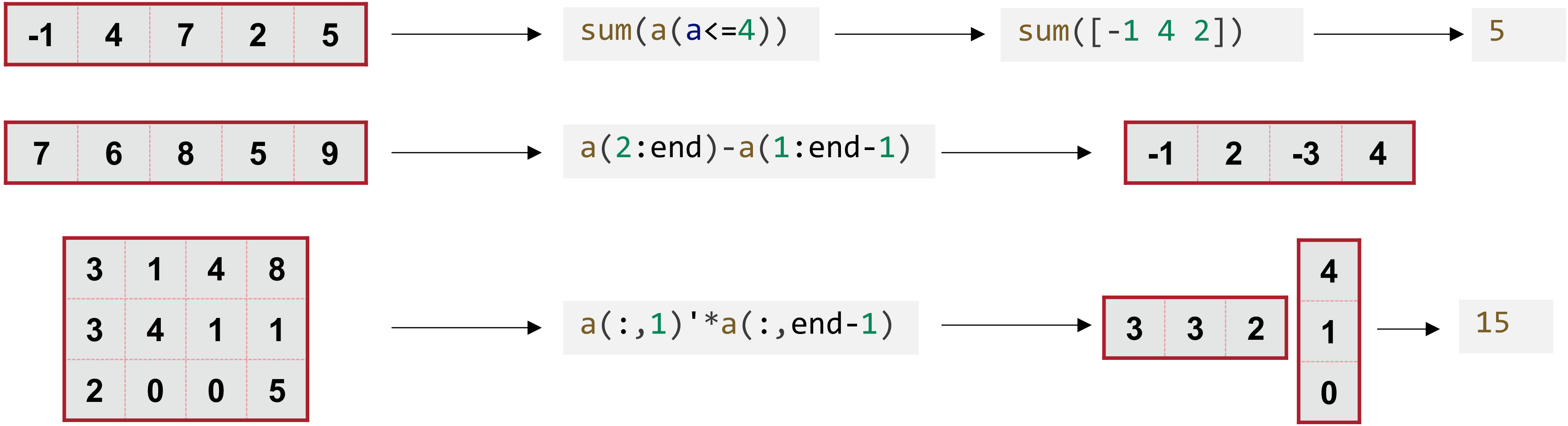
```
a(2:end)-a(1:end-1)
```

-1	2	-3	4

3	1	4	8
3	4	1	1
2	0	0	5

```
a(:,1)'*a(:,end-1)
```

Some More Examples



-1	4	7	2	5

```
sum(a(a<=4))
```

```
sum([-1 4 2])
```

```
5
```

7	6	8	5	9

```
a(2:end)-a(1:end-1)
```

-1	2	-3	4

3	1	4	8
3	4	1	1
2	0	0	5

```
a(:,1)'*a(:,end-1)
```

3	3	2

4
1
0

```
15
```

Core Coding Constructs

Control Flow - IF

- In MATLAB, we can use conditional statements (if, else if, else) to branch off during the execution of our code.
- Unlike Python, the keyword ‘end’ is used to determine the end of the statement.

MATLAB

```
if <expression>
   <statements>
elseif <expression>
   <statements>
else
   <statements>
end
```

MATLAB

```
if aileronIN > 30.0
   aileronOUT = 30.0;
elseif aileronIN < -30.0
   aileronOUT = -30.0;
else
   aileronOUT = aileronIN;
end
```

Note

Indenting is only aesthetic (unlike Python, where it forms part of the syntax)

Control Flow - IF

- The full *if-elseif-else* statement can be simplified down to *if*, *if-else* or *if-elseif* as appropriate.
- Common boolean operators are similar to those found in Python:

MATLAB

```
if aileronIN > 30.0
   aileronOUT = 30.0;
end
```

Expression	True if
`a < b`	a is strictly less than b
`a > b`	a is strictly greater than b
`a <= b`	a is less than or equal to b
`a >= b`	a is greater than or equal to b
`a == b`	a is equal to b
`a ~= b`	a is not equal to b
`a & b`	both a and b are true
`a | b`	either a or b is true

Control Flow – Switch

- A similar flow can be achieved with a switch-case statement
- A switch-case block selects and executes code based on the value of a variable, running the “code block” for the first matching case.

MATLAB

```
mode = 1;
switch a
      case 1
         disp('Enabled')
      case 0
         disp('Disabled')
      case 2
         disp('Paused')
      otherwise
         disp('Unknown Mode')
end
```

MATLAB

```
Day = "mon";
switch Day
   case {"Mon", "Tue", "Wed", "Thu"}
      disp("Weekday")
   case "Friday"
      disp("End of the work week")
   case {"Sat", "Sun"}
      disp("Weekend")
   otherwise
      disp("Unknown Day")
end
```

What will this script display?

Note

The function ‘*disp*’ prints to the command line.

Control Flow - FOR

- A for loop repeats a block of code a specific number of times. It’s used when you know how many times you want to loop.

MATLAB

```
for <variable> = <array>
   <code>
end
```

“Assign each value from the array to the variable and execute the code block”

MATLAB

```
A = zeros(1,5);
for i = [1,3,5]
   A(i) = i^2;
end

for i = 1:length(A)
   A(i) = A(i) + i^2;
end
```

MATLAB

```
A = zeros(4,3);
for i = 1:size(A,1)
   for j = 1:size(A,2)
      A(i,j) = i*j;
   end
end
```

Note

‘length(A)’ returns the length of the longest dimension in A
‘size(A,n)’ returns the size of the N’th dimension of A (rows=dim 1,columns=dim 2)

Control Flow - FOR

- Two special keywords can alter the flow of for loops:
  - “continue”: stop execution of code block and go to next iteration
  - “break”: exit the loop

MATLAB

```
for i = 1:10
   if i==4 | i==1
      continue; % skip rest of code block
   elseif i==7
      break;    % exit for loop
   end
   disp(i)
end
```

Control Flow - FOR

- Two special keywords can alter the flow of for loops:
  - “continue”: stop execution of code block and go to next iteration
  - “break”: exit the loop

MATLAB

```
for i = 1:10
    if i==4 | i==1
        continue; % skip rest of code block
    elseif i==7
        break;    % exit for loop
    end
    disp(i)
end
```

→

```
2
3
5
6
```

Control Flow - WHILE

- A while loop keeps running as long as a condition is true. It’s used when you don’t know in advance how many times you’ll need to loop.
- ‘continue’ and ‘break’ can also be used in a while loop

MATLAB

```
i = 1;
res = [];
while i < 60
   i = i*3;
   if mod(i,2) == 0 % skip even numbers
      continue;
   end
   res(end+1) = i;
end
disp(res);
```

Note

‘mod(x,n)’ computes the modulus, which is the remainder when x is divided by n

Control Flow - WHILE

- A while loop keeps running as long as a condition is true. It’s used when you don’t know in advance how many times you’ll need to loop.
- ‘continue’ and ‘break’ can also be used in a while loop

MATLAB

```
i = 1;
res = [];
while i < 60
   i = i*3;
   if mod(i,2) == 0 % skip even numbers
      continue;
   end
   res(end+1) = i;
end
disp(res);
```

→

```
3   9   27   81
```

Note

‘mod(x,n)’ computes the modulus, which is the remainder when x is divided by n

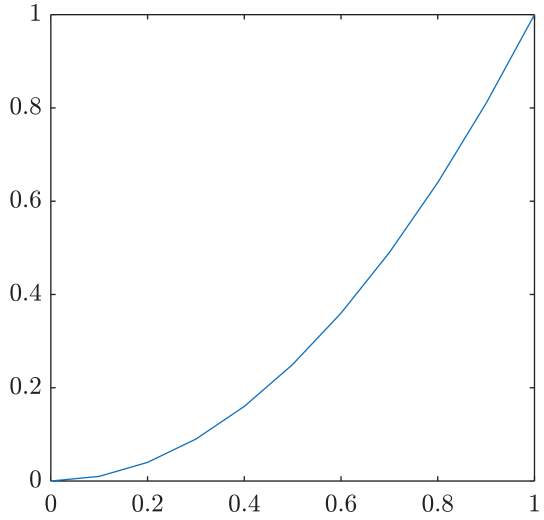
Plotting – Quick Intro

Plotting A Single Line

- MATLAB has a powerful built-in plotting facility.
  - ‘help’ and ‘doc’ commands are helpful here. The array of options & parameters available is vast!
- We’ll quickly cover the basics in these slides.
- MATLAB allows for 2D plots of one vector against another
  - ‘figure’ command creates a new figure window

MATLAB

```
figure;
t = 0:0.1:1;
y = t.^2;
plot(t,y);
```



Note

The vectors must have the same length!

Plotting Multiple Lines

- By default, multiple calls to ‘plot’ will overwrite the previous line
- Use the command ‘hold on’ to allow multiple lines on the same graph

Note

The syntax ‘hold on’ may seem strange. This is a piece of *syntactic sugar.* A space passes the next word as a character array to the function. e.g.

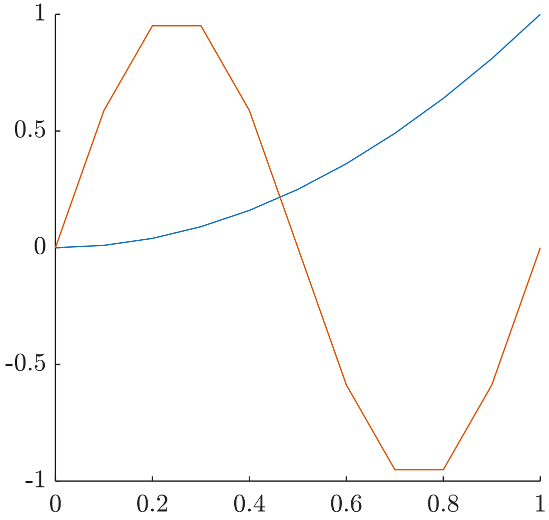
```
hold on;
hold('on’);
```

Are equivalent

MATLAB

```
figure;
hold on;
t = 0:0.1:1;
y = t.^2;
plot(t,y);

y2 = sin(2*pi*t);
plot(t,y2);
```



Plotting Multiple Lines

- We can also change various aspects of the plot area and the line styles.
- To change the line style, add a string of characters to the plot command
  - See ‘*doc plot*’ for full details
- Other common commands:

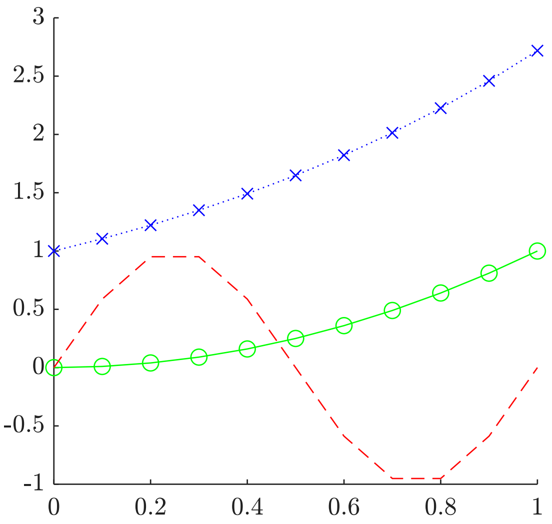
Command	Action
`axis([XMIN XMAX YMIN YMAX])`	Scale axes
``title(`text')``	Add the title ‘text’ to the graph
`xlabel('text')`	Add the label ‘text’ to the x axis
``legend(`line1',`line2')``	Add a legend for each of the lines
`grid`	Toggle grid lines on and o
`close all`	Close all opened figure windows
`clf`	Clf plots on current figure

MATLAB

```
y = t.^2;
plot(t,y,'go-’);
%green line with circle markers

y2 = sin(2*pi*t);
plot(t,y2,'r--’);
%red dashed line

y3 = exp(t);
plot(t,y3,'bx:’);
%blue dotted line with x markers
```



SUMMARY

Summary

- Have introduced MATLAB syntax
- Have re-introduced coding constructs
- Have covered the basics of plotting

https://i2sc.fintanhealy.co.uk/

This Week

- Attempt as many of the worksheets as possible
- Second (and final) Lab next Tuesday
- Note – more tasks than you can complete in four hours - provided for practice