Complex Numbers, Matrices & MatLab

Contents

1 Logic, Binary, Bits & Bytes
2 Complex Numbers
- 2.1 Butterflies & fish
- 2.2 Cartesian representation
- 2.3 Roots of unity
- 2.4 Polar representation

3 Introduction to Matrices
- 3.1 What *is* a matrix?
- 3.2 Basic arithmetic
- 3.3 Creating matrices in MatLab
- 3.4 Creating sequences in MatLab

4 Complex Numbers in MatLab
5 Elemental Operations
- 5.1 Element-wise arithmetic
- 5.2 Examples of element-wise arithmetic
- 5.3 Examples of element-wise functions
- 5.4 Extracting elements of a matrix
- 5.5 Changing elements of a matrix
- 5.6 Strings in MatLab

6 Functions and Plots in MatLab
- 6.1 Simple functions
- 6.2 Functions and matrices
- 6.3 Functions and ***fplot***
- 6.4 Simple plots
- 6.5 Multiple plots

7 *m*-Files
- 7.1 Execution & editing
- 7.2 General comments
- 7.3 Checklist

8 Input / Output
- 8.1 Numerical input / output
- 8.2 String input / output

9 Basic Programming
- 9.1 Loops with ‘for’
- 9.2 Logical expressions and ‘if’
- 9.3 Controlling loops
- 9.4 More on getting user input
- 9.5 Comparing strings
- 9.6 Checking numerical input
- 9.7 Warnings, errors and asserts

10 Function m-Files
- 10.1 Function declaration and help
- 10.2 General comments
- 10.3 Using functions
- 10.4 Nested loops with ‘for’

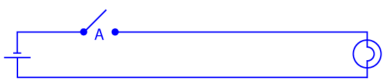
Contents

11 Properties of Plots
11.1 Line plots
11.2 3D plots
12 Vectors & Matrices
12.1 Vector scalar (or ‘dot’) product
12.2 Matrix multiplication
12.3 Matrix powers and inverse
12.4 Simultaneous equations
12.5 Eigenvalues & eigenvectors
13 Ordinary Differential Equations
13.1 First order ODEs
13.2 Vector ODEs
13.3 Second order ODEs

1 Logic, Binary, Bits & Bytes

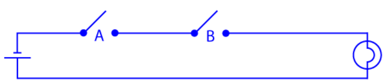
Computers are all about ones and zeros. Computer scientists have a joke:

There are 10 types of people in the world: Those who understand binary, and those who don’t…



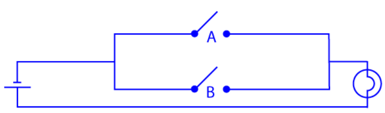
Switch A	Lamp
Up	Off
Down	On

The lamp is On if Switch A is Down.



Switch A	Switch B	Lamp
Up	Up	Off
Up	Down	Off
Down	Up	Off
Down	Down	On

The lamp is On if Switch A is Down **AND** Switch B is Down.



Switch A	Switch B	Lamp
Up	Up	Off
Up	Down	On
Down	Up	On
Down	Down	On

The lamp is On if Switch A is Down **OR** Switch B is Down.

Essentially the lamp has two states: On (if there is a voltage across the lamp) and Off.

In binary, ‘0’ means ‘false’ or ‘no’ or ‘nothing’ or ‘off’; while ‘1’ means ‘true’ or ‘not 0’. Data is stored in computer memory, or on hard drives (or USB pen drives or DVDs, etc.) as a large collection of ones and zeros. Digital transmissions are long strings of ones and zeros.

1 Logic, Binary, Bits & Bytes

Each individual 1 or 0 is called a ‘bit’:

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

A byte is a sequence of 8 bits and, by itself, represents an integer in the range 0-255:

1	0	0	1	1	0	0	1	1	0	0	0	1	0	1	1	0	1	0	0	1	1	0	1
128	*64*	*32*	*16*	*8*	*4*	*2*	*1*	*128*	*64*	*32*	*16*	*8*	*4*	*2*	*1*	*128*	*64*	*32*	*16*	*8*	*4*	*2*	*1*

153	139	77
99	8B	4D

The last row is the same values represented in Base 16 (hexadecimal) which is often used to represent values of bytes. In Base 16 the letters A-F (or a-f) represent the numbers 10-15:

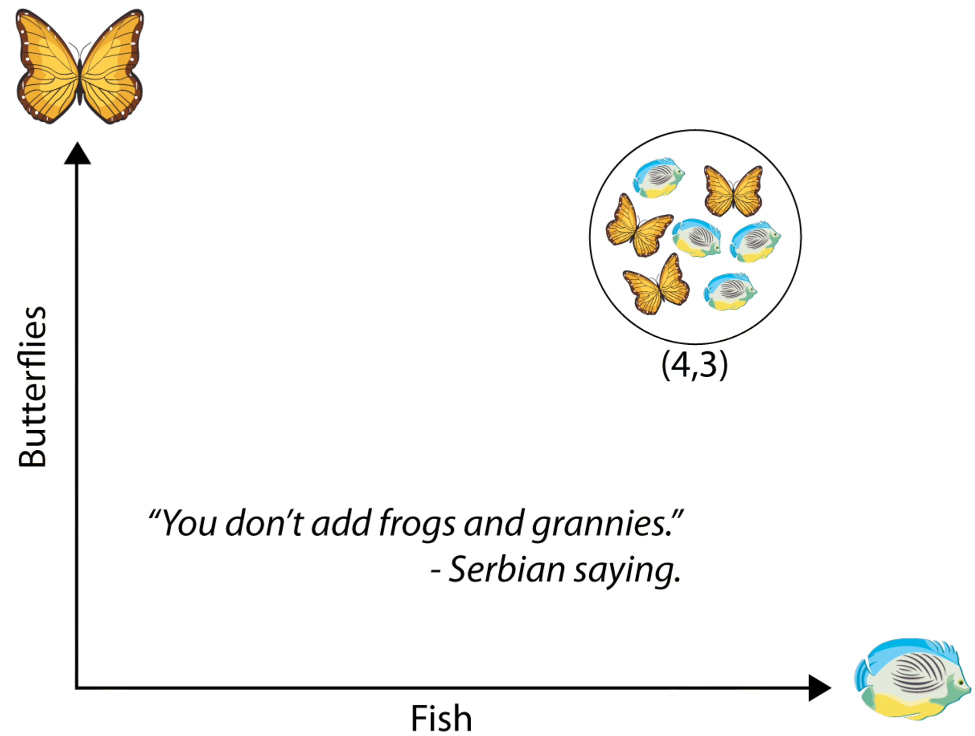
Base 10	Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Base 16	Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Computers usually use a sequence of 4 bytes to represent an integer and a sequence of 8 bytes to represent a ‘double precision’ floating point number (a *real* number, with 16 significant figures).

For example, the sequence of three bytes at the top might represent the red, green and blue components of colour (0=‘none’, 255=‘full’) of a single pixel in a 24-bit colour image. My ‘six megapixel’ (6MP) camera takes 2816×2112 photos, i.e., 5947392 pixels, or 17842176 bytes of red-green-blue data (which compresses to about 13% of this when saved).

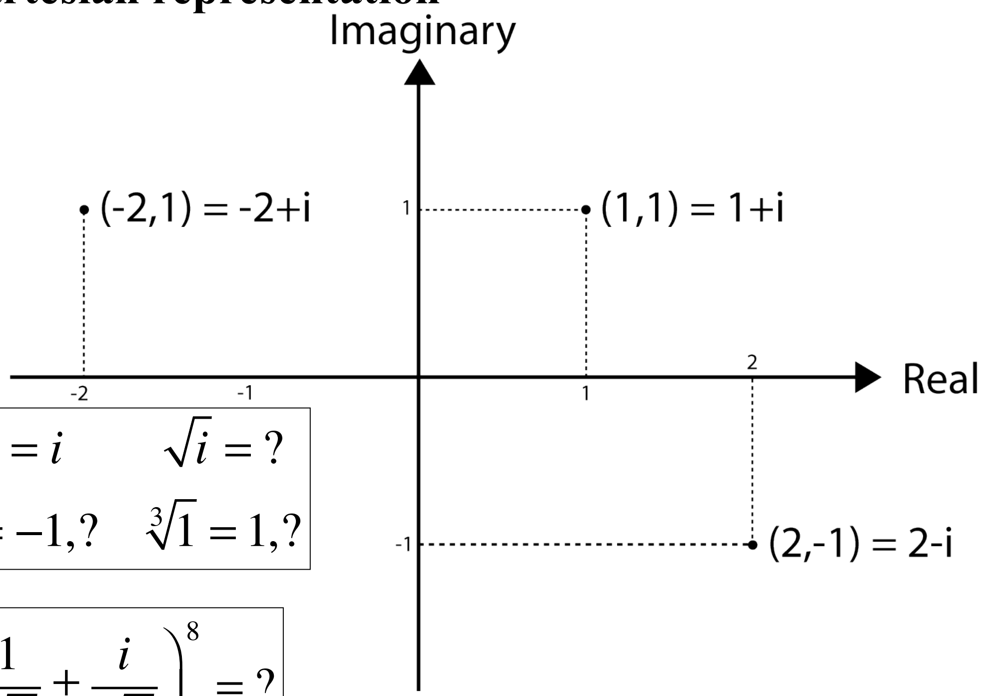
2 Complex Numbers

2.1 Butterflies & fish



2 Complex Numbers

2.2 Cartesian representation



$$\sqrt{-1} = i \qquad \sqrt{i} = ?$$

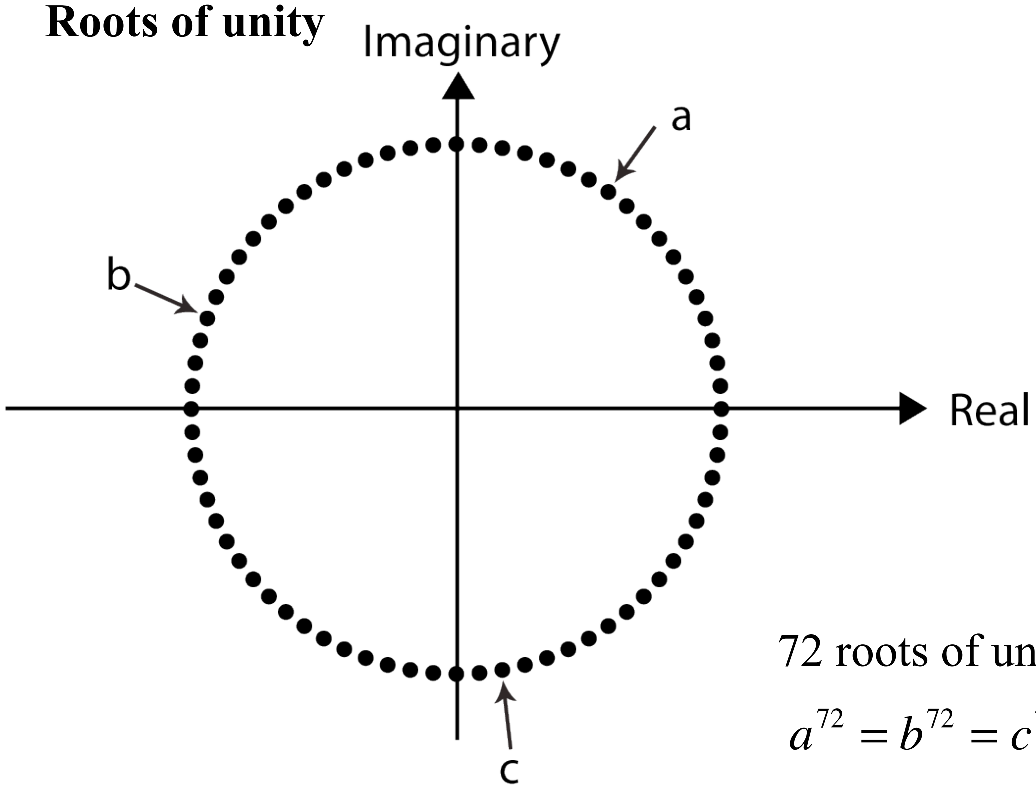
$$\sqrt[3]{-1} = -1, ? \qquad \sqrt[3]{1} = 1, ?$$

$$\left(\pm \frac{1}{\sqrt{2}} \pm \frac{i}{\sqrt{2}} \right)^8 = ?$$

Mechanical Engineering Professional Skills: Introduction to Computing

2 Complex Numbers

2.3 Roots of unity



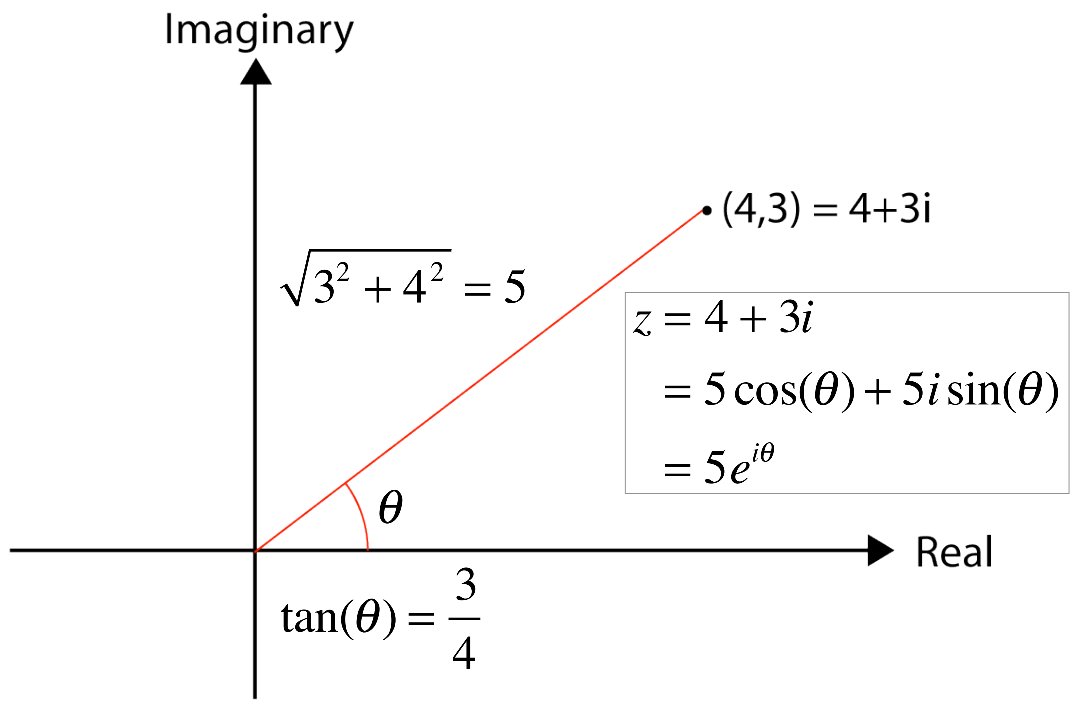
72 roots of unity:

$$a^{72} = b^{72} = c^{72} = 1$$

University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

2 Complex Numbers

2.4 Polar representation



3 Introduction to Matrices

3.1 What *is* a matrix?

A matrix is an ordered list of numbers.

$$7 = \text{a } \textbf{scalar}.$$

$$\begin{pmatrix} 7 \end{pmatrix} = \text{a } 1\times 1 \textbf{ matrix}.$$

$$\begin{pmatrix} 1 & 3 & 4 \end{pmatrix} = \text{a } 1\times 3 \textbf{ matrix}\text{, or } \underline{\textbf{row vector}}.$$

$$\begin{pmatrix} 4 \\ 2 \end{pmatrix} = \text{a } 2\times 1 \textbf{ matrix}\text{, or } \underline{\textbf{column vector}}.$$

$$\begin{pmatrix} 1 & 0 & 6 \\ -7 & 1 & 4 \\ 3 & -7 & 2 \end{pmatrix} = \text{a } 3\times 3 \ \underline{\textbf{square}} \textbf{ matrix}.$$

$$\begin{pmatrix} 1 & 0 & -1 & -3 \\ 3 & 4 & 0 & 6 \end{pmatrix} = \text{a } 2\times 4 \textbf{ matrix} \text{ (i.e., 2 rows, 4 columns)}$$

3 Introduction to Matrices

3.2 Basic arithmetic

Multiplication (by scalar) $$2*\begin{pmatrix}1 & 3 & 4\end{pmatrix}=\begin{pmatrix}2 & 6 & 8\end{pmatrix}$$

Division (by scalar) $$\begin{pmatrix}1 & 3 & 4\end{pmatrix}\div 2=\begin{pmatrix}\frac{1}{2} & \frac{3}{2} & 2\end{pmatrix}$$

Addition $$\begin{pmatrix}1 & 3 & 4\end{pmatrix}+\begin{pmatrix}2 & 6 & 8\end{pmatrix}=\begin{pmatrix}3 & 9 & 12\end{pmatrix}$$

Subtraction $$\begin{pmatrix}1 & 3 & 4\end{pmatrix}-\begin{pmatrix}2 & 6 & 8\end{pmatrix}=\begin{pmatrix}-1 & -3 & -4\end{pmatrix}$$

Matrices of different sizes cannot be added to or subtracted from each other!

Addition and subtraction are element-by-element.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

3 Introduction to Matrices

3.3 Creating matrices in MatLab

```
>> x = [1 3 7]
x =
     1     3     7
```

$$x=\begin{pmatrix}1 & 3 & 7\end{pmatrix}$$

```
>> y = [pi,i]
y =
     3.1416      0 + 1.0000i
```

$$y=\begin{pmatrix}\pi & i\end{pmatrix}$$

```
>> z = [1;3;7]
z =
     1
     3
     7
```

$$z=\begin{pmatrix}1\\3\\7\end{pmatrix}$$

To create a matrix, place values between '[' and ']'.

Use semicolons (';') to separate matrix rows.

Use commas (',') to separate elements within rows.

```
>> A = [1,3;7,9;5,-5]
A =
     1     3
     7     9
     5    -5
```

$$A=\begin{pmatrix}1 & 3\\7 & 9\\5 & -5\end{pmatrix}$$

3 Introduction to Matrices

3.4 Using sequences in MatLab

Use a colon to create a matrix with a sequence of numbers. By default this increases in steps of 1:

```
>> x = [1:3]
x =
     1     2     3
```

$x = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$

In general, `[a:b:c]` starts at `a`, increases in steps of `b` (which may be negative or non-integer, but not complex), and ends at or before `c`:

```
>> y = [0:2:5]
y =
     0     2     4
>> z = [7:-3:-8]
z =
     7     4     1    -2    -5    -8
```

$y = \begin{pmatrix} 0 & 2 & 4 \end{pmatrix}$

$z = \begin{pmatrix} 7 & 4 & 1 & -2 & -5 & -8 \end{pmatrix}$

Sequences can also be combined. This uses two:

```
>> [-1:1,2:-0.25:1]
ans =
    -1.00   0      1.00   2.00   1.75   1.50   1.25   1.00
```

$\begin{pmatrix} -1 & 0 & 1 & 2 & 7/4 & 3/2 & 5/4 & 1 \end{pmatrix}$

Important: It is usually best to make `a`, `b`, and `c` integers to avoid numerical accuracy problems.

4 Complex numbers in MatLab

```
>> x = 4 + 3 * i
x =
   4.0000 + 3.0000i
```

Let $x = 4 + 3i$

```
>> real (x)
ans =
     4
```

Re(x) - i.e., what is the real component of x?

```
>> imag (x)
ans =
     3
```

Im(x) - i.e., what is the imaginary component of x?

```
>> abs (x)
ans =
     5
```

$|x|$ - i.e., what is the absolute value of x?

```
>> i.^[0:3]
ans =
   1.0000     0 + 1.0000i    -1.0000     0 - 1.0000i
>>
```

$\begin{pmatrix} i^0 & i^1 & i^2 & i^3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & i & -1 & -i \end{pmatrix}$

5 Elemental Operations

5.1 Element-wise arithmetic subtitle

Let **A** and **B** be $n \times m$ matrices, i.e., matrices with n rows and m columns:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{pmatrix}$$

In MatLab the element-wise operators such as `.^` work like:

$$\mathbf{A}.\hat{}\,\mathbf{B} = \begin{pmatrix} {a_{11}}^{b_{11}} & {a_{12}}^{b_{12}} & \cdots & {a_{1m}}^{b_{1m}} \\ {a_{21}}^{b_{21}} & {a_{22}}^{b_{22}} & \cdots & {a_{2m}}^{b_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ {a_{n1}}^{b_{n1}} & {a_{n2}}^{b_{n2}} & \cdots & {a_{nm}}^{b_{nm}} \end{pmatrix}$$

but the matrices (**A** and **B**) <u>must be the same size</u>.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

5 Elemental Operations

5.2 Examples of element-wise arithmetic

```
>> [1,2,3] .* 2
ans =
     2     4     6
>> [1,2,3] .* [4,5,6]
ans =
     4    10    18
>> [1,2,3] .^ 2
ans =
     1     4     9
>> [1,2,3] .^ [4,5,6]
ans =
     1    32   729
>> 2 .^ [4,5,6]
ans =
    16    32    64
>> 2 ./ [4,5,6]
ans =
    0.5000    0.4000    0.3333
>>
```

$$(1 \quad 2 \quad 3).*2 = (1*2 \quad 2*2 \quad 3*2) = (2 \quad 4 \quad 6)$$

$$(1 \quad 2 \quad 3).*(4 \quad 5 \quad 6) = (1*4 \quad 2*5 \quad 3*6) = (4 \quad 10 \quad 18)$$

$$(1 \quad 2 \quad 3).\hat{}\,2 = (1^2 \quad 2^2 \quad 3^2) = (1 \quad 4 \quad 9)$$

$$(1 \quad 2 \quad 3).\hat{}\,(4 \quad 5 \quad 6) = (1^4 \quad 2^5 \quad 3^6) = (1 \quad 32 \quad 729)$$

$$2.\hat{}\,(4 \quad 5 \quad 6) = (2^4 \quad 2^5 \quad 2^6) = (16 \quad 32 \quad 64)$$

$$2./(4 \quad 5 \quad 6) = \begin{pmatrix} \frac{2}{4} & \frac{2}{5} & \frac{2}{6} \end{pmatrix} = (0.5 \quad 0.4 \quad 0.333\dot{3})$$

Mechanical Engineering Professional Skills: Introduction to Computing

University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

5 Elemental Operations

5.3 Examples of element-wise functions

```
>> log ([-1,1,i])
ans =
      0 + 3.1416i     0     0 + 1.5708i
```

$$=\begin{pmatrix}\ln(-1) & \ln(1) & \ln(i)\end{pmatrix}$$
$$=\begin{pmatrix}\pi i & 0 & \pi i/2\end{pmatrix}$$

```
>> sin([0:(pi/2):(pi*2)])
ans =
      0     1.0000     0.0000     -1.0000     -0.0000
```

$$=\begin{pmatrix}\sin(0) & \sin(\pi/2) & \sin(\pi) & \sin(3\pi/2) & \sin(2\pi)\end{pmatrix}$$
$$=\begin{pmatrix}0 & 1 & 0 & -1 & 0\end{pmatrix}$$

```
>> f = inline ('z.^2+c','z','c');
>> f([-1+i,1+i;-1-i,1-i],i)
ans =
      0 - 1.0000i     0 + 3.0000i
      0 + 3.0000i     0 - 1.0000i
>>
```

Let: $f(z,c)=z.\hat{}2+c$

$$f\left(\begin{pmatrix}-1+i & 1+i\\ -1-i & 1-i\end{pmatrix},i\right)=\begin{pmatrix}-1+i & 1+i\\ -1-i & 1-i\end{pmatrix}.\hat{}2+i$$
$$=\begin{pmatrix}(-1+i)^2+i & (1+i)^2+i\\ (-1-i)^2+i & (1-i)^2+i\end{pmatrix}$$
$$=\begin{pmatrix}-i & 3i\\ 3i & -i\end{pmatrix}$$

Mechanical Engineering Professional Skills: Introduction to Computing

University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

5 Elemental Operations

5.4 Extracting elements of a matrix

Elements of vectors and matrices can be extracted; rows and columns of matrices can also be extracted.

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

Assign this 2×4 matrix to **A**.
What is **A**?

```
>> A(1,:)
ans =
      1      2      3      4
```

What is the first ***row*** of **A**?

```
>> b = A(2,:)
b =
      5      6      7      8
```

Assign the second ***row*** of **A** to **b**, i.e., **b** is a ***row vector***.

```
>> A(:,3)
ans =
      3
      7
```

What is the third ***column*** of **A**?

```
>> A(1,4)
ans =
      4
```

What is the element in the fourth ***column*** of the first ***row***?

5 Elemental Operations

5.5 Changing elements of a matrix

Elements, rows and columns of matrices can also be changed.

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

Assign this 2×4 matrix to **A**. What is **A**?

```
>> A(:,2) = [-2;-6]
A =
     1    -2     3     4
     5    -6     7     8
```

Overwrite the second ***column*** of **A** with a new ***column vector***.

```
>> A(1,:) = -A(2,:)
A =
    -5     6    -7    -8
     5    -6     7     8
```

Overwrite the first ***row*** of **A** with the negative of the second ***row*** of **A**.

```
>> A(2,3) = A(1,1) + A(2,4)
A =
    -5     6    -7    -8
     5    -6     3     8
>>
```

Add the first ***element*** of the first ***row*** to the fourth ***element*** of the second ***row***, and assign the value (-5+8=3) to the third ***element*** of the second ***row***.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

5 Elemental Operations

5.6 Strings in MatLab

A ***string*** in MatLab is treated like a ***row vector*** of characters (i.e., letters, numbers, punctuation), e.g.:

```
>> H = ['H','e','l','l','o',','];
>> W = 'World!';
>> HW = [H,' ',W]
HW =
Hello, World!
```

Let: H = ‘Hello,’.
Let: W = ‘World!’.
Let: $HW = H +$ ‘ ’ $+ W$
i.e., HW = ‘Hello, World!’.

```
>> W(6)
ans =
!
```

The 6th character in W… … is ‘!’.

```
>> [H(2),H(6),W(2)]
ans =
e,o
>>
```

The 2nd and 6th characters in H and the 2nd character in W… … i.e., ‘e’, ‘,’ and ‘o’ … so ‘e,o’.

6 Functions and Plots in MatLab

6.1 Simple functions

A function is a mathematical object which takes one value (or set of values) and turns it into another value (or set of values).

For example, the function *sin* takes any real number and turns it into a real number between -1 and 1.

When a function f takes a value x (the variable) and returns a value y, we write:

$$y = f(x)$$

MatLab has many functions defined, such as *sin*, *tanh*, *acos*, *exp* and *log*.

It is possible to define new functions. For example, if we want:

$$f(x) = \sec^2(x) - \tan(x)$$

The traditional way to define functions in MatLab is to use the ***inline*** command.

```
>> f = inline ('sec(x)^2-tan(x)','x');
```

The first parameter is the function definition and the other parameters are the function variables. In the most recent release of MatLab, it is also possible (and recommended) to define functions using the new @ command like this:

```
>> f = @(x) sec(x)^2-tan(x);
```


6 Functions and Plots in MatLab

6.2 Functions and matrices

Mechanical Engineering Professional Skills: Introduction to Computing

Functions can take vectors or matrices as variables, and the function value can also be a vector or matrix, e.g.:

```
>> g = inline ('x.^2 - x.*y + y.^2', 'x', 'y');
>> a = [1,2,3];
>> b = [0,1,-1];
>> g(a,b)
ans =
      1     3    13
```

Or, as an example of a function taking a real number and returning a matrix (representing a rotation through angle w in three dimensions about the z-axis):

$$R_Z(w) = \begin{pmatrix} \cos w & -\sin w & 0 \\ \sin w & \cos w & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This is defined like:

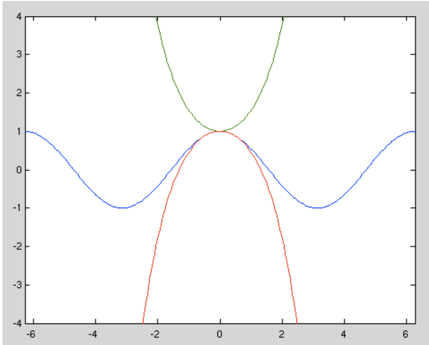
```
>> Rz = @(w) [cos(w),-sin(w),0;sin(w),cos(w),0;0,0,1];
```

University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

6 Functions and Plots in MatLab

6.3 Functions and *fplot*

In MatLab, one way to plot several different functions at the same time is to define a single function which returns several different values.



For example, the function, $pqr(x)$, takes a value, x, and returns a ***row vector*** of values:

$$\text{Let: } pqr(x) = \begin{pmatrix} \cos(x) & \cosh(x) & 2-\cosh(x) \end{pmatrix}$$

You can define this functions like this:

```
>> pqr = @(x) [cos(x),cosh(x),2-cosh(x)];
```

or you can define it using the ***inline*** command:

```
>> pqr = inline ('[cos(x),cosh(x),2-cosh(x)]','x');
```

To plot this function in MatLab, you can use ***fplot***:

Bounding Box
Specifies the range on the x-axis and y-axis.

```
>> fplot (pqr, [-2*pi,2*pi,-4,4]);
```

Alternatively, you can just pass the function definition to ***fplot***:

```
>> fplot ('[cos(x),cosh(x),2-cosh(x)]', [-2*pi,2*pi,-4,4]);
```

Because the function definition returns a ***row vector***, ***fplot*** plots multiple curves.

6 Functions and Plots in MatLab

6.4 Simple plots

MatLab provides many ways to plot graphs. The command ***plot*** takes two vectors of the same size representing x and y values and creates a plot of these as points and/or lines, e.g.:

```
>> x = [0,1,3,7,8];
>> y = [2,3,2,-3,-5];
>> plot (x, y, 'r+');
```

plots data as points using red '+' marks; no line is drawn. Alternatively:

```
>> plot (x, y, 'b-');
```

would draw a solid blue line between the points. For more information on ***plot***, type:

```
>> help plot
```

To plot a function rather than vectors of discrete data, use ***fplot*** instead.

You can change a figure's current axes and add a title and axis labels using the ***axes***, the ***title*** and the ***xlabel***, ***ylabel*** and ***zlabel*** commands respectively. See:

```
>> help legend
```

about adding a legend. Also useful is the ***text*** command which adds a text comment at a particular x-y location.

6 Functions and Plots in MatLab

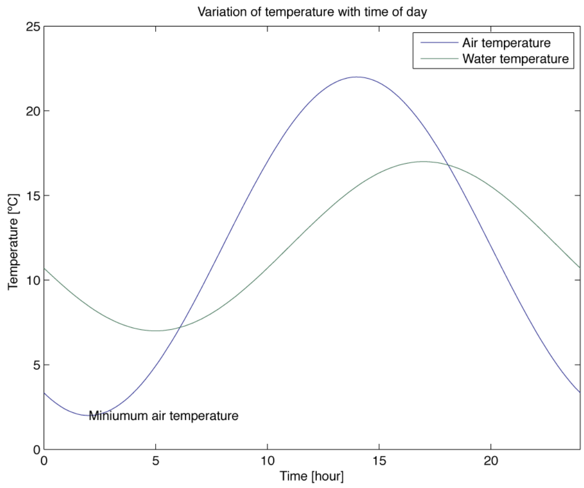
6.4 Simple plots (contd)

```
>> f=@(x)[12+10*sin((x-8)*2*pi/24),12+5*sin((x-11)*2*pi/24)];
>> fplot (f, [0,24,0,25])
>> xlabel ('Time [hour]')
>> ylabel (['Temperature [',176,'C]'])
>> title ('Variation of temperature with time of day')
>> legend ('Air temperature','Water temperature')
>> text (2,2,'Miniumum air temperature')
```

Here is an example of a function returning a row vector with two values, being plotted at the same time using fplot.

Labels, a title, a legend and some text have been added to describe the plot.

The degree symbol has been used in the label on the *y* axis by combining two strings and the integer value (176) of the UNICODE symbol for degrees.



6 Functions and Plots in MatLab

6.5 Multiple plots

By default, MatLab uses only one figure window and each time you create a new plot it removes the old one. To add a new plot to an existing one, type:

```
>> hold on
```

before adding a new plot so that MatLab knows to keep the current plot. Later, to remove the current plot, type:

```
>> hold off
```

before adding a new plot.

Also, you can have more than one figure window. To open a new figure window, use the ***figure*** command:

```
>> figure
```

and, if you have multiple figure windows open, you can select one to make it the active window for new figures by typing, e.g.:

```
>> figure(2)
```

to make Figure 2 active.

7 *m*-Files

7.1 Execution & editing

An ***m***-file is just a text file with commands you would normally type into the MatLab command window, e.g.:

```
v = inline ('[(-(2*n+1)):2:(2*n+1)]*L/(2*n+1)','n','L');
x = v(8,pi);
y = x;
[X,Y] = meshgrid (x,y);
Z = X + i * Y;
Fr = real (cos (Z));
Fi = imag (cos (Z));
figure(1);
mesh (x,y,Fr);
title ('real (cos (Z))');
xlabel ('Real');
ylabel ('Imaginary');
figure(2);
mesh (x,y,Fi);
title ('imag (cos (Z))');
xlabel ('Real');
ylabel ('Imaginary');
```

Execution

This is the file 'icos.m'.

If this file is in your *present working directory* - see the MatLab command ***pwd*** - then you can execute the file from MatLab by typing:

```
>> icos
```

Note: You do not add the suffix '.m' when executing the file.

Navigation

To see the files in your *present working directory*, use the MatLab command ***ls*** (or ***dir***). To change directory, use ***cd***.

Editing ***m***-files are **TEXT** files. Create using the MatLab menu *File → New → M-File*, or use a text editor (***Notepad***, for example, which is in the ***Accessories*** menu in ***Microsoft Windows***) to edit and create ***m***-files.

Do not use a word processor (i.e., **do not use *Microsoft Word*!**) unless you are desperate.

The file must be saved as a text file with suffix '.m'.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

7 *m*-Files

7.2 General comments

Suffix

The suffix (file extension) is '.m', not '.m.m', '.txt' or '.m.txt'.

Viewing file extensions in Windows XP:

- Open **My Computer**, and select **Folder Options** from the **Tools** menu.
- Click on the **View** tab, turn off **Hide MS-DOS file extensions for known file types**, and press OK.

http://www.annoyances.org/exec/show/article01-401

Comments *(i.e., lines beginning with '%' which are ignored by MatLab)*

- add comments - it's a good habit, especially in longer, more complicated ***m***-files

Suppressing Output

- use ';' at the end of lines to prevent MatLab echoing the answer

7 *m*-Files

7.2 General comments (contd)

Exponents

Computers use a special notation for representing ***real*** (or 'floating point') numbers:

```
>> x = +1.23E-6
x =
   1.2300e-06
>> y = -0.45E+3
y =
  -450
>> z = .6789e9
z =
   678900000
```

$x = 1.23 \times 10^{-6}$

$y = -0.45 \times 10^{3}$

$z = 0.6789 \times 10^{9}$

Variable Names

Do not use ***i*** (or ***j*** or ***pi***) as a variable name, or you'll end up with nonsense, e.g.:

```
>> i = 3;
>> (-1)^0.5 - i
ans =
  -3.0000 + 1.0000i
```


7 *m*-Files

7.3 Checklist

1. Does your ***m***-file open in ***Notepad***?
   - Yes? Okay.
   - No? It's *not a text file*, and therefore not an ***m***-file!
2. Can you execute your ***m***-file from MatLab? (If the ***m***-file is called, e.g., 'myMFile.m', can you execute it from MatLab by typing 'myMFile'?)
   - Yes? Okay.
   - No? Clearly *something* is wrong. Either MatLab can't find 'myMFile.m' or it can't understand it.

- Do you have text in the ***m***-file that you would not actually *type* into MatLab? (See below.) *You should also remove any commands which are not relevant!*
  - No? Okay.
  - Yes? *Remove the unnecessary text!*

Previous command that didn't work.

MatLab error message.

Value of previous command.

MatLab command prompt.

```
>> e^(i*pi)
??? Undefined function or variable 'e'.

>> exp(i*pi)

ans =

  -1.0000 + 0.0000i

>>
```

8 Input / Output

8.1 Numerical input / output

When writing programs it is often useful to request information or data from the "user" (i.e., the person using the program).

The ***input*** command is used to write a request for data from the user, and to return the answer. For example, to ask the user what his or her age and height are:

```
age = input ('What is your age? ');
height = input ('What is your height (in metres)? ');
```

Tip: *Leave a space at the end to separate question from answer.*

Important: Put a semi-colon after the ***input*** command to stop the answer echoing.

Here the answers are stored in the variables *age* and *height*, which can be used later:

```
% Calculate average growth rate (AGR) in metres / year
AGR = height / age;
```

Numbers or strings can be written easily to the screen using the ***disp*** command:

```
disp ('Average growth rate (AGR) in metres per year is:')
disp (AGR)
```

This would put the text and the number on different lines, which is a bit messy. Try to put everything on one line. Use the ***num2str*** command to convert the number into a string:

```
disp (['Average growth rate (AGR) in metres per year is ',num2str(AGR),'.'])
```

Important:
1. Square brackets [ ] join strings together.
2. Do not put a space after ***num2str***.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

8 Input / Output

8.2 String input / output

To get a string from the user rather than a number, use the input command with the option 's' specified at the end:

```
address = input ('Please enter your address? ','s');
```

To put an apostrophe in a string, type two apostrophes together, e.g.:

```
friend = input ('What is your friend''s name? ','s');
```

Sometimes commands in MatLab can get very long, especially when using the ***disp*** command. However, you can split commands over multiple lines, putting … to indicate that the command continues on the next line.

```
disp (['Your age is ',num2str(age),', your height is ',num2str(height), ...
       ', ','and your address is ',address,' and your friend''s name is ', ...
       friend,'.'])
```

Important!

When asking the user for input, or when providing information to the user:

1. Be brief, but be informative - say what is necessary, and <u>don't confuse</u>.
2. Be neat and be correct - think about spelling and the use of spaces for clarity.
3. Be polite!

9 Basic Programming

9.1 Loops with 'for'

When a similar or identical action has to be taken multiple times, it is usually a good idea to create a loop. The program will then cycle repeatedly through a set of commands.

The simplest way to create loops is with the command ***for***. For example, to get five numbers from the user and add them up:

```
total = 0;
for k = 1:5
  number = input ('Please enter a number: ');
  total = total + number;
end
disp (['The total is: ', num2str(total)])
```

The start of the loop is marked by ***for***, and the end by ***end***. The number of times the loop is cycled through depends on the number of terms in the loop's defining sequence, in this case 1:5 which has five numbers: [1,2,3,4,5].

Any simple sequence can be used. Even a function returning a sequence is allowed:

```
>> X = @(n) -1 + 2 * [0:n] / n;
>> N = 100; y = 0; for x = X(N); y = y + x^2 * (2 / (N + 1)); end; disp (y)
    0.6800
```

This example integrates x^2 between -1 and 1; the precision increases as the integer N increases.

9 Basic Programming

9.1 Loops with 'for' (contd)

Each time MatLab goes through the loop, the loop counter (m, in the case below) takes the next value in the sequence.

```
>> for m = 1:3:10; disp (['m = ',num2str(m)]); end
m = 1
m = 4
m = 7
m = 10
```

for loops are generally used for accessing elements in a vector:

```
>> A = zeros (1,5) % create a 1x5 matrix (row vector) with zeros in it
A =
     0     0     0     0     0
>> for e = 1:2:5; A(e) = e; end
>> A
A =
     1     0     3     0     5
```

Tip: *note the use of the* ***zeros*** *command to create a vector or matrix of the correct size.*

or matrix:

```
B = zeros (2,3);
for row = 1:2
  for col = 1:3
    B(row,col) = row + col;
  end
end
```

$$A = \begin{pmatrix} 1 & 0 & 3 & 0 & 5 \end{pmatrix}$$

$$B = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \end{pmatrix}$$

9 Basic Programming

9.2 Logical expressions and ‘if’

Where there is the possibility of two or more different program behaviours, depending on the values of certain variables, the ***if*** command is used to select which parts of the program are executed. The expression after each ***if*** (or ***elseif***) evaluates to ***true*** or ***false***.

```
age = input ('How old are you? ');
if (age < 18)       % if age less than 18
    disp ('Sorry. This program is for over-18s only.')
elseif (age >= 65) % if age is 65 or over
    disp ('Really? You look younger than that!')
else
    kids = input ('How many children do you have? ');
    % if age less than 30 and number of children is not zero
    if ((age < 30) && (kids ~= 0))
        disp ('Wow! So soon!')
    else
        disp ('I''ve forgotten what I was going to say...')
    end
end
```

elseif: check condition only if none of the previous ***if***/***elseif*** conditions were ***true***
else: if all else fails, do this…

For comparison, use:
== ‘is equal to’
~= ‘is not equal to’

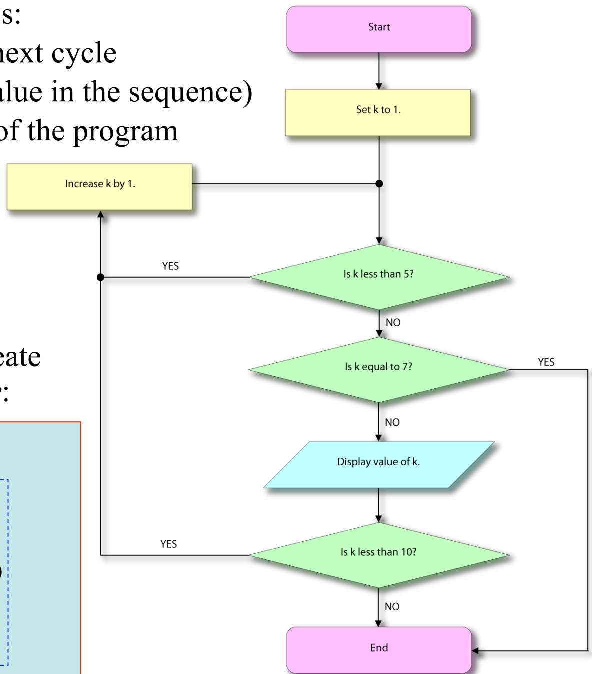
To combine expressions:
&& ‘and’
|| ‘or’

if … ***end***
if … ***else*** … ***end***
if … ***elseif*** … ***else*** … ***end***
if … ***elseif*** … ***elseif*** … ***else*** … ***end***
if … ***elseif*** … ***elseif*** … ***elseif*** … ***else*** … ***end***

9 Basic Programming

9.3 Controlling loops

Two commands are particularly useful inside loops:
continue: return to the start of the loop for the next cycle (in ***for*** loops this jumps to the next value in the sequence)
break: jump out of the loop to the next part of the program
The flow chart illustrates the program below:




```
for k = 1:10
    if (k < 5)
        continue;
    elseif (k == 7)
        break;
    end
    disp (k)
end
```

Another way to create loops is with ***while***:

```
k = 0;
while (k < 10)
    k = k + 1;
    if (k < 5)
        continue;
    elseif (k == 7)
        break;
    end
    disp (k)
end
```

while cycles through the loop as long as the following expression evaluates to ***true***.

while true … end loops forever!

9 Basic Programming

9.4 More on getting user input

Sometimes the program wants to ask the user a question which requires text as an answer, such as a filename, or other name, or perhaps just ‘yes’ or ‘no’. Do this using the command ***input*** again, but add a 2nd argument ‘s’:

```
name = input ('What is your name? ', 's');
```

If Appropriate, Suggest Possible Answers

If you are looking for a particular answer, it is useful to indicate this by suggesting possible answers:

```
likes_kittens = input ('Do you like kittens? (y/n) ', 's');
```

Default Answer

If you have a good idea what the answer will be, it is helpful to provide a default answer, which you should specify in square brackets after the question. (This is a good idea when you have to ask lots of questions.) Then, if the user just presses enter, the value will be returned as the empty matrix, which you can test for using ***isempty***:

```
likes_kittens = input ('Do you like kittens? (y/n) [y] ', 's');
if isempty (likes_kittens)
    likes_kittens = 'y';
end
```

Simple way to check the answer.

```
while 1
  answer = input ('Would you like to exit this loop? (y/n) [y] ', 's');
  if isempty (answer) || (answer == 'y')
    break;
  elseif (answer ~= 'n')
    disp ('I don''t recognise your answer!');
  end
end
```


9 Basic Programming

9.5 Comparing strings

Another way to compare two strings is to use the function ***strcmp***:

```
if strcmp (likes_kittens, 'y')
    disp ('So do I!');
end
```

This function can be used to compare against multiple possibilities, e.g.:

```
>> likes_kittens = 'y';
>> strcmp (likes_kittens, {'y','yes'})
ans =
     1     0
```

i.e., the value is checked against a *cell array* (a type of matrix) of strings, and a matrix of 1s or 0s indicates whether a match is found. To determine whether any of the strings in the cell array matches, use the command ***any***, which determines whether any element in a matrix is non-zero.

To compare strings without worrying about case (i.e., a/A…z/Z) use ***strcmpi***.

Better way to check the answer.

```
while 1
  answer = input ('Would you like to exit this loop? (y/n) [y] ', 's');
  if isempty (answer) || any (strcmpi (answer, {'y','yes'}))
    break;
  elseif any (strcmpi (answer, {'n','no'}))
    continue;
  end
  disp ('I don''t recognise your answer!');
end
```

9 Basic Programming

9.6 Checking numerical input

The command ***isempty*** can also be used with numerical input:

```
v = input ('Pick a number between 1 and 10 [5]: ');
if isempty (v)
    v = 5;
end
```

You can check to see what kind of number or matrix the answer is:

```
if ~isscalar (v) % a scalar is a number or a 1x1 matrix
    disp ('I expected an ordinary number, not a matrix!')
end
if isvector (v) % true if v is a row vector, a column vector - or a number!
    disp ('Hey! That''s a vector!')
end
if v == round (v) % true if v is an integer
    disp ('Great! That''s an integer!')
elseif v == imag (v) % true if v is purely imaginary
    disp ('Wow! That''s an imaginary number!')
elseif v == real (v) % true if v is purely real
    disp ('Excellent! That''s a real number!')
elseif ((v >= 1) && (v <= 10)) % true if v is between 1 and 10 inclusive
    disp ('Thank you!')
end
```


Mechanical Engineering Professional Skills: Introduction to Computing

University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

9 Basic Programming

9.7 Warnings, errors & asserts

Values used in ***m***-files (but defined outside or through user input) may need to be checked. Perhaps you need positive or negative or non-zero numbers, as in the example here, or maybe you want matrices of a particular size or particular characteristics… It is possible to issue warnings and error messages, or even just to give up without comment (although this is not really polite - it's nice to explain what the problem is…).

```
>> myFunction (1,-1,1)
ans =
     1
```

x, *y* and *z* all okay - no problems.

```
>> myFunction (-1,-1,1)
Warning: I don't like negative numbers!
ans =
    -1
> In myFunction at 3
```

oops, *x* is negative! issue warning… (***warning***) … but calculate answer anyway.

```
>> myFunction (1,1,1)
??? Error using ==> myFunction at 6
I really don't like positive numbers!
```

oops, *y* is positive! issue error and stop. (***error***)

```
>> myFunction (1,-1,0)
??? Error using ==> myFunction at 8
Assertion failed.
```

oops, *z* is zero! just stop (i.e., without message). (***assert***)

```
function a = myFunction (x,y,z)
if (x < 0)
  warning ('I don''t like negative numbers!');
end
if (y > 0)
  error ('I really don''t like positive numbers!');
end
assert (z ~= 0);
a = x + y + z;
```

Note: assert *was added to MatLab very recently and may not work in older versions of MatLab.*

10 Function m-Files

10.1 Function declaration and help

m-files can be used to define functions, e.g.:

```
>> help myFunction
  myFunction takes the values x, y and z
  and returns the values
    a = x + y + z
    b = x .* y .* z
>> [a,b] = myFunction (1,2,3)
a =
      6
b =
      6
>> [c,d] = myFunction (2,3,4)
c =
      9
d =
     24
>> [e,f] = myFunction ([1,2],[2,3],[3,4])
e =
      6      9
f =
      6     24
```

(1) (2) (3) (4)

```
function [a,b] = myFunction (x,y,z)
% myFunction takes the values x, y and z
% and returns the values
%   a = x + y + z
%   b = x .* y .* z
a = x + y + z;
b = x .* y .* z;
```

*The **m**-file 'myFunction.m'*

1. Output variables, e.g., the *y* in *y=f(x)*.
2. Function name (file name is this with suffix '.m' added) , e.g., the *f* in *y=f(x)*.
3. Input variables, e.g., the *x* in *y=f(x)*.
4. Comments at the start of the function ***m***-file become the 'help' statement.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

10 Function m-Files

10.2 General comments

Function Name

- The function declaration should be the first (non-empty or non-comment) line in the file.
- Use a *descriptive* function name, e.g., 'deflection(…)' rather than 'f3(…)'
- The file name must match the function name, e.g., 'deflection.m' for 'deflection(…)'

Suppressing Output

- As with normal m-files, use ';' where necessary to suppress output, and use disp where communication with the user is intended.
- ';' is not needed at end of the ***function*** declaration line.

Function Variables

- Do not assign values to the function's input variables. You provide values to the function when you run it!
- There are zero or more input variables.
- There are zero or more output variables.
- Input and output variables can be scalars, vectors, matrices or strings (or even more advanced objects…).

Comments *(i.e., lines beginning with '%' which are ignored by MatLab)*

- function help (the comments immediately before or after the ***function*** declaration) - essential!

10 Function m-Files

10.3 Using functions

Most Important

If you defined, e.g., 'myFunction(…)' in the file 'myFunction.m',

- can you *successfully* use myFunction in MatLab?
- do you get any help when you type 'help myFunction' in MatLab?

Sample solution:

*The **m**-file 'deflection.m'.*

```
function y = deflection(d,L)
% y = deflection(d,L)
% Deflection, y, of the end of a steel
% cantilever beam
%   Young's modulus, E = 209GPa
%   Density, rho = 7800kg/m3
%   Square cross-section, width d
%   Beam length L
E = 209E9;
rho = 7800;
% Second moment of area
I = d^4 / 12;
% Mass per unit length
w = rho * d^2;
% Deflection
y = w * L^4 / (8 * E * I);
```

```
>> help deflection
  y = deflection(d,L)
  Deflection, y, of the end of a steel
  cantilever beam
    Young's modulus, E = 209GPa
    Density, rho = 7800kg/m3
    Square cross-section, width d
    Beam length L
>> deflection (0.005, 1)
ans =
    0.0022
```


10 Function m-Files

10.4 Nested loops with 'for'

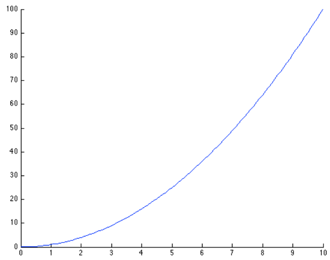
Example of a function ***m***-file with a nested ***for*** loop. The function has two input variables: a vector (X) of *x* values, and an integer (n) which says how many Fourier sequence terms to use in the approximation to the square wave. There is one output variable (Y) which is a vector of *y* values representing the height of the square wave.

```
function Y = square (X, n)
% Y = square (X, n) - square wave generator with n terms in sequence

% Check that n is a scalar integer and greater than or equal to 1:
assert (isscalar (n));
assert ((n == round (n)) && (n >= 1));

% Check that the input matrix/array X is a vector:
assert (isvector (X));

% Find the number of elements:
N = max (size (X));

% Create the output vector (same size as the input vector):
Y = X;

for m = 1:N
  % For each element of X, determine the corresponding value of Y
  Y(m) = 1 / 2;
  for k = 1:n
    Y(m) = Y(m) + 2 * cos (k * X(m)) * sin (k * pi / 2) / (k * pi);
  end;
end;
```

Elements in vectors (both row and column) can be extracted or changed by referring to the position in the vector, i.e.:
- if Z is a row vector, then Z(3) is the same as Z(1,3);
- if Z is a column vector, then Z(3) is the same as Z(3,1).

11 Properties of Plots

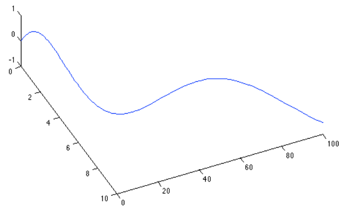
11.1 Line plots



In MatLab, as an alternative to using ***plot***, you can add a line to plots using the ***line*** command:

```
>> X = [0:0.1:10];
>> Y = X.^2;
>> line (X,Y)
```

The above results in the top-right figure. You can also add a line to a 3D plot. The middle-right figure was created like this:

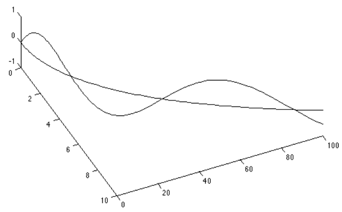


```
>> Z = sin (X);
>> line (X,Y,Z)
>> view (65,70)
```

X, Y and Z can be matrices containing data for multiple lines. At a much more fundamental level in MatLab, single lines are added like this:

```
>> line ('XData',X,'YData',Y)
>> line ('XData',X,'YData',Y,'ZData',Z)
>> view (65,70)
```

where no additional line properties are set (see bottom-right figure). By storing the *handle* (a reference) to the line, i.e.:



```
>> h = line ('XData',X,'YData',Y)
```

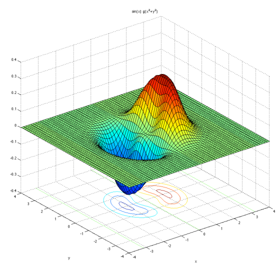
you can use it to change the line's properties later, such as colour and line width, and even the X-Y (or X-Y-Z) data used to draw the line:

```
>> set (h,'Color','r','LineWidth',4)
>> set (h,'ZData',Z.^2)
```


Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

11 Properties of Plots

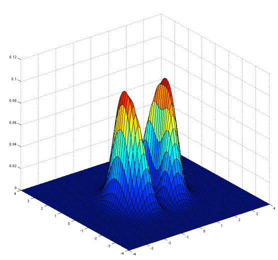
11.2 3D plots



There are a set of commands that are useful for plotting functions in 3D, e.g.: ***ezcontour*** and ***ezsurf***. The top-right figure was created like this:

```
>> g = @(r) r .* exp (-r);
>> f = @(x,y) sin (x) .* g (x.^2 + y.^2);
>> ezsurfc (f, [-4,4,-4,4])
```

In general, 3D plots are based on a vector of X values, a vector of Y values, and a matrix of Z values. An identical plot can be created like this:

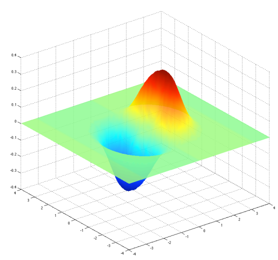


```
>> X = 4 * [-59:2:59] / 59;
>> Y = X;
>> Z = zeros (60,60);
>> for iy = 1:60; Z(iy,:) = f(X,Y(iy)); end
>> surfc (X,Y,Z)
```

As with lines (and plots generally), you can store a *handle* to a surface and change properties later. The middle-right figure was created like this:

```
>> s = surf (X,Y,Z);
>> set (s,'ZData',Z.^2)
```

There are lots of properties associated with surfaces, mostly to do with different colour and lighting models. For example, the bottom-right figure has smoother colour across surface faces, no edges, and is slightly transparent:



```
>> s = surf (X,Y,Z);
>> set (s,'FaceColor','interp','EdgeColor','none')
>> set (s, 'FaceAlpha',0.85)
```

12 Vectors & Matrices

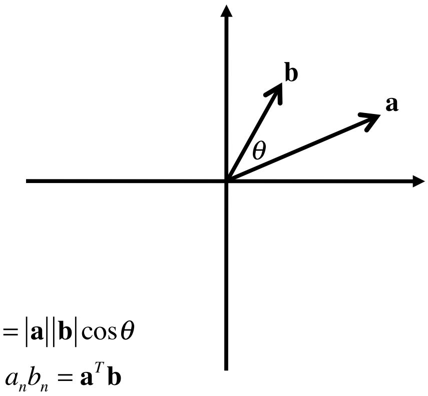
12.1 Vector scalar (or 'dot') product

The scalar product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is written $\mathbf{a}\cdot\mathbf{b}$.

The length of the vector $\mathbf{a}$ is the absolute value, $|\mathbf{a}|$, and similarly $\mathbf{b}$ has length $|\mathbf{b}|$.

In Cartesian coordinates, using Pythagoras theorem, if $\mathbf{a}$ and $\mathbf{b}$ are n-dimensional column vectors:

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$



then the lengths (magnitudes) of the vectors are:

$$|\mathbf{a}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \qquad |\mathbf{b}| = \sqrt{b_1^2 + b_2^2 + \cdots + b_n^2}$$

and if the angle between $\mathbf{a}$ and $\mathbf{b}$ is θ, the scalar product is: $\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$

but also the scalar product is defined as: $\mathbf{a}\cdot\mathbf{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n = \mathbf{a}^T\mathbf{b}$

where $\mathbf{a}^T$ is the transpose of $\mathbf{a}$, i.e.: $\mathbf{a}^T = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix}$

In MatLab, the scalar product can be calculated using the command ***dot***:

```
>> dot(a,b)
```

<u>The transpose of a vector or matrix</u> in MatLab is calculated using an apostrophe:

```
>> a' * b
```

The length (magnitude) of a vector can be calculated using the command ***norm***.

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

12 Vectors & Matrices

12.2 Matrix multiplication

Let $\mathbf{A}$ be an $n \times m$ matrix and $\mathbf{B}$ an $m \times p$ matrix, then the product:

$$\mathbf{C} = \mathbf{AB}$$

is an $n \times p$ matrix:

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ \vdots & \vdots & & \vdots \\ \boxed{a_{i1} \quad a_{i2} \quad \cdots \quad a_{im}} & & & \\ \vdots & \vdots & & \ddots \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} b_{11} & \cdots & b_{1j} & \cdots \\ b_{21} & \cdots & b_{2j} & \cdots \\ \vdots & & \vdots & \\ b_{m1} & \cdots & b_{mj} & \ddots \end{pmatrix}$$

where:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{im}b_{mj}$$

Example - a **permutation** matrix rearranges the elements in a vector:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \\ x \end{pmatrix}$$

12 Vectors & Matrices

12.3 Matrix powers and inverse

Let $\mathbf{A}$ be an $n \times n$ square matrix, then:

$$\mathbf{A}^0 = \mathbf{I} \quad \mathbf{A}^1 = \mathbf{A} \quad \mathbf{A}^2 = \mathbf{AA} \quad \mathbf{A}^3 = \mathbf{AAA} \quad \text{etc.}$$

where $\mathbf{I}$ is the $n \times n$ identity matrix.

Negative powers are also possible:

$$\mathbf{A}^{-2} = \left(\mathbf{A}^{-1}\right)^2 \quad \mathbf{A}^{-3} = \left(\mathbf{A}^{-1}\right)^3 \quad \text{etc.}$$

where $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$, and is defined by:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \mathbf{A}\mathbf{A}^{-1}$$

and this behaves like a normal power of $\mathbf{A}$, e.g.: $\mathbf{A}^{-1}\mathbf{A}^3 = \mathbf{A}^2$

In MatLab, matrix powers are calculated in the usual way, i.e., don't use element-wise methods!

```
>> A^2
```

The inverse of a matrix $\mathbf{A}$ can be found using the command ***inv***, or by raising it to the power -1:

```
>> A^(-1)
>> inv (A)
```


12 Vectors & Matrices

12.4 Simultaneous equations

Simultaneous equations, e.g.:

$$3x + 4y = 5$$
$$7x + 12y = 13$$

can be written in matrix form:

$$\begin{pmatrix} 3 & 4 \\ 7 & 12 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 13 \end{pmatrix}$$

In general, this can be written as the problem:

$$\mathbf{Ax} = \mathbf{b}$$

where $\mathbf{A}$ is a known $n \times n$ matrix and $\mathbf{b}$ is a known n-dimensional vector.

The problem is to find x.

One way is to find the inverse of $\mathbf{A}$ and multiply $\mathbf{b}$:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

Computationally, it is better (faster and more accurate) to use Gaussian elimination.

In MatLab this is done using a backslash:

```
>> x=A\b
```

12 Vectors & Matrices

12.5 Eigenvalues & eigenvectors

The eigenvalues and eigenvectors of a matrix are solutions of the equation:

$$\mathbf{Mv} - \lambda\mathbf{v} = \mathbf{0}$$

(not including the case where **v**=**0**). For instance, the principal stresses are the eigenvalues of the stress matrix.

In MatLab, you can find eigenvalues and eigenvectors using the ***eig*** command, e.g.:

```
>> M = [1,2,3;4,5,6;7,8,9];
>> [V,L] = eig (M)
V =
   -0.2320   -0.7858    0.4082
   -0.5253   -0.0868   -0.8165
   -0.8187    0.6123    0.4082
L =
   16.1168         0         0
         0   -1.1168         0
         0         0   -0.0000
```

This is an example of a MatLab function which returns two *matrix* values; here they are saved as **V** and **L**. The matrix **V** contains the eigenvectors in columns, and the matrix **L** contains the eigenvalues along the diagonal elements. The eigenvector in column *n* of **V** corresponds to the eigenvalue in row *n*, column *n* of **L**, so for *n*=1:

```
>> M * V(:,1) - L(1,1) * V(:,1)
ans =
   1.0e-14 *
    0.5329
   -0.1776
   -0.1776
```

i.e., very small!

13 Ordinary Differential Equations

13.1 First order ODEs

First-order ordinary differential equations (ODEs) have the general form:

$$\frac{d}{dt}y = f(t,y)$$

where y can be a single variable of time, t, or a vector of variables of time. For example:

$$\frac{d}{dt}y = t \quad \Rightarrow \quad y = \tfrac{1}{2}t^2 + c$$

$$\frac{d}{dt}y = y \quad \Rightarrow \quad \ln(y) = t + c$$

$$\frac{d}{dt}y = y + t \quad \Rightarrow \quad y = ce^t - t - 1$$ (use an integrating factor)

Note: Although the ODE is expressed in terms of $f(t,y)$, the function does not have to depend on t and y (or, if y is a vector, on all components of y).

In MatLab, you can solve the ODE using the ***ode23*** command (there are other ODE solver commands also), e.g.:

$$\frac{d}{dt}y = \sin(y+t) \quad \Rightarrow$$

```
>> f = @(t,y) sin (y + t);
>> ode23 (f,[0 20],0);
```

The 2nd function argument, `[0 20]`, tells MatLab to work out the function starting at time t=0 and finishing at t=20. The 3rd function argument, `0`, tells MatLab to start at y=0 when t=0.

To get the data of function values (`Y`) and corresponding times (`T`) instead of a plot, type:

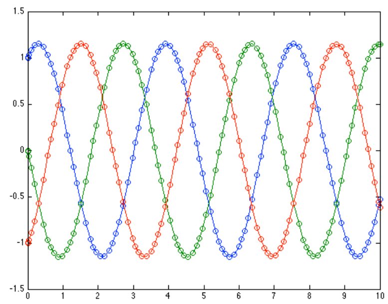
```
>> [T,Y] = ode23 (f,[0 20],0);
```

13 Ordinary Differential Equations

13.2 Vector ODEs

A system of first-order ordinary differential equations (ODEs) may be expressed as a vector:

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} y_2 - y_3 \\ y_3 - y_1 \\ y_1 - y_2 \end{pmatrix}$$



In MatLab, solving again using the ***ode23*** command:

```
>> f = @(t,y) [y(2)-y(3);y(3)-y(1);y(1)-y(2)];
>> ode23 (f,[0 10],[1;0;-1]);
```

produces the figure on the right.

To get the data of function values (`Y`) and corresponding times (`T`) instead of a plot, type:

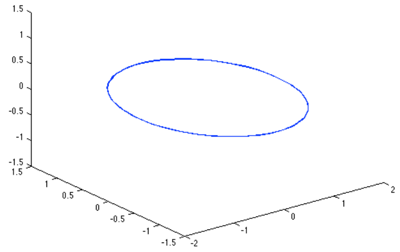
```
>> [T,Y] = ode23 (f,[0 10],0);
```

This creates a vector `T` of times between 0 and 10 inclusive (the time interval requested), and a matrix `Y` with three columns (one column for each element in the vector returned by the function `f`) of *y*-values corresponding to the time-values in `T`.

For example,

```
>> plot3 (Y(:,1),Y(:,2),Y(:,3))
```

produces the figure below-right.



13 Ordinary Differential Equations

13.3 Second order ODEs

Second-order ordinary differential equations (ODEs) involve higher-order derivatives, e.g.:

$$\frac{d^2}{dt^2}y + \frac{d}{dt}y + y = f(t,y)$$

where y can be a single variable of time, t, or a vector of variables of time. For example:

$$\frac{d^2}{dt^2}y + 2\frac{d}{dt}y + y = t \quad \Rightarrow \quad y = Ae^{-t} + Bte^{-t} + t - 2$$

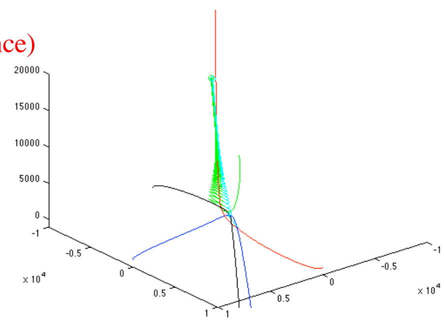
$$\frac{d^2}{dt^2}y + k^2 y = \sin(kt) \quad \Rightarrow \quad y = A\sin(kt) + B\cos(kt) - \frac{t}{2k}\cos(kt)$$

$$\frac{d^2}{dt^2}\mathbf{r} + \frac{GM\mathbf{r}}{r^3} = 0 \quad \Rightarrow \quad \text{(orbit equation)}$$

$$\frac{d^2}{dt^2}\mathbf{x} + \mu\frac{d}{dt}\mathbf{x} = \mathbf{g} \quad \Rightarrow \quad \text{(projectile equation with resistance)}$$

The last two are equations of motion (***r*** and ***x*** are position vectors), relating acceleration to position and velocity.

Practical 6 has an example of several large moons passing close to one another, with motions described by the orbit equation given above. The figure on the right shows the result of including all five bodies.



13 Ordinary Differential Equations

13.3 Second order ODEs (contd)

To solve higher-order ordinary differential equations, you can add new variables, e.g.:

$$\frac{d^2}{dt^2}y + \frac{d}{dt}y + y = f(t,y) \quad \Rightarrow \quad \begin{cases} \frac{d}{dt}y_1 = y_2 \\ \frac{d}{dt}y_2 + y_2 + y_1 = f(t,y_1) \end{cases}$$

This turns a higher-order problem into a first-order vector ODE problem.

In other words, y started out as a single variable but has been turned into a vector:

$$\frac{d}{dt}\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ f(t,y_1) - y_1 - y_2 \end{pmatrix}$$

It can often be useful to think about first and second order ODEs as equations of motion, even if the system being described has nothing to with physical motion.

In this (one-dimensional) case, let the position be s, the velocity v and the acceleration a:

$$a + v + s = f(t,s) \quad \Rightarrow \quad \begin{cases} \frac{d}{dt}s = v \\ \frac{d}{dt}v + v + s = f(t,s) \end{cases} \quad \Rightarrow \quad \begin{pmatrix} v \\ a \end{pmatrix} = \frac{d}{dt}\begin{pmatrix} s \\ v \end{pmatrix} = \begin{pmatrix} v \\ f(t,s) - s - v \end{pmatrix}$$

Mechanical Engineering Professional Skills: Introduction to Computing
University of Newcastle upon Tyne, School of Mechanical & Systems Engineering

13 Ordinary Differential Equations

13.3 Second order ODEs (contd)

As another example, motion in 2D with resistance:

$$\frac{d^2}{dt^2}\begin{pmatrix} x \\ y \end{pmatrix} + \mu\frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ -g \end{pmatrix}$$

may be rearranged:

$$\left.\begin{array}{l} \frac{d}{dt}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \\ \frac{d}{dt}\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} -\mu v_x \\ -\mu v_y - g \end{pmatrix} \end{array}\right\} \quad \Leftrightarrow \quad \frac{d}{dt}\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ -\mu v_x \\ -\mu v_y - g \end{pmatrix}$$

Here a second-order vector ODE has been rearranged to give a first-order vector ODE, but the vector has more elements.