Today's Topics

- Using MATLAB as a calculator.
- Constants
- Arithmetic operators and expressions
- Elementary functions, help, `format`
- Variables, reserved words,
- Introduction to script files
- Input
- Simple graphs
- Workspace, who, clear

Reading

Introduction to Applied and Computational Mathematics, sections 1.1, 1.2.1, 1.2.4, 1.2.7, 1.2.9
Introduction to Matlab 6, sections 2.1, 2.2.1(end), 2.2.2(part), 2.2.3, 2.3.1, 3.1, 3.6

MATLAB is a computer language/application designed for scientific computation.

Starting

Click on the Windows start symbol at the bottom left of the screen, click Programs from the menu that appears, then move to click Matlab. This opens the Command Window where you can use MATLAB like a calculator.

Prompt

When MATLAB is ready to accept a new instruction it shows a prompt `>>`.

Enter

Press the enter key when you have typed in your instruction. It does not matter if the cursor is not at the end of the line.

Continuation of a Line

If your instruction is too long to conveniently fit on a single line you can break it after an operator by typing three dots then using return and continuing your instruction.

Constants

Some constants are:

```
5       -206            0.001       8.1468
pi      3.1638E-3       5.24E12
```

Arithmetic Operators
The arithmetic operators in *decreasing* order of precedence are:

arithmetic operator	operation performed
^	raise to a power
/	division
*	multiplication
-	subtraction
+	addition

You often need to use brackets to make sure that the operations are carried out in the correct order.

```
>>12/(1+3)
ans=3
```

Arithmetic Expressions

```
>>2*3
ans=6
>>12/5
ans=2.400
>>2^3
ans=8
>>10-(1+3)
ans=6
```

Write in MATLAB

1. $3^2 + 5$
2. 3^{2+5}
3. $\frac{60}{2+3}$
4. $\frac{60+3}{2}$
5. $-2 \times 5 \times -7$
6. $\frac{12-3}{5+1}$
7. $\frac{1}{2^4}$
8. $(\frac{3}{4})^4$
9. 5π

Elementary Functions
Most of the usual mathematical functions are implemented in MATLAB.

```
>> cos(0)
>> 6*sin(pi/2)
>> exp(1)
>> log(exp(3))
>> sqrt(9)
```

Note:

- MATLAB uses radians, not degrees.
- The `log` is the natural log (often labelled `ln` on calculators). The log base 10 is `log10`.

Help

If you know the name of the function that you want help on use the help command. For example,

```
>> help log
```

will give you some notes on the function `log`.

`format` Command

MATLAB always calculates internally to 16 significant figures. You can use `format` to control the number of decimal places in the output.

`format short`	4 decimal places (the default)
`format long`	14 decimal places
`format short e`	scientific notation with 4 d.p.
`format long e`	scientific notation with 14 d.p.

Example

```
>>5.55744525*2
ans=
     11.1149
>>format long e
>>5.55744525*2
ans=
     1.111489050000000e+01
```

Variables and Memory

A variable is a labelled piece of computer memory.

```
>> s=5.6
```

The number 5.6 is stored in the computer memory and labelled.

- Matlab is case sensitive. That means that the upper case and lower case of a letter are different. For example, the variable called `s` is different from the variable called `S`.
- Reserved words are variable names it is best not to use. This is often because they are names of built-in functions. Check using `help`.
- A semicolon after a statement suppresses the output. For example, the statement `x=2*3` will set the value of variable `x` to 6 and also show the result on the screen. However, `x=2*3;` still sets the value of `x` to 6, but does not show the result of the multiplication on the screen.

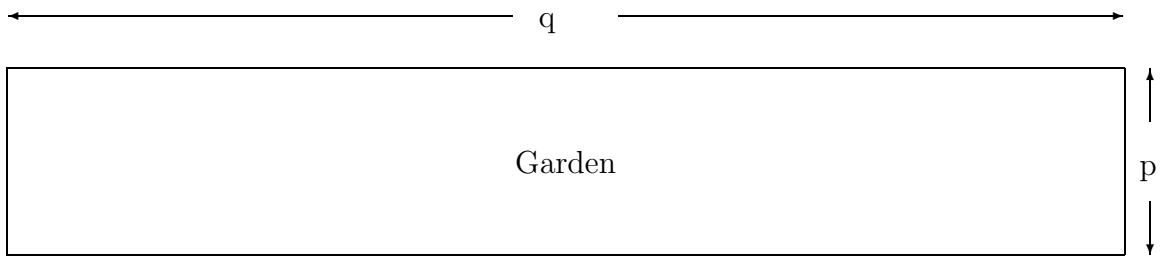
Copying Previous Input Lines

The key ↑ will copy the previous command into the current line. Use it twice to get the one before etc.

Example
Find the area of a circle with radius 5cm.

Now let radius be 7cm.

Example
A piece of wire netting to put a fence around a garden is 10 metres long.



If $p = 2$, what is q and what is the area enclosed?

But first, can we use "length" as our variable name for the length of wire netting? Is it a reserved word?

Script Files
A *program* is a sequence of statements that are performed in order. They are stored in a *script file.*

A *file* is the basic unit that the computer stores on mass storage (such as hard disk or floppy disk). Files can store pictures, or programs (such as MATLAB script files) or other data (such as text files).

To *run* a program either type the name of its file in the Command Window or use the Debug menu in its window and choose Save and Run.

Examples

1. Write a script file to set a value for `d`, a distance in km, and calculate how long it takes to drive that distance at 80 kph.

2. Write a script file to set a value for `r`, the radius of a circle, and calculate the area of the circle and its circumference.

`input` statement

In the example above we put values directly into Matlab. The `input` statement is another way to enter values for a MATLAB program. It includes a string which is used as a prompt so the user knows what to enter.

Example

```
x=input('Enter a positive number ');
```

When this statement is executed, the string
`Enter a positive number` will appear in the command window. The user then types in a value, and this value is assigned to the variable `x`. Note that just this statement will not check that the number entered is positive.

A Simple Graph

```
>> x=linspace(0,2*pi,50);
>> plot(x,cos(x))
```

The first line sets up a variable called `x` which is 50 equally spaced numbers starting with 0 and ending with 2π. The second statement plots the values in `x` along the horizontal axis and the cos of each value up the vertical axis.

Adding Labels

```
>> xlabel('x')
>> ylabel('cos x')
>> title('A graph')
```

Multiple Graphs

```
>> plot(x,cos(x),x,sin(x))
```

Alternatively

```
>> x=linspace(0,2*pi,50);
>> y=x+2*cos(x);
>> plot(x,y)
```

The above examples are done in the Command Window. It is often more convenient to put the statements in a script file.

Componentwise (or element-by-element) operators

When the arithmetic operations `^`, `*`, and `/` are used with vectors (as in the above plot examples), then we must use the corresponding componentwise operators which are `.^`, `.*`, and `./`. For example, to draw a graph of $y = x^2$ we use the following statements.

```
>> x=linspace(-2,2);
>> y=x.^2;
>> plot(x,y)
>> title('My plot of y=x^2')
```

Workspace

- The workspace is the memory and variables that are in use. This can be saved.
- `who` lists the variable names in current use.
- `whos` lists the variables and how much memory they use.
- `clear X` clears the variable `X`.
- `clear all` clears all the variables.

Today's Topics

- Strings, `disp` and `sprintf`
- Relations
- Boolean expressions
- `if` statements - flow control
- Vectors

Reading
Introduction to Applied and Computational Mathematics, sections 1.2.2, 1.2.3, 1.2.5-1, 1.2.6(part).

Introduction to Matlab 6, parts of sections 2.2.1 and 2.2.2, 3.5.1, 3.5.2, 3.5.3, 3.5.4.

Revision Exercises

1. Write Matlab expressions to calculate:
   - $4^2 + 3\cos\frac{\pi}{2}$
   - $\sqrt{1-\pi}$
   - $5e^3$
2. Write a Matlab script file to:
   - Prompt the user to enter the value of the variable `r`; and
   - Calculate the circumference of a circle of radius `r`.
3. Write a Matlab script file to plot the graph
$$y = x^3$$
from $x = -2$ to $x = 2$. Label the axes and put a title on the graph.

Well Depth Formula

$$depth = (\frac{g}{k}) \times (time + \frac{e^{-k \times time} - 1}{k}).$$

g: acceleration due to gravity, $g \approx 9.81$.
k: friction constant, $k \approx 0.05$ for a stone (roughly).

Write Matlab to set the values of g and k and then calculate the depth for a given time.

Character Strings
Variables can be used to store numeric values (eg -6.5e3) or strings (eg 'Hello').
Note that the single quote marks are used for strings in MATLAB.

```
>> n=5;
>> y='Course MATHS 162';
```

The first character of string `y` above is `y(1)`.
The 7th to 13th characters inclusive are `y(7:13)`.

`length(y)` gives the length of string `y`.

Note that because `length` is the name of a function, it is not a good idea to use it as a variable name.

Concatenation
This is following one string by another.

```
>> x='I am studying'
x =
I am studying
>> y='Course MATHS 162'
y =
Course MATHS 162
>>[x y]
ans =
I am studyingCourse MATHS 162
```

`disp` command
`disp(x)` will display `x`, whether it is a number or a string.
If `x` and `y` are as above, then use

```
>> disp([x y])
```

Examples

`disp` command	output
`disp(5.7)`	`5.7000`
`disp('Hello')`	`Hello`
`disp(15*4)`	`60`
`disp('15*4')`	`15*4`

A script file example

```
greeting='Hello.';
question='What is your name?';
disp([greeting,' ',question])
```

Output

```
Hello. What is your name?
```

The `;` after a statement suppresses the output.

Well Depth Formula in a Script File

```
% Program to calculate depth of well from
% time a stone takes to reach the bottom.
% Use g=9.81, k=0.05
g=9.81;
k=0.05;
time=2;
depth=g*(time+(exp(-k*time)-1)/k)/k;
disp('Depth of well is ')
disp(depth)
disp('metres')
```

Output

```
Depth of well is
   18.9820
metres
```

Better Looking Output

Use function `sprintf` to produce strings which include numerical results.
For example, in the well depth program

```
disp(sprintf('Depth is %4.2f metres.',depth))
```

The output would then be

```
Depth is 18.98 metres.
```

The `%4.2f` indicates that a value is to be put in here. It will have 2 decimal places and there will be 4 digits or signs overall. There are other formats such as `g` for significant figures, `d` for integers, `s` for strings and `\n` for a new line. See reference section 1.6 of the coursebook. Also, in *Introduction to Matlab*, see section 2.2.2 for the similar function `fprintf`.

Example

Write a script file which prompts the user to enter a number and then displays the square of that number.

```
n=input('Enter a number ');
disp(sprintf('The square is %4.0f.',n^2))
```

Example

Write a script file which prompts the user to enter their name, where name is always in the form initial then space then family name (e.g. 'A Heard'), and then displays the name in the original form and then with family name then comma then initial.

Relations
Relations are comparisons which are either true or false.

In MATLAB: false is represented by 0
true is represented by 1
(or any other non-zero number)

Examples

relation	result	relation	result
`6>3`	true (or 1)	`3>6`	false (or 0)
`7==4`	false (or 0)	`7~=4`	true (or 1)

Boolean Expressions
Relations may be combined using logical operators. Use brackets to make sure of the order of evaluation.

`~`	logical NOT	changes to opposite truth value
`|`	logical OR	at least one relation is true
`&`	logical AND	both relations are true

Examples

expression	result
`(6>3) & (4>2)`	true (or 1)
`7==4 | 2<6`	true (or 1)
`~(2>4)`	true (or 1)
`(~(2>4)) | 2>1`	true (or 1)
`(2<0)&((1>-1)|(4==2))`	false(or 0)

Quick Quiz

```
 ( (5>0) | (5<10)) &(2==3)
```

```
(~(5>0)) | ((1+3==4) & (5<12))
```

Introduction to `if` statements

Use an `if` statement to execute some commands only when appropriate. Use `end` to show the end of the statements to be executed. For example,

```
n=input('Enter the number now ')
if n<0
    disp('This number is negative')
else
    disp(sprintf('Square root is %4.2f.',sqrt(n)))
end
```

Example
Write a script file to prompt the user to input a value for `d`, a distance in km, and calculate how long it takes to drive that distance at 80 kph. If it is greater than 10 hours, display a message that it will take more than one day.

More on IF statements

```
if (x>2)|(x<0)
   y=2;
elseif x<1
   y=1;
else
   y=0;
end
```

Some Types of `if` statements

- `if` *boolean expression*
  some statements
  `end`
- `if` *boolean expression*
  some statements
  `else`
  some more statements
  `end`
- `if` *boolean expression*
  some statements
  `elseif` *boolean expression*
  some more statements
  `else`
  some more statements
  `end`

Examples

1. No more than 10 drinks may be served to one person. If more than 10 are ordered, the number is reduced to 10 and a message printed out. Write a script file to prompt the user to enter the number of drinks ordered and display the appropriate messages.

   Sample Output

   ```
   You have ordered 20 drinks. Reduced to 10.
   You have ordered 5 drinks.
   ```

2. Write a script file to prompt the user to enter an integer, and then display whether the integer is zero, positive or negative.

3. Grades are to be assigned as follows:
   A 80% - 100%
   B 65% - 79%
   C 50% - 64%.
   Write a script file to prompt the user to input a mark and display the appropriate grade. If the user enters a number greater than 100 or less than zero, display a message that the mark is invalid.

Vectors
A vector is a one dimensional array.
Some examples are:
`[1,3,2,9,5], [-1 3 8 0 -2 1],`
`1:100,`
`1:0.1:10,`
`linspace(0,10,6)`

We can find how many elements in a vector by using the function `length`.
`length([1,3,2,9,5])` is 5.
`length(1:0.1:10)` is 91.

A vector of length 1 (ie just a number) is called a scalar. The vector `[]` is the null vector. It has length zero.

We can refer to the element of a vector by using its index. For example, if we use
`v=[1,3,2,9,5];`
then we can use `v(3)` to refer to the third element, which is 2.
We can use a vector for the index in order to refer to more than one element. For the vector above, `v(2:4)` refers to the second to fourth elements inclusive of `v`. This will be `[3,2,9]`.

Examples
Let `w=[9,2,10,-5,0,-2,4,1]`.

Write down:

1. `w(5)`
2. `2:3:8`
3. `1:2:8`
4. `w(2:3:8)`
5. `w(8:-3:2)`
6. `w(length(w):-1:1)`

Input statements, Vectors and Strings
An input statement can prompt the user to enter a vector or a string. The user must enter a vector in square brackets and a string in single quote marks. When the computer displays them, it does not use square brackets or quotes.

Today's Topics

- `for` statement
- `rand` and `ceil` functions
- `while` statement
- Functions

Reading
Introduction to Applied and Computational Mathematics, section 1.2.5-2, 1.2.5-3, 1.2.8.
Introduction to Matlab 6, section 3.5.6, 3.6.

Revision Examples

- We started with 15 kg of apples. Write a script file to:
  - Prompt the user to enter how many kg have been used today;
  - Display how many were used, and how many are now left.

  Remember that you cannot use more than you have left!

- Let `r=[-4,5,0,2,-1,0]`
  Write down:

  1. `r(2)`
  2. `1:2:7`
  3. `3:3:7`
  4. `r(1:2:7)`
  5. `r(6:-2:2)`
  6. `r(length(r):-1:1)`

`for` statement

```
for variable=vector
        some statements
end
```

Example
Write a script file to calculate the squares of the integers up to 20.

```
% A script file to print out the squares
% from 1 to 20.
for i=1:20
 disp(sprintf('%2.0f squared is %4.0f.',...
  i,i^2))
end
```

Now change the script file to calculate the squares of the odd numbers up to 21.

```
% A script file to print out the odd
% squares to 21.
for i=1:2:21
 disp(sprintf('%2.0f squared is %4.0f.',...
  i,i^2))
end
```

Now change it to display them backwards

```
% A script file to print out the odd
% squares backwards from 21.
for i=21:-2:1
 disp(sprintf('%2.0f squared is %4.0f.',...
  i,i^2))
end
```

Example
Write a script file to calculate the sum of the integers up to 100.

```
total=0;
for n=1:100
   total=total+n;
end
disp(sprintf('Sum up to 100 is %3.0f.',...
 total))
```

More Examples

1. Write a script file to prompt the user to enter a vector, and then display every second element in the vector.
   Sample Output

   ```
   Enter a vector: [3,-3,2,8,4]
   3  2  4
   ```

2. Write a script file to prompt the user to enter a vector, and then display it in reverse order.

3. Write a script file to plot a graph of $\sin x$ from $-\pi$ to π using:
   (a) 100 values of x;
   (b) increments of $\pi/10$ in x.

4. Write a script file to prompt the user to input a vector, then calculate and display how many of the components, starting with the second, are greater than the preceding one.

`rand` function
The function `rand` produces a random number between 0 and 1.

`ceil` function
This function rounds to the next integer towards infinity. For example, `ceil(8.2)=9` and `ceil(-8.2)=-8`. This function can be used with `rand` to produce random integers. When we are simulating the tossing of dice, for example, we need random integers from 1 to 6. We can use `ceil(6*rand)`.

`while` statement
The `while` statement is used to execute a loop while a given boolean expression is true.

The syntax is

```
while  boolean expression
       some statements
end
```

Example
Write a script file which prompts the user for a number M and then displays the integers and their squares while the square is less than M.

```
% Calculates squares while square is
% less than M
M=input('Enter the maximum number ');
k=1;
while k^2 < M
 disp(sprintf('%2.0f squared is %4.0f.',...
 k,k^2))
 k=k+1;
end
```

Use of `for` and `while`
The `for` statement is used when we know in advance how many times we want to execute a loop. For example, to evaluate

$$\sum_{i=1}^{n} i^2$$

we know that the loop must be repeated n times even if we do not know the value of n before we run the program. With the `while` statement we can test for the desired condition each time we execute the loop. For example, to keep repeating some process until the user enters a zero.

Example
Write a script file to repeatedly prompt the user for a student mark and display the student mark along with the grade, until a negative mark is entered.Grades are:
A 80% - 100%, B 65% - 79%, C 50% - 64%.

We will need a `while` loop with a boolean expression to recognise the negative mark.

Example
Write a script file to keep a count of how many mobile phones are left. First prompt the user to enter the number available at the start. As they are sold, display each sale and update the number available. When all phones have been sold, display a message that all are sold.

Sample output:

```
Enter the number of phones available 6
Enter the number of phones to be sold 3
3 phones sold
Enter the number of phones to be sold 1
1 phones sold
Enter the number of phones to be sold 2
2 phones sold
All sold
```

Example
The following script file asks users to guess values of x for which $x - cos(x)$ will be zero. The variable `tol` is used to indicate how far from zero is acceptable. Variable `max_guess` is used to set the maximum number allowed. (Why?)

```
disp('* When is x-cos(x) is zero? *')
disp(' ')
tol=1e-2;  max_guess=8;
y= ______ ;
n=0;
while abs(y)>tol & ______ < max_guess
   x=input('Enter your guess ');
   y=x-cos(x);
   disp(sprintf('Value is %4.2f',y))
   n=n+1;
end
if abs(y)<=______
  disp(sprintf('%4.2f is close enough.',x))
  disp(sprintf('Value is %4.2f',y))
else
  disp('Your guess limit is reached.')
  disp('Bad luck!')
end
```

Functions
We can use 'built-in' functions like `sin`, `cos` and `sqrt`. But we can also write our own functions.

Function Examples
1. Can of soda (book section 1.2.8)

Input:
Height of can
Radius of can
Price of can

Output:
Volume of can/soda
Price per litre of soda

2. Large jaffas (Practice Exercises)

Input:
Diameter of a jaffa
Number of jaffas

Output:
Volume of chocolate

Syntax of a Function Header
`function [` output `]=` funct-name `(`input`)`

If there is only one output variable, you omit the [ ]. There may be no output variables.

For example,
`function [y,z]= f(x)`

The statements that follow the function header will calculate the output variables in terms of the input variables.

The function header and statements are typed into a file. Each function is in a separate file.

Example

```
function [vol,p_l]=can(h,r,p_c)
%returns a vector containing the volume
%vol (litres) and price/litre p_l of can.
%Inputs are height h (cm), radius r (cm),
%price p_c (dollars).
vol=pi*h*r^2/1000;
p_l=p_c/vol;
```

This will be typed into a file and saved as `can`. *It is important that the function name is the same as the file name.* We can use the function by typing `[v,p]=can(12,3,1)`.

Use of Variables
A function communicates with the rest of the workspace only through the input and output variables.
Any other variables that you use inside the function are known only inside the function. They are not defined in the Command Window workspace for example.

Examples for Functions

1. Write a function file to calculate the volume of chocolate required for the jaffas.

2. Write a function file with a student mark as the input variable, and the grade (as a character string) as the output variable.

3. Write a function file that returns the sum of the positive components and the sum of the negative components of the input vector.

4. Write a script file for the large jaffas that:
   (a) prompts the user to input the diameter
   (b) prompts the user to input the number
   (c) calls the function in Example 1
   (d) displays the amount of chocolate required.

Today's Topics

- 2D Arrays / Matrices
- Vectorisation

Reading

Introduction to Applied and Computational Mathematics, sections 1.2.6, 1.3.2.

Introduction to Matlab 6, parts of sections 2.2, 4.1.

Revision Examples

1. Write a Matlab script file which prompts the user to enter a number and displays a message showing whether the number is less than or equal to 2, or greater than 2.

2. Write a Matlab function file with input parameter K and output parameter s where

$$s = \sum_{j=1}^{K} j^2.$$

3. Write a Matlab script file which prompts a user to enter a negative number, and keeps prompting until a negative number is entered.

Revision

```
function [a,b]=first(w)
% w is a vector
i=1;
while w(i)<=0
    i=i+1;
end
a=i;
b=w(i:length(w));
```

- What is the result of `[r,s]=first([-2 9 -3 8 9])`?

- What does the function `first` do?

Introduction to 2-D Arrays - Matrices

Suppose we have some students' results from an assignment marked out of 10, a test out of 50 and an exam out of 100. In the final mark, the assignment is worth 20%, the test 30% and the exam 50%.

Name	Assignment	Test	Exam
John	8	36	81
Ying	7	44	75
Jenny	5	39	77
Alice	9	29	62

How do we calculate the final marks?
Multiply the assignment mark by

Multiply the test mark by

Multiply the exam mark by then add them.

Writing the students' marks as a matrix and the factors above as a column vector, we get

$$\begin{pmatrix} 8 & 36 & 81 \\ 7 & 44 & 75 \\ 5 & 39 & 77 \\ 9 & 29 & 62 \end{pmatrix} \begin{pmatrix} 2.0 \\ 0.6 \\ 0.5 \end{pmatrix}$$

2-D Arrays - Matrices

We have studied 1-D arrays (vectors) which are like a list of numbers. The 2-D arrays go both across and down, like the matrices we study in mathematics.

Example

The matrix

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

can be written in Matlab by

`A=[1 -5 -4; -2 7 0; 5 0 -1]` or

`A=[1, -5 ,-4; -2 ,7, 0; 5, 0, -1]`

The numbers in a row are separated by spaces or commas (as in a vector), and the end of the row is marked by a `;`.
If we want to refer to the element in the 3rd row and the 1st column then we use `A(3,1)`.
Note that the row number is the first index and the column number is the second index.
If a matrix is multiplied by a vector in Matlab, then the result is the same as the usual matrix/vector multiplication.

If `v=[1; 4; -2]` find `A*v`.

$$Av = \begin{pmatrix} 1 & -5 & -4 \\ -2 & 7 & 0 \\ 5 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 4 \\ -2 \end{pmatrix}$$

Exercise
Write Matlab statements to calculate

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

Picking out rows and columns
If we want to pick out just the first row of matrix `A`, then we use `A(1,:)`. It picks out the components which have their first index as 1.
To pick out the second column use `A(:,2)`. This gives us a vector.

Examples

```
C=[25,14,0,29,21;19,12,10,28,18];
```

Find `C(1,:)` and `C(:,2)`.

Some Vector Functions
`length` returns the length of a vector (i.e. how many elements or components it has).
`min` and `max` returns minimum and maximum elements respectively of the vector.
`sum` returns sum of the elements in vector.
`prod` returns the product of the elements in the vector.
`transpose` returns the transpose of the vector i.e. a row vector becomes a column vector and vice versa. Also use `v'`.

Examples

```
v=[1 -3 56 -3 2 7];
```

```
max(v)=                 min(v)=
min(abs(v))=            sum(v)=
sum(abs(v))=            prod(v)=
v'=
```

Some Vector Operations

You can add and subtract vectors, and multiply vectors by scalars in the usual way.

If you multiply vectors together, i.e. `v*w`, then it is matrix multiplication that you do. This means that you can only multiply a row vector by a column vector (and the result is a scalar), or multiply a column vector by a row vector (and the result is a matrix). All other multiplications will give an error.

```
v=[1,3,5,7]; w=[2;4;5;6];
```

Find:

```
v*w                    w*v
```

Example

Write a function with input parameters of two row or two column vectors and output parameter the dot product of the vectors, $\sum_{i=1}^{n} v_i w_i$.

Element by Element Operations

Two arrays can be multiplied together element by element. For example, for two vectors, `v` and `w`, `v.*w` is a vector of the products of the corresponding elements of `v` and `w`. The two arrays multiplied will need to have the same size and shape. *Also called componentwise operations.*

We can also take powers of arrays element by element. Each element of the array is raised to that power to form the result.

Examples

If `v=[-5 0 2 1]; w=[9 1 5 2];` then

`v+w` is `[4 1 7 3]`.
`v-w` is `[-14 -1 -3 -1]`.
`v.*w` is `[-45 0 10 2]`.
`v.^2` is `[25 0 4 1]`.
`v.+w` and `v.-w` will give an error. Adding or subtracting arrays is done element by element anyway, so we do not need these operations.
`v*w` will give an error because we cannot do matrix multiplication with two row vectors.
`v^2` will give an error because we cannot square a row vector using matrix multiplication.

Plotting

When we plot a graph, we make a vector of x values for the horizontal axis and then a vector of y values that correspond to these x values. For example, suppose we want a rough graph of $y = x^2$ from $x = -5$ to $x = 5$. We could use `x=[-5,-4,-3,-2,-1,0,1,2,3,4,5]`. To make the `y` vector we need to square *each* of these `x` values. This means we want element by element squaring, not to multiply the vector `x` by itself in matrix multiplication. So we write `y=x.^2`.

MATLAB makes plotting the vector of values in `x` against the vector of values in `y` very easy: `plot(x,y)`. It is also easy to plot multiple sets of data on the same axes, for instance:

```
plot(x,x.^2,x,sin(x))
```

Most of the built in functions in Matlab, such as `sin` and `cos`, can be used with vectors or matrices. The function is applied to each element. If we write our own functions that we might want to use for plotting we should make them suitable for vector input parameters.

Examples

1. Write a function file with input parameter matrix `A` and output parameter `v` which is the first row of `A` as a *column* vector.

2. Write a function file to find the maximum element in a particular row of a matrix. Use the matrix and the number of the row as input parameters, and the set the output parameter to be the maximum element in that row.

3. Write a script file to
   - prompt the user to enter an integer `n`
   - make a square matrix with `n` rows and columns of random numbers between 0 and 1
   - use the above function to find the maximum element in the last row
   - display the value of this element to 5 decimal places.

`whos`
This is the command to return the number and size of the variables in use.

Boolean Expressions with Vectors

We can use Boolean expressions to pick out elements of vectors which satisfy Boolean expressions.
For example, if `r=[7,-4,0,6,-2,1]`,
then `r>0` is `[1,0,0,1,0,1]`. It shows the positions of the elements of `r` which are positive.

We can find the values of the positive elements of `r` by using `r(r>0)`. For the above `r`, it will be `[7,6,1]`.

Example

Write a function with input parameter n to return (i.e. have as output parameter) a vector of numbers from throwing a six sided die n times. The function `ceil` rounds to next integer towards infinity. Use `rand(1,n)`.

Write a script file, calling the above function, to prompt the user to input the number of throws of the die and to display the results of the throws and how many throws were of numbers greater than 3.

Some Matrix Functions

`size` returns the dimensions of a matrix (i.e. how many rows and columns it has).

`sum` returns the sum over rows or columns.
`sum(A,1)` is a vector of the sums of the columns.
`sum(A,2)` is a vector of the sums of the rows.

`prod` as for `sum` but with products.

`transpose` returns the transpose of the matrix, i.e. a rows become columns and vice versa.

Some Matrix Operations

You can add and subtract matrices, and multiply matrices by scalars in the usual way.
If you multiply two matrices together then it is matrix multiplication and the number of columns of the first matrix must equal the number of rows of the second.
If you want the power of a matrix then the matrix must square, i.e. the number of rows equals the number of columns.

Matrices can also be multiplied together, or powers may be evaluated, element by element as for vectors.

Example
Write a function with input parameter a matrix `K`. The output parameter will be a column vector of the sum of the absolute values of the elements in each row.

Boolean Expressions with Matrices

We can also use Boolean expressions with matrices.
For example, if `C=[0,4,-3,1;9,-2,0,-1]`
`C>2` is `[0,1,0,0;1,0,0,0]`.
Also `sum(C>2,1)` is `[1,1,0,0]` and
`sum(C>2,2)` is `[1;1]`.

Example
Suppose we have a matrix of marks for 3 assignments for a group of students, such as

$$\begin{pmatrix} 25 & 14 & 0 & 29 & 21 & 18 & 27 & 30 \\ 19 & 12 & 10 & 28 & 18 & 27 & 30 & 26 \\ 26 & 18 & 16 & 25 & 20 & 29 & 29 & 28 \end{pmatrix}$$

Number the students 1, 2, ... 8.
How will we enter this matrix into Matlab?

Write Matlab statements to find:

1. A row vector of the total marks for each student.

2. A column vector for the highest marks in each assignment.

3. How many students got more than 20 marks for Assignment 1.

4. How many marks (over all assignments) were below 12.

5. A column vector of assignment means.

Today's Topics

- More on Vectors and Matrices
- Systems of Linear Equations
- Larger Programs

Revision Examples

- Write the matrix B and the vector z in Matlab.

$$B = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \qquad z = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

- Evaluate Bz.

- Write a Matlab command to evaluate Bz.

- `w=[1,-4,-2,3,7,-1,0,2]`.
  Find `length(w)`, `w(1:4)` and `w(3:length(w))`.

- Write a Matlab function to evaluate the function

$$f(x) = \frac{2x+1}{x+3}e^{-x},$$

  and a script file to plot a graph of f using 50 values of x from 0 to 20.

Systems of Linear Equations

A system of linear equations can be written in matrix form, and then be solved by Matlab. For example,

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 32 \\ 77 \\ 68 \end{pmatrix}.$$

If A is the matrix on the left, x is the vector on the left and b is the vector on the right, then we can write this as

$$Ax = b.$$

The solution will be

$$x = A^{-1}b.$$

This can be written in Matlab as `A\b`.
Similarly the system

$$xA = b$$

has the solution

$$x = bA^{-1}$$

which is written in Matlab as `b/A`.

Example

Write Matlab to solve the above system of linear equations.

Example
Four friends want to buy some stationery. The prices and their orders are

Lecture Pad	\$3.20
Pen	\$2.50
Ruler	\$1.40

	Pad	Pen	Ruler
Victoria	3	2	1
Jun	4	3	1
Anne	2	1	2
Henare	3	1	0

We can write these this data as a matrix, N, of the numbers wanted, and a vector, p, of prices.

$$N = \begin{bmatrix} 3 & 2 & 1 \\ 4 & 3 & 1 \\ 2 & 1 & 2 \\ 3 & 1 & 0 \end{bmatrix}, \qquad p = \begin{bmatrix} 3.20 \\ 2.50 \\ 1.40 \end{bmatrix}.$$

Let $q = Np$, (matrix vector multiplication).

$$q = \begin{bmatrix} 3 \times 3.20 + 2 \times 2.50 + 1 \times 1.40 \\ 4 \times 3.20 + 3 \times 2.50 + 1 \times 1.40 \\ 2 \times 3.20 + 1 \times 2.50 + 2 \times 1.40 \\ 3 \times 3.20 + 1 \times 2.50 + 0 \times 1.40 \end{bmatrix} = \begin{bmatrix} 16.00 \\ 21.70 \\ 11.70 \\ 12.10 \end{bmatrix}.$$

What does q represent?

Example
Write a Matlab script file that

- sets the value of the vector p of stationery prices above
- prompts the user to enter a matrix, N, of the orders (numbers wanted by each person)
- displays how many orders (people) there were
- calculates the total amount each person owes
- numbering the people $1, 2, 3, \ldots$, use a `for` loop to display for each person the number of that person and the amount they owe.

Example
After the friends had ordered their stationery, they found the prices had changed. All we know is the amounts they owe are

Victoria	$16.30
Jun	$22.00
Anne	$12.20
Henare	$12.30

Write a Matlab script file to set the matrix of the orders and the vector of the amounts owed, and to use that information to find what the prices have changed to.

Larger Programs
When we want to write a large program we should break it up into modules. This is illustrated in the coursebook in section 1.3.1 for the disk cutting problem. The problem is divided into smaller problems before we start to write the Matlab code.

Example
Write a Matlab script file to do the calculations when people order their stationary. We want to find out how much people owe and how much stock is left.

1. **Specify the problem**
   We will be given prices, initial amount of stock and matrices of orders. We will calculate the amount owed and how much stock is left.
2. **Design the algorithm**
   - *Module 1* will control the whole process.
   - *Module 2* Set up the prices in a vector and the initial amounts of stock also in a vector.
   - *Module 3* Get the user to input the matrix for an order.
   - *Module 4* Calculate the amounts for the order and the total amount using matrices and vectors.
   - *Module 5* Update the amounts of stock in the stock vector.
3. **Implement and comment the algorithm**
   - *Module 1* will be a script file which calls the other modules.
   - *Module 2* will be a function with no input parameters but two output vectors, *prices* and *stock*.
   - *Module 3* will be a function with no input parameters but one output matrix *order*.
   - *Module 4* will be a function with input parameters for prices and the order and output parameter for the amounts owed.
   - *Module 5* will be a function with the stock vector as an input parameter and output vector the updated stock vector.

Now write the functions and the script file.

4. **Debug and test the files**
   One way is to leave off all the semicolons so you can see the intermediate results.

   Do you need to put in checks in case the user does not enter suitable vectors and matrices? What about when the stock is finished?