Simple Calculations with MATLAB

Lecture 1

Dr. Hao Zhu

Beijing-Dublin International College,
Beijing University of Technology

2019 Autumn

Outline

Outline

Why MATLAB?

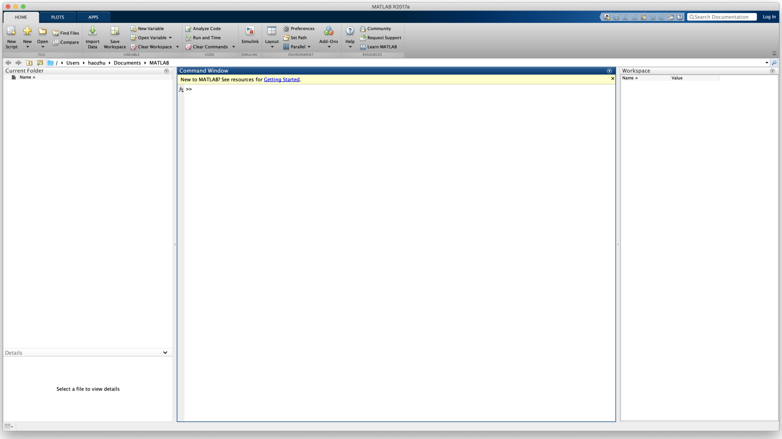
- This module provides an introduction to the numerical methods that are typically encountered and used in science and technology.
- The package MATLAB provides an environment in which students and researchers can learn to programme and explore the structure of the numerical methods.

Why Numerical Method?

- There are so many problems which simply do not have analytical solutions, or those whose exact solution is beyond our current state of knowledge.
- There are also many problems which are too long (or tedious) to solve by hand.
- For problems mentioned above, we can exploit numerical analysis and use a computer to solve the resulting equations.

Introduction to MATLAB

- MATLAB(**MAT**rix **LAB**oratory) is an incredibly powerful tool for numerical computation.
- MATLAB uses an interpreter to try to understand what you type.



New to MATLAB? See resources for Getting Started.

```
>>
```

Outline

Scalar Quantities and Variables

We begin with the basic equations and variables. Try entering the commands as they are given below.

```
>> a = 3
a =
       3
>> b = 4;
```

The MATLAB prompt is denoted by >>

The second one ends with a semicolon(;) which means executing the command but suppressing any output. The commands can be read as

```
set a equal to 3
set b equal to 4 (and suppress output)
```

Scalar Quantities and Variables

It is impossible to have the command of the form `7 = x` (`set 7 equal to x`), whereas we could have `x = 7` (`set x equal to 7`).

This variables can be used again, for example

```
>> a = 3;
>> b = a+1;
>> x = a+b;
```

- The first line sets the variable a to be equal to 3, the semicolon instructing MATLAB to execute the command but to suppress the output.
- The second line sets `b` to be equal to a plus one, namely 4: again the semicolon suppresses output.
- The third line sets `x` to be `a+b` which is 7(again output is suppressed).

Scalar Quantities and Variables

In MATLAB, its operations fall into two basic groups: *unary* and *binary*, the former operating on one quantity and the latter on two. For instance typing 3*4 generates

```
>> 3*4
ans =
    12
```

MATLAB uses the variable `ans` to store the result of our calculation and it can be used in the subsequent commands. For instance the command `ans*3` will generate the result 36 (and now the variable `ans` will have the value 36).

We could also have used the commands `a = 3; b = 4; x = a*b` which can be typed on one line.

Example 1.1

Try entering the following commands into MATLAB, but before you do so try to work out what output you would expect.

```
>> 3*5*6
>> z1 = 34;
>> z2 = 17;
>> z3 = -8;
>> z1/z2
>> z1-z3
>> z2+z3-z1
```

Hopefully you should get the answer, 90, 2, 42 and -25.

Example 1.2

Here we give an example of the simple use of bracket:

```
>> format rat
>> a = 2; b = 3; c = 4;
>> a*(b+c)
>> a*b+c
>> a/b+c
>> a/(b+c)
>> format
```

In this example you should get the answers, 14, 10, 14/3 and 2/7. Hopefully this gives you some idea that brackets make MATLAB perform those calculations first. (The command format rat has been used to force the results to be shown as rationals, the final command format reverts to the default, which happens to be format short.)

Rules for Naming of Variables

The rules for naming variables in MATLAB can be summarised as follows:

- Variable names must start with a letter and can be up to 31 characters long. The last characters can be numbers, letters or underscores. Many variable names should be forbidden like `a*b` and `a.b`. The rules for naming variables also hold for naming files.
- Variable names in MATLAB are case sensitive, so that `a` and `A` are two different objects.
- It is good to employ meaningful variable names, which makes variable names more informative.
- Variables names should not coincide with a predefined MATLAB command or with any user-defined function.

Precedence: The Order Calculations Performed

Precedence is the order in which commands are executed.

Consider the mathematical expression $a(b + c)$ which you might read as ‘a times b plus c‘ which would appear to translate to the MATLAB command `a*b+c`. Hopefully you can see that this actually is equal to $ab + c$. The correct MATLAB command for $a(b + c)$ is `a*(b+c)`.

Example 1.3

Determine the value of the expression $a(b+c(c+d))a$*, where* $a = 2$*,* $b = 3$*,* $c = 4$ *and* $d = 3$*.*

Although this is a relatively simple example it is worth constructing the MATLAB statement to evaluate the expression:

```
>> a = 2; b = 3; c = 4; d = -3;
>> a*(b+c*(c+d))*a
```

This gives the answer 124. The commands each end with semicolons; we have chosen to place all four commands on one line: however they could just as easily be placed on separate lines.

Example 1.4

Evaluate the MATLAB expressions by hand and then check answers with MATLAB.

```
(1+3)*(2-3)/3*4
(2-3*(4-3))*4/5
```

Recall that the operations of division and multiplication take precedence over addition and subtraction. The expressions are given by

$$\texttt{(1+3)*(2-3)/3*4} = \frac{4\times(-1)}{3}4 = -\frac{16}{3}$$
$$\texttt{(2-3*(4-3))*4/5} = (2-3\times 1)\frac{4}{5} = -\frac{4}{5}$$

We can use the command format rat to force MATLAB to output the results as rational numbers (that is, fractions).

Example 1.5

Use MATLAB to calculate the expression, where $a = 3$, $b = 5$ *and* $c = -3$.

$$b - \frac{a}{b + \frac{b+a}{ca}}$$

The code for this purpose is:

```
a = 3;
b = 5;
c = -3;
x = b-a/(b+(b+a)/(c*a));
```

with the solution being contained in the variable **x**.

Example 1.6

Enter the numbers $x = 45 \times 10^9$ *and* $y = 0.0000003123$ *using the exponent - mantissa (*$mantissa \times 10^{exponent}$*) syntax. Calculate the quantity* xy *using MATLAB and by hand.*

This is accomplished using the code

```
x = 45e9;
y = 3.123e-7;
xy = x*y;
```

Notice that here we have used a variable name **xy** *which should not be confused with the mathematical expression* xy *(that is* $x \times y$*).*

Mathematical Function

- Arithmetic functions: `+`, `-`, `/` and `*`.
- Trigonometric functions: `sin`, `cos` and `tan` and their inverses `asin`, `acos` and `atan`. This functions take an argument in radians. The syntax of the command is `sin(x)` rather than `sin x`.
- Exponential functions: `exp`, `log`, `log10` and `^`. The default in MATLAB for a logarithm is the natural logarithm `lnx` (`log`). The final command takes two arguments (and hence is a binary operation) so that `a^b` gives a^b.
- Other functions: `round(x)`(round a number to the nearest integer), `ceil(x)`(round a number up to the nearest integer), `floor(x)`(round a number down to the nearest integer), `fix(x)`(round a number to the nearest integer towards zero), `rem(x,y)`(The remainder left after division), `mod(x,y)`(The signed remainder left after division), `abs(x)`, `sign(x)`, `factor(x)`(The prime factors of x that give multiple outputs).

Example 1.7

Calculate the expressions: $\sin 60^\circ$ *(and the same quantity squared),* $\exp(\ln(4))$, $\cos 45^\circ - \sin 45^\circ$, $\ln\exp(2+\cos\pi)$ *and* $\tan 30^\circ/(\tan\pi/4+\tan\pi/3)$.
We shall give the MATLAB code used for the calculation together with the results:

```
>> x = sin(60/108*pi);
x =
    0.8660

>> y = x^2;
y =
    0.7500

>> exp(log(4))
ans =
    4
```

Example 1.7(Cont'd)

```
>> z = 45/180*pi; cos(z)-sin(z)
ans =
    1.1102e-16

>> log(exp(2+cos(pi)))
ans =
    1

>> tan(30/180*pi)/(tan(pi/4)+tan(pi/3))
ans =
    0.2113
```

Notice that zero has been approximated by `1.1102e-16`, which reflects the accuracy to which this calculation is performed.

Outline

Format: The Way in Which Numbers Appear

- All numbers saved in MATLAB are in double precision, and the formats of outputs would have more options.
- There are other options for `format` which you can see by typing `help format`. The default option is `format short` (which can be reverted back to by simply typing `format`).
- The above options are `short` - 5 digits; `long` - 15 digits; `rat` - try to represent the answer as a rational.

Example 1.8

Consider the following code:

```
s = [1/2 1/3 pi sqrt(2)];
format short; s
format long; s
format rat; s
format ; s
```

this generates the output:

Example 1.8(Cont'd)

```
>> format short; s
s =
    0.5000    0.3333    3.1416    1.4142

>> format long; s
s =
    0.500000000000000    0.333333333333333    3.141592653589793
    1.414213562373095

>> format rat; s
s =
    1/2    1/3    355/113    1393/985

>> format ; s
s =
    0.5000    0.3333    3.1416    1.4142
```

Outline

Initialising Vector Objects

We shall start with simple objects and construct these by the colon symbol:

```
r = 1:5;
```

This sets the variable `r` to be equal to the vector `[1 2 3 4 5]`. This is a row vector, which we can see by typing `size(r)` (which returns `[1 5]`, indicating that `r` has one row and five columns).

Initialising Vector Objects

This simple way of constructing a vector `r = a:b` creates a vector `r` which runs from `a` to `b` in steps of one. We can change the step by using the syntax `r = a:h:b`, which creates the vector `r` running from `a` to `b` in steps of `h`, for instance

```
r = 1:2:5;
s = 1:0.5:3.5;
```

gives `r = [1 3 5]` and `s = [1 1.5 2 2.5 3 3.5]`. We note that if the interval `b-a` is not exactly divisible by `h`, then the loop will run up until it exceeds `b`, for instance `t = 1:2:6` gives `t = [1 3 5]`.

We can also initiate vectors by typing the individual entries; this is especially useful if the data is irregular, for instance `t = [14 20 27 10]`.

Initialising Vector Objects

There are many other ways of setting up vectors. This is the command `linspace`(linearly spaced): this has two syntaxes

```
s = linspace(0,1);
t = linspace(0,1,10);
```

Here `s` is set up as a row vector which runs from zero to one and has **one hundred elements** and `t` again runs from zero to one but now has **ten elements**.

Manipulating Vectors and Dot Arithmetic

This allows us to manipulate vectors in an element-wise fashion rather than treating them as mathematical objects.

Let's see example of multiplying a vector by a number:

```
>> a = [1 2 3];
>> 2*a;
ans =
    2 4 6
```

Manipulating Vectors and Dot Arithmetic

Suppose now we try to multiply a vector by a vector, as in

```
>> a = [1 2 3];
>> b = [4 5 6];
>> a*b
???  Error using ==> *
Inner matrix dimensions must agree.
```

An error message appears because both `a` and `b` are row vectors and therefore cannot be multiplied together, which means their dimensions are not matched.

Manipulating Vectors and Dot Arithmetic

What if we want to multiply the elements of vector a by the elements of vector b in an element by element sense? We can achieve this in MATLAB by using dot arithmetic:

```
>> a = [1 2 3];
>> b = [4 5 6];
>> a.*b
ans =
    4    10    18
```

The answer shows that MATLAB has returned a vector containing the elements $[a_1b_1, a_2b_2, a_3b_3]$. The . indicates to MATLAB to perform the operation term by term and the * indicates we require a multiplication.

Manipulating Vectors and Dot Arithmetic

We can also do a term by term division with

```
>> a = [1 2 3];
>> b = [4 5 6];
>> a./b
ans =
    0.2500    0.4000    0.5000
```

The result is, as we would expect,

$$\left[\frac{a_1}{b_1}, \frac{a_2}{b_2}, \frac{a_3}{b_3}\right]$$

Example 1.9

We shall create two vectors running from 1 to 6 and from 6 to 1 and then demonstrate the use of the dot arithmetical operations:

```
s = 1:6;
t = 6:-1:1;
s+t
s-t
s.*t
s./t
s^2
1./s
s/2
s+1
```

Example 1.9(Cont'd)

This produces the output:

```
>> s+t
ans =
    7    7    7    7    7    7

>> s-t
ans =
    -5    -3    -1    1    3    5

>> s.*t
ans =
    6    10    12    12    10    6
```

Example 1.9(Cont'd)

```
>> s./t
ans =
    0.1667    0.4000    0.7500    1.3333    2.5000    6.0000

>> s^2
ans =
    1    4    9    16    25    36

>> 1./s
ans =
    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667
```

Example 1.9(Cont'd)

```
>> s/2
ans =
    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

>> s+1
ans =
    2    3    4    5    6    7
```

We note that in order for these operations to be viable the vectors need to be of the same size (unless one of them is a scalar - as in the last three examples).

Outline

Setting Up Mathematical Functions, Example 1.10

In this section, we will discuss the ways in which you can set up the input to the function.

Set up a vector **x** *which contains the values from 0 to 1 in steps of 1/10. This can be done in a variety of ways:*

```
% Firstly just list all the values:
x = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0];

% Use the colon construction
x = 0:0.1:1.0;

% Or use the command linspace
x = linspace(0,1,11);
```

Setting Up Mathematical Functions

If we want to set up a mathematical function $y = x^2$, we would type `x^2` but this will generate the error message. This is because it performs the mathematical operation $\mathbf{x} \times \mathbf{x}$ which is not possible.

```
???  Error using ==> ^
Matrix must be square.
```

Instead we need to use `y = x.^2` or `y = x.*x` which gives

```
>> y = x.^2
y =
  Column 1 through 7
    0    0.0100    0.0400    0.0900    0.1600    0.2500    0.3600
  Column 8 through 11
0.4900    0.6400    0.8100    1.0000
```

Example 1.11

Construct the function $y = \frac{x^2}{x^3+1}$ *for values of* x *from 1 to 2 in steps of 0.01. Here we give the solution:*

```
x = 1:0.01:2;
f = x.^2;
g = x.^3+1;
y = f./g
```

We could have combined the last three lines into the single expression `y = x.^2./(x.^3+1);`.

It would be highly recommended to use intermediate functions when constructing complicated functions.

Outline

Some MATLAB Specific Commands

In this section, we will introduce a couple of commands which can be used to make calculations.

- `polyval`: This command takes two inputs, namely the coefficients of a polynomial and the values at which you want to evaluate it.
- `plot`: This command plots the results of this calculation.
- `roots`: The input to the routine is simply these coefficients and the output is the roots of the polynomial.

Example 1.12

Evaluate the cubic $y = x^3 + 3x^2 - x - 1$ *at the points* $x = (1, 2, 3, 4, 5, 6)$. *We provide the solution to this example as a commented code:*

```
% Firstly set up the points at which the polynomial
% is to be evaluated
x = 1:6;

% Enter the coefficients of the cubic (note that
% these are entered starting with the
% coefficient of the highest power first
c = [1 3 -1 -1];

% Now perform the evaluation using polyval
y = polyval(c,x)
```

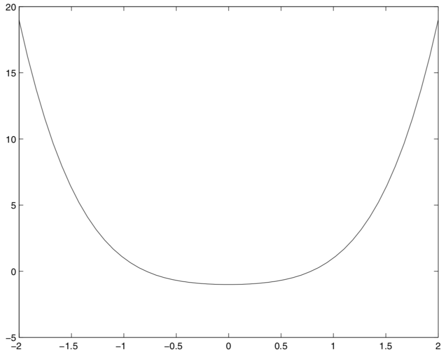
For c, starting with the highest power and zeros are included in the sequence.

Example 1.13

Plot the polynomial $y = x^4 + x^2 - 1$ *between* $x = -2$ *and* $x = 2$ *(using fifty points).*

```
x = linspace(-2,2,50);
c = [1 0 1 0 -1];
y = polyval(c,x);
plot(x,y)
```

This produces the output:



Example 1.14

Find the roots of the polynomial $y = x^3 - 3x^2 + 2x$ *using the command roots.*

```
c = [1 -3 2 0];
r = roots(c)
```

This returns the answers as 0, 2 and 1.

The converse command also exists, which is poly. This takes the roots and generates the coefficients of the polynomial having those roots (the coefficient of the highest term is unity).

Looking at Variables and Their Sizes, Example 1.15

To list the variables which are currently defined we can use the command `whos`.

What does the following code give?

```
clear all
a = linspace(0,1,20);
b = 0:0.3:5;
c = 1.;
whos
```

The output gives below:

Example 1.15(Cont'd)

```
Name    Size    Bytes    Class

a      1x20      160    double array
b      1x17      136    double array
c      1x1        8      double array

Grand total is 38 elements using 304 bytes
```

The clear all command is to remove all previously defined variables. The command length(a) give the size of variable(the answer is 20). The command size(a) will give two dimensions of the array(the answer is [1 20]).

Outline

Accessing Element of Arrays

Let us start by considering a simple array `x = 0:0.1:1.;`.

The elements of this array can be called by using the format `x(1)` through to `x(11)`. The number in the bracket is the index and refers to which value of `x` we require. A convenient mathematical notation for this would be x_j where $j = 1, ..., 11$. This programming notation should not be confused with $x(j)$; that is x is a function of j.

Let us consider the following illustrative example:

Example 1.16

Construct the function $f(x) = x^2 + 2$ on the set of points $x = 0$ to 2 in steps of 0.1 and give the value of $f(x)$ at $x = 0, x = 1$ and $x = 2$. The code to construct the function is:

```
x = 0:0.1:2;
f = x.^2+2;

% Function at x=0
f(1)
% Function at x=1
f(11)
% Function at x=1
f(21)
```

Note that the three points are **NOT** *f(0), f(1) and f(2)!*

Example 1.17

We now show how to extract various parts of the array **x**.

```
x = linspace(0,1,10);
y = x(1:end);         % Whole of x
y = x(1:end/2);       % First half
y = x(2:2:end);       % Even indices only
y = x(2:end-1);       % All but the last one and the first one
```

The expression `end` is very useful at this point, since it can be used to refer to the final element within an array.

Important point: In MATLAB, `f(j)` the value of `j` refers to the index within the array rather than the function `f(.)` evaluated at the value `j`!

Outline

Tasks

Task 1.1 *Calculate the values of the following expressions(try to use help command or help tan)*

$p(x) = x^2 + 3x + 1$ at $x = 1.3$,

$y(x) = \sin(x)$ at $x = 30^\circ$,

$f(x) = \tan^{-1}(x)$ at $x = 1$,

$g(x) = sin\left(\cos^{-1}(x)\right)$ at $x = \frac{\sqrt{3}}{2}$.

Tasks

Task 1.2 *Calculate the value of the function* $y(x) = |x| \sin x^2$ *for values of* $x = \pi/3$ *and* $\pi/6$ *(use the MATLAB command abs(x) to calculate* $|x|$*).*

Tasks

Task 1.3 *Calculate the quantities* $\sin(\pi/2), \cos(\pi/3), \tan 60^\circ$ *and* $\ln(x + \sqrt{x^2+1})$*, where* $x = 1/2$ *and* $x = 1$.
Calculate the expression $x/\left((x^2+1)\sin x\right)$ *where* $x = \pi/4$ *and* $x = \pi/2$. *(If you are getting strange answers in the form of rationals you may well have left the format as rat, so go back to the default by typing format).*

Tasks

Task 1.4 *Explore the use of the functions round, ceil, floor and fix for the values* $x = 0.3, x = 1/3, x = 0.5, x = 1/2, x = 1.65$ *and* $x = -1.34$.

Tasks

Task 1.5 *Compare the MATLAB functions rem(x,y) and mod(x,y) for a variety of values of x and y (try $x = 3, 4, 5$ and $y = 3, 4, -4, 6$). (Details of the commands can be found using the help feature).*

Tasks

Task 1.6 *Evaluate the functions for x from 1 to 2 in steps of 0.1*

1. $y = x^3 + 3x^2 + 1$
2. $y(x) = \sin x^2$
3. $y(x) = (\sin x)^2$
4. $f(x) = \sin 2x + x \cos 4x$
5. $y(x) = x/(x^2 + 1)$
6. $f(x) = \cos /(1 + \sin x)$
7. $y = 1/x + x^3/(x^4 + 5x \sin x)$

Tasks

Task 1.7 *Evaluate the function*

$$y = \frac{x}{x + \frac{1}{x^2}}$$

for $x = 3$ *to* $x = 5$ *in steps of 0.01.*

Tasks

Task 1.8 *Evaluate the function*

$$y = \frac{1}{x^3} + \frac{1}{x^2} + \frac{3}{x}$$

for $x = -2$ *to* $x = -1$ *in steps of 0.1.*

Tasks

Task 1.9(D) *The following code is supposed to evaluate the function*

$$f(x) = \frac{x^2 \cos \pi x}{(x^3+1)(x+2)}$$

for $x \in [0,1]$ (using 200 steps). Correct the code and check this by evaluating the function at $x = 1$ using $f(200)$ which should be $-1/6$.

```
x = linspace(0,1);
clear all
g = x^3+1;
H = x+2;
z = x.^2;
y = cos xpi;
f = y*z/g*h
```

Tasks

Task 1.10(D) *Debug the code which is supposed to plot the polynomial* $x^4 - 1$ *between* $x = -2$ *and* $x = 2$ *using 20 points.*

```
x = -2:0.1:2;
c = [1 0 0 -1];
y = polyval(c,x);
plot(y,x)
```

Tasks

Task 1.11(D) *Debug the code which is supposed to set up the function* $f(x) = x^3 \cos(x+1)$ *on the grid* $x = 0$ *to* 3 *in steps of 0.1 and give the value of the function at* $x = 2$ *and* $x = 3$.

```
x = linspace(0,3);
f = x^3.*cos x+1;
% x = 2
f(2)
% x = 3
f(End)
```