The McGraw-Hill Companies

PowerPoint to accompany

Introduction to MATLAB for Engineers, Third Edition

William J. Palm III

Chapter 3

Functions and Files

Common mathematical functions:

Exponential	
`exp(x)`	Exponential; e^{x}
`sqrt(x)`	Square root; $\sqrt{x}$
Logarithmic	
`log(x)`	Natural logarithm; $\ln x$
`log10(x)`	Common (base 10) logarithm; $\log x = \log_{10} x$

(continued…)

Some common mathematical functions (continued)

Complex

`abs(x)`	Absolute value.
`angle(x)`	Angle of a complex number.
`conj(x)`	Complex conjugate.
`imag(x)`	Imaginary part of a complex number.
`real(x)`	Real part of a complex number.

(continued…)

Some common mathematical functions (continued)

Numeric

`ceil(x)`	Round to nearest integer toward ∞.
`fix(x)`	Round to nearest integer toward zero.
`floor(x)`	Round to nearest integer toward -∞.
`round(x)`	Round toward nearest integer.
`sign(x)`	Signum function:
	+1 if $x > 0$; 0 if $x = 0$; -1 if $x < 0$.

Operations on Arrays

MATLAB will treat a variable as an array automatically. For example, to compute the square roots of 5, 7, and 15, type

```
>>x = [5,7,15];
>>y = sqrt(x)
y =
   2.2361    2.6358    3.8730
```

Expressing Function Arguments

We can write sin 2 in text, but MATLAB requires parentheses surrounding the 2 (which is called the *function argument* or *parameter*).

Thus to evaluate sin 2 in MATLAB, we type `sin(2)`. The MATLAB function name must be followed by a pair of parentheses that surround the argument.

To express in text the sine of the second element of the array *x*, we would type `sin[x(2)]`. However, in MATLAB you cannot use square brackets or braces in this way, and you must type `sin(x(2))`.

(continued …)

Expressing Function Arguments (continued)

To evaluate $\sin(x^2 + 5)$, you type `sin(x.^2 + 5)`.

To evaluate $\sin(\sqrt{x}+1)$, you type `sin(sqrt(x)+1)`.

Using a function as an argument of another function is called *function composition.* Be sure to check the order of precedence and the number and placement of parentheses when typing such expressions.

Every left-facing parenthesis requires a right-facing mate. However, this condition does not guarantee that the expression is correct!

Expressing Function Arguments (continued)

Another common mistake involves expressions like $\sin^2 x$, which means $(\sin x)^2$.

In MATLAB we write this expression as `(sin(x))^2`, *not* as `sin^2(x)`, `sin^2x`, `sin(x^2)`, or `sin(x)^2`!

Expressing Function Arguments (continued)

The MATLAB trigonometric functions operate in radian mode. Thus `sin(5)` computes the sine of 5 rad, not the sine of 5°.

To convert between degrees and radians, use the relation $q_{\text{radians}} = (p\ /180)q_{\text{degrees}}$.

Trigonometric functions:

`cos(x)`	Cosine; cos *x*.
`cot(x)`	Cotangent; cot *x*.
`csc(x)`	Cosecant; csc *x*.
`sec(x)`	Secant; sec *x*.
`sin(x)`	Sine; sin *x*.
`tan(x)`	Tangent; tan *x*.

Inverse Trigonometric functions:

`acos(x)`	Inverse cosine; arccos *x*.
`acot(x)`	Inverse cotangent; arccot *x*.
`acsc(x)`	Inverse cosecant; arccsc *x*.
`asec(x)`	Inverse secant; arcsec *x*.
`asin(x)`	Inverse sine; arcsin *x* .
`atan(x)`	Inverse tangent; arctan *x* .
`atan2(y,x)`	Four-quadrant inverse tangent.

Hyperbolic functions:

`cosh(x)`	Hyperbolic cosine
`coth(x)`	Hyperbolic cotangent.
`csch(x)`	Hyperbolic cosecant
`sech(x)`	Hyperbolic secant
`sinh(x)`	Hyperbolic sine
`tanh(x)`	Hyperbolic tangent

Inverse Hyperbolic functions:

`acosh(x)`	Inverse hyperbolic cosine
`acoth(x)`	Inverse hyperbolic cotangent
`acsch(x)`	Inverse hyperbolic cosecant
`asech(x)`	Inverse hyperbolic secant
`asinh(x)`	Inverse hyperbolic sine
`atanh(x)`	Inverse hyperbolic tangent;

User-Defined Functions

The first line in a function file must begin with a *function definition line* that has a list of inputs and outputs. This line distinguishes a function M-file from a script M-file. Its syntax is as follows:

```
function [output variables] = name(input variables)
```

Note that the output variables are enclosed in *square brackets,* while the input variables must be enclosed with *parentheses.* The function name (here, `name`) should be the same as the file name in which it is saved (with the .m extension).

User-Defined Functions: Example

```
function z = fun(x,y)
u = 3*x;
z = u + 6*y.^2;
```

Note the use of a semicolon at the end of the lines. This prevents the values of `u` and `z` from being displayed.

Note also the use of the array exponentiation operator (`.^`). This enables the function to accept `y` as an array.

(continued …)

User-Defined Functions: Example (continued)

Call this function with its output argument:

```
>>z = fun(3,7)
z =
   303
```

The function uses x = 3 and y = 7 to compute z.

(continued …)

User-Defined Functions: Example (continued)

Call this function without its output argument and try to access its value. You will see an error message.

```
>>fun(3,7)
ans =
     303
>>z
???  Undefined function or variable ’z’.
```

(continued …)

User-Defined Functions: Example (continued)

Assign the output argument to another variable:

```
>>q = fun(3,7)
q =
   303
```

You can suppress the output by putting a semicolon after the function call.

For example, if you type `q = fun(3,7);` the value of `q` will be computed but not displayed (because of the semicolon).

Local Variables: The variables `x` and `y` are *local* to the function `fun`, so unless you pass their values by naming them `x` and `y`, their values will not be available in the workspace outside the function. The variable `u` is also local to the function. For example,

```
>>x = 3;y = 7;
>>q = fun(x,y);
>>x
x =
   3
>>y
y =
   7
>>u
??? Undefined function or variable ’u’.
```

Only the order of the arguments is important, not the names of the arguments:

```
>>x = 7;y = 3;
>>z = fun(y,x)
z =
   303
```

The second line is equivalent to `z = fun(3,7)`.

You can use arrays as input arguments:

```
>>r = fun(2:4,7:9)
r =
   300      393      498
```

A function may have more than one output. These are enclosed in square brackets.

For example, the function `circle` computes the area *A* and circumference *C* of a circle, given its radius as an input argument.

```
function [A, C] = circle(r)
A = pi*r.^2;
C = 2*pi*r;
```

The function is called as follows, if the radius is 4.

```
>>[A, C] = circle(4)
A =
  50.2655
C =
  25.1327
```

A function may have no input arguments and no output list.

For example, the function `show_date` clears all variables, clears the screen, computes and stores the date in the variable `today`, and then displays the value of `today`.

```
function show_date
clear
clc
today = date
```

Examples of Function Definition Lines

1. One input, one output:

```
function [area_square] = square(side)
```

2. Brackets are optional for one input, one output:

```
function area_square = square(side)
```

3. Two inputs, one output:

```
function [volume_box] = box(height,width,length)
```

4. One input, two outputs:

```
function [area_circle,circumf] = circle(radius)
```

5. No named output: `function sqplot(side)`

Function Example

```
function [dist,vel] = drop(g,v0,t);
% Computes the distance travelled and the
% velocity of a dropped object,
% as functions of g,
% the initial velocity v0, and
% the time t.
vel = g*t + v0;
dist = 0.5*g*t.^2 + v0*t;
```

(continued …)

Function Example (continued)

1. The variable names used in the function definition may, but need not, be used when the function is called:

```
>>g = 32.2;
>>initial_speed = 10;
>>time = 5;
>>[feet_dropped,speed] = . . .
drop(a,initial_speed,time)
```

(continued …)

Function Example (continued)

2. The input variables need not be assigned values outside the function prior to the function call:

```
  [feet_dropped,speed] = drop(32.2,10,5)
```

3. The inputs and outputs may be arrays:

```
[feet_dropped,speed]=drop(32.2,10,0:1:5)
```

This function call produces the arrays `feet_dropped` and `speed`, each with six values corresponding to the six values of time in the array `time`.

Local Variables

The names of the input variables given in the function definition line are local to that function.

This means that other variable names can be used when you call the function.

All variables inside a function are erased after the function finishes executing, except when the same variable names appear in the output variable list used in the function call.

Methods for Calling Functions

There are four ways to invoke, or “call,” a function into action. These are:

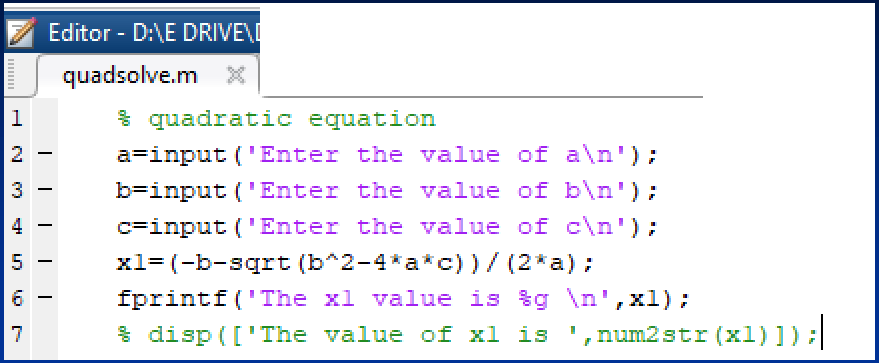
1. As a character string identifying the appropriate function M-file,

2. As a function handle,

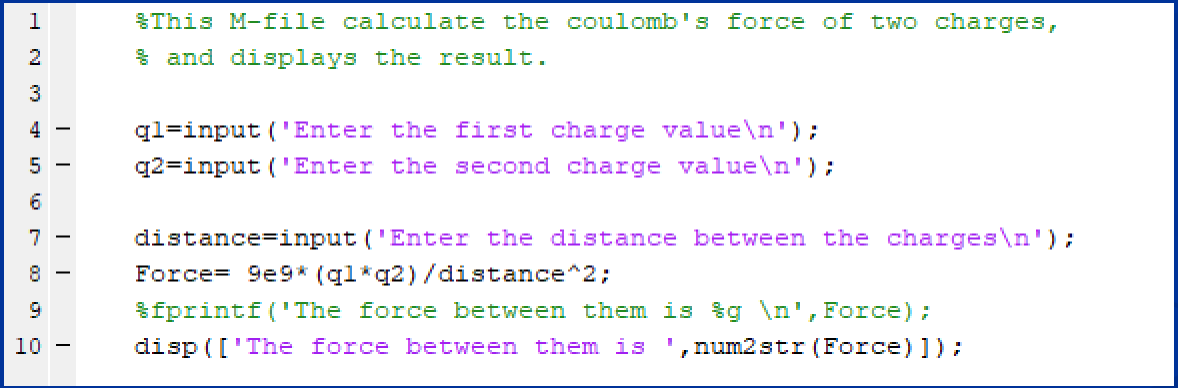
3. As an “inline” function object, or

4. As a string expression.

(continued …)



```
% quadratic equation
a=input('Enter the value of a\n');
b=input('Enter the value of b\n');
c=input('Enter the value of c\n');
x1=(-b-sqrt(b^2-4*a*c))/(2*a);
fprintf('The x1 value is %g \n',x1);
% disp(['The value of x1 is ',num2str(x1)]);
```



```
%This M-file calculate the coulomb's force of two charges,
% and displays the result.

q1=input('Enter the first charge value\n');
q2=input('Enter the second charge value\n');

distance=input('Enter the distance between the charges\n');
Force= 9e9*(q1*q2)/distance^2;
%fprintf('The force between them is %g \n',Force);
disp(['The force between them is ',num2str(Force)]);
```