EVAL FUNCTION

eval

Execute a string containing a MATLAB expression

The eval command is one of the most powerful and flexible commands in MATLAB. eval is short for evaluate, which is exactly what it does; it evaluates MATLAB expressions. Any command you can execute from the MATLAB prompt, you can use eval to execute the command from an M-file.

Syntax

```
eval(expression)
eval(expression,catch_expr)
[a1,a2,a3,...] = eval(function(b1,b2,b3,...))
```

Description

eval(expression) executes expression, a string containing any valid MATLAB expression. You can construct expression by concatenating substrings and variables inside square brackets:

expression = [string1,int2str(var),string2,...]

eval(expression,catch_expr) executes expression and, if an error is detected, executes the catch_expr string. If expression produces an error, the error string can be obtained with the lasterr function. This syntax is useful when expression is a string that must be constructed from substrings. If this is not the case, use the try...catch control flow statement in your code.

[a1,a2,a3,...] = eval(function(b1,b2,b3,...)) executes function with arguments b1,b2,b3,..., and returns the results in the specified output variables.

Remarks

Using the eval output argument list is recommended over including the output arguments in the expression string. The first syntax below avoids strict checking by the MATLAB parser and can produce untrapped errors and other unexpected behavior.

```
eval('[a1,a2,a3,...] = function(var)')     % not recommended
[a1,a2,a3,...] = eval('function(var)')     % recommended syntax
```

% Example 1

```
clc, clear
```

```
a='x^2';
b='-4*x';
c='+2';
yy=[a,b,c];        % This concatenate a,b,c in a single string;
                   % It is equivalente to yy='x^2-4*x+2';
x=2;
y=eval(yy)         % This evaluates yy=x^2-4*x+2 for x=2;

y2=x^2-4*x+2       % So you can see that y & y2 are equivalent
```

OUTPUT:

```
y =
   -2

y2 =
   -2
```

% Example 2

The code below do the same as Example 1 but with a different syntax (not recommended, but sometimes it is necessary)

```
clc, clear

a='y=';
b='x^2';
c='-4*x';
d='+2';

yy=[a,b,c,d];      % This concatenate a,b,c,d in a single string;
                   % It is equivalent to 'y=x^2-4*x+2';
                   % this means the expression may look like an 'assignment' string
x=2;
eval(yy)           % This syntax form is not recommended
                   % This evaluates yy=x^2-4*x+2 for x=2
```

OUTPUT:

```
y =
   -2
```

% Example 3

This for loop generates a sequence of 10 matrices named M1 through M10:

```
for n = 1:10
```

```
    magic_str = ['M',int2str(n),' = magic(n)'];
    eval(magic_str)
end
```

OUTPUT

The output is long. I present here part of it:

```
M1 =
   1


M2 =
   1   3
   4   2


M3 =
   8   1   6
   3   5   7
   4   9   2
```

…

Some more advance uses for eval

Some of the more common uses for eval are to load and save files with variable filenames, shell out (!, not supported on the Macintosh) to the operating system, and concatenate strings.

It is essential that a user understand the eval statement before using the powerful GUIs provided with MATLAB 4.x because the CallBack, ButtonDownFcn, WindowButtonDownFcn, WindowButtonMotion, WindowButtonUp, and KeyPressFcn properties are variations of eval.

Below are some basic examples of how to use eval. These examples will provide you with the ability to create just about any expression in MATLAB.

EXAMPLE 1: Saving data to incrementally numbered ASCII files, i.e., FILE1, FILE2, FILE3, ...

```
rootname = 'file';      % Root filename
extension = '.dat';     % Extension for the files
% The following loop concatenates the root filename,
% an integer value, and the extension to create the
% file in which the data is saved.
for data = 1:10
```

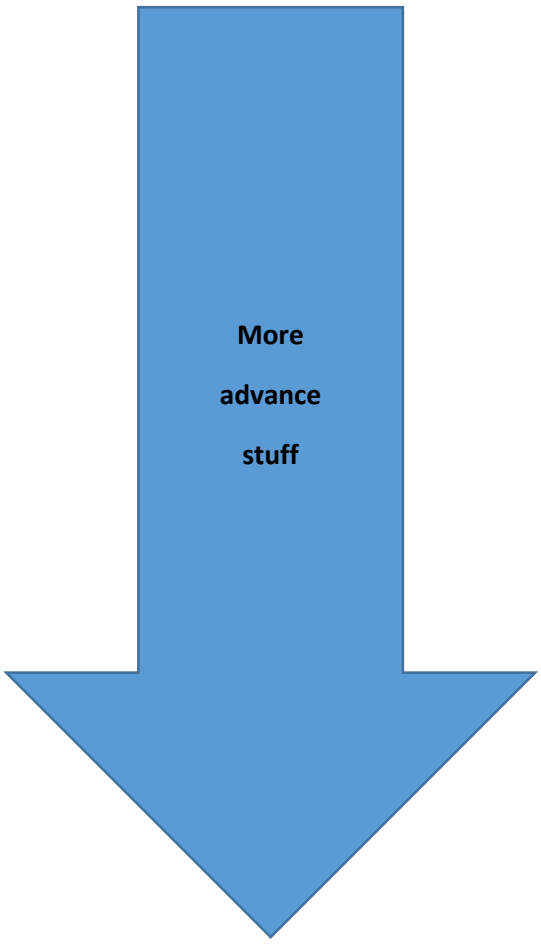
```
  filename = [rootname, int2str(data), extension];
  eval(['save ', filename , ' data /ascii'])
end
```

EXAMPLE 2: Load data from the ASCII files generated above.

```
rootname = 'file';      % Root filename
extension = '.dat';     % Extension of the files
% The following loop concatenates the root filename,
% an integer value, and the extension to create the
% file to be loaded.
for data = 1:10
  variable = [rootname, int2str(data)];
  filename = [variable, extension];
  eval(['load ', filename])
  eval(['data', num2str(data), ' = ', variable, ';'])
  eval(['clear ', variable])
end
```

REFERENCES

https://nf.nci.org.au/facilities/software/Matlab/techdoc/ref/eval.html
https://www3.nd.edu/~dtl/cheg258/notes/doc/tec1.3.html



1.3 The eval Command

The eval command is one of the most powerful and flexible commands in MATLAB. eval is short for evaluate, which is exactly what it does; it evaluates MATLAB expressions. Any command you can execute from the MATLAB prompt, you can use eval to execute the command from an M-file.

Some of the more common uses for eval are to load and save files with variable filenames, shell out (!, not supported on the Macintosh) to the operating system, and concatenate strings.

It is essential that a user understand the eval statement before using the powerful GUIs provided with MATLAB 4.x because the CallBack, ButtonDownFcn, WindowButtonDownFcn, WindowButtonMotion, WindowButtonUp, and KeyPressFcn properties are variations of eval.

Below are some basic examples of how to use eval. These examples will provide you with the ability to create just about any expression in MATLAB.

EXAMPLE 1: Saving data to incrementally numbered ASCII files, i.e., FILE1, FILE2, FILE3, ...

```
rootname = 'file';      % Root filename
extension = '.dat';     % Extension for the files
% The following loop concatenates the root filename,
% an integer value, and the extension to create the
% file in which the data is saved.
for data = 1:10
  filename = [rootname, int2str(data), extension];
  eval(['save ', filename , ' data /ascii'])
end
```

EXAMPLE 2: Load data from the ASCII files generated above.

```
rootname = 'file';      % Root filename
extension = '.dat';     % Extension of the files
% The following loop concatenates the root filename,
% an integer value, and the extension to create the
% file to be loaded.
for data = 1:10
  variable = [rootname, int2str(data)];
  filename = [variable, extension];
  eval(['load ', filename])
  eval(['data', num2str(data), ' = ', variable, ';'])
  eval(['clear ', variable])
end
```

EXAMPLE 3: Using eval to print to random files.

```
rootname = 'fig';                  % Root filename
for x = 1:10
  figure(x)
  plot(rand(x))                    % Random plot
  filename = [rootname, int2str(x)];  % Concatenate the
                                   % root filename and
                                   % the integer.
  eval(['print -pds ',filename])     % Print to the file
                                   % using eval
end
```

EXAMPLE 4: Reassign the contents of a variable. In this case, you are prompted to enter the name of a variable. In order to use the data in the M-file, you must assign the contents of the unknown variable to a known variable.

```
var = magic(10);
<----------------  fun1.m  ---------------->
% Enter 'var' as the variable name.
variablename = input('Enter the name of the variable:  ','s');
a = eval(variablename);            % Reassign the contents of
                                   % variablename to a.
b = 2*a;
mesh(b)
```

EXAMPLE 5: Evaluate a MATLAB command. Generate a bode plot based on the numerator and denominator entered by the user.

```
num = input('Enter the numerator: ');
den = input('Enter the denominator: ');
eval(['bode(',mat2str(num),',',mat2str(den),')'])
```

EXAMPLE 6: Converting a numeric string to its numeric value, i.e., convert '1' to 1.

```
x = '1';
y = eval(x);
```

EXAMPLE 7: Error trapping. In MATLAB v4.x, you can trap errors using the eval command. The following example illustrates this concept.

```
% CASE 1: A is not defined
clear all
eval('B = A','disp(''A is undefined'')')
```

```
% The message "A is undefined" is displayed
% CASE 2: A is defined
A = 1;
eval('B = A','disp(''A is undefined'')')
% The following is displayed in the Command Window:
%
% B =
%     1
```

The statement, eval('B = A','disp(''A is undefined'')'), contains two inputs. The underlying concept here is that eval will try the first input, and if an error is detected, it will then use the second input. This method is referenced as eval(<try>,<catch>).
In this example, the first input, 'B = A' is the primary input. It is the input that eval will attempt to evaluate first. If an error is detected, then the second input, 'disp(''A is undefined'')', is evaluated.

Examples 1 - 4 use eval to concatenate a series of strings together to form a MATLAB expression. eval substitutes the value of the variable(s) into the expression, and then evaluates the entire expression. Understanding this basic concept is all you really need to do in order to use the eval command.

Note that the numeric values are converted to strings. This is necessary because eval only accepts string inputs. Also, it is necessary to convert numbers to strings because everything inside the brackets, [], must be the same type. Matrices in MATLAB are either string or numeric.

Example 6 is useful when using editable text blocks for numeric input. Since an editable text block stores its input in its String property, when you do a get(h,'String'), a string is returned. By using eval, it is easy to convert the string to the numeric value.

Another important characteristic of eval is the single quote ('). Example 7 illustrates this. When a single quote is nested in a string, it must be defined by using two single quotes, ''. Note that this is not a double quote. Keeping track of quotes is extremely important. eval is often used to create very complicated expressions, which have many single quotes in a row. For example, eval('disp(''''This is a string'''')') displays 'This is a string' in the command window.

There are several traps of which you should be aware. First, using the eval statement when it is not necessary. For example:

```
eval(['title(''Plot #', num2str(1), ''')'])
```

This is the same as the following:

```
title(['Plot #',num2str(1)])
```

The latter is much easier to read and write.
The second has to do with how eval operates in a function when the input variable is redefined in an eval statement. For example:

```
function out = foo(in)
eval('in = in + 1')
out = in;
```

When this function is called, the value of the variable passed to foo is changed. For example:

```
>> a = 1;
>> b = foo(a)
in =
    2
b =
    2
>> a
a =
    2
```

This occurs because MATLAB passes a variable into a function by reference. This means that 'in' is essentially a pointer to 'a'. Since eval is not returning a left-hand argument, the change is made to 'a'. To avoid this behavior, modify the eval statement so that a left-hand argument is used. For example:

```
in = eval('in+1');
```

In this case, MATLAB assigns the value on 'in+1' to 'in', and 'a' is not affected.

Additional functions similar to eval are feval, unix, and dos. feval is used to evaluate MATLAB functions. For example, example 5 can be re-written as:

```
feval('bode',num,den)
```

The first input is the name of the function to be evaluated, and subsequent inputs are the inputs to the function. As you can see, this is much more clear than eval(['bode(',mat2str(num),',',mat2str(den),')']).

UNIX and DOS are used to open unix or dos shells to execute the given command. For example:

```
file = input('Enter the file name and extension: ','s');
unix(['lpr -Pprinter ',file])
dos(['copy /b ',file', lpt1'])
This can be re-written using eval:
file = input('Enter the filename and extension: ','s');
eval(['!lpr -Pprinter ',file])
eval(['!copy /b ',file', lpt1'])
```

The only difference between using unix or dos and eval is that with eval, you must use the bang(!) to shell out. The unix and dos commands do this automatically, so it is best to use these commands when shelling out to the operating system.

REFERENCES

https://www3.nd.edu/~dtl/cheg258/notes/doc/tec1.3.html