
Dealing with MATLAB errors

When you use Matlab to perform calculations, you are bound to make error - from simple typos to subtle errors in logic. Steps to take when you encounter an error.

- Tell yourself that the computer does not hate you.
- Articulate clearly what you are trying to do.
- Read the error message carefully and completely. From the error message, identify what is causing the error and why (if possible).
- Read the on-line help information on the function (if that is the problem) or on the topic.
- See if other people have encountered such an error and work with them to resolve it.
- If your friends cannot determine the problem or find a solution, ask the instructor.

Following are few pointers in dealing with Matlab errors.

Omitting operators in expressions

Remember that MATLAB does not understand a statement like

```
b(3 + a)
```

to mean "b times the quantity (3 + a)". Rather, MATLAB interprets it as "the (3+a)th element of b". This interpretation is not likely to give you the result you are expecting. To produce the desired result, be explicit about the multiplication operation.

```
b*(3 + a)
```

Confusing array and matrix operations

The MATLAB operators \wedge , $*$, \backslash and $/$ all "do the right thing" when applied to array variables - they compute the linear-algebra operation, if it is defined. This is fine if you are expecting to do numerical linear algebra. It is frustrating if you are trying to do element-by-element computations.

Very often you will get an error message when you "forget to add the dot". For example,

```
v1 = [2 5 1]
1/v
??? Error using ==> /
Matrix dimensions must agree.
```

Since 1 is a 1-by-1 array and v1 is a 1-by-3 array, the shapes of these arrays are not compatible for right division.

In some cases, the matrix operation is defined and so you won't get an error message that can help you fix a mistake. For example,

```
v1 = [2 5 1]
v2 = [3 4 2]
v1/v2
ans =
    0.9655
```

In this case, the shapes of the arrays were such that right-division is defined and so a valid (though possibly unexpected) result is returned. This type of error can produce subtle logic errors in a program. Pay careful attention to your computations and be "picky" about what you type in. Read them carefully and use the "dot" when you mean it.

MATLAB is case-sensitive

There is a difference between Tom, tom and TOM to MATLAB for names of variables and functions. Generally, MATLAB function names should be rendered to in lower-case letters (sin, not SIN) though sometimes MATLAB is flexible enough to recognize the equivalence (as seems to be the case for TITLE and related plot-labelling commands). Be safe - use lower case.

Note: Help information available via the help command often uses upper-case for emphasis. Unless the help information for a function explicitly says that a function name is to include upper-case letters, assume that all names should be in lower-case.

Reading MATLAB error messages

MATLAB error messages can sometimes be very helpful and can sometimes be quite obscure. The degree to which a message can help is significantly improved if you spend a little time reading what information is provided in the message.

Read the whole message. Start from the last typed command to the new cursor location. For example, consider the following error in a script called plotex.

```
>> plotex
??? Error using ==> plot
Vectors must be the same lengths.

Error in ==> c:myMatlabStuff:plotex.m
On line 28 ==> plot(t,y,'ob',t,yfit,'r-')
```

It is clear that we need to start with line 28 of the script. Most likely the problem is in the variables (t, y and yfit).

Read from the bottom up. If you are using "function functions" (doing things like root-finding or integration), the error message will display the "stack trace" of the functions involved and the specific lines (and line numbers) where the execution halted. If you do not recognize a function as one you have written, it is probably a MATLAB built-in function. Read the error message to see where your activity starts.

Look for the pointer. Sometimes the error message "points" to the place where the interpreter bombed out. This is the usual response to syntax errors. For example, forgetting the closing parenthesis is a common way to see the pointer (the pipe character, |):

```
>> x = sin(t
??? x = sin(t
    |
Improper function reference. A ",", " or ")" is expected.
```

The pointer is telling you where to start looking.

Negative numbers in functions

Many engineering problems use three functions/operations that can cause problems when their inputs are negative numbers:

- log, log10: The logarithm of a negative number is a complex number. For example

```
>> log(-3)
ans =
1.0986 + 3.1416i
```

- sqrt: The square root of a negative number is usually the first place you see imaginary numbers. As with log, MATLAB handles the operation in a "natural" manner that is mathematically correct but practically frustrating, at times.

```
>> sqrt(-3)
ans =
    0 + 1.7321i
```

- \wedge , \wedge^{\cdot} : Raising a number to a power less than 1 is a generalization of the square-root function. For example,

```
>> (-3)^0.2
ans =
    1.0078 + 0.7322i
```

Note: If parentheses are not used to force precedence, the result is different.

```
>> -3^0.2
ans =
   -1.2457
```

Thus make sure you let MATLAB know what you mean.

Most engineering computations and formulas assume that the results from computations are real-valued. To avoid the problems noted above:

- Use `abs` to ensure a positive input (e.g., `log(abs(x))`).
- Check the sign of the argument before using any of these functions.
- Ask yourself why your computation would lead to a negative input into these functions.

Creating Function M-files

User defined functions are stored as M-files. To use them, they must be in the current directory. The difference between scripts and functions is that a function typically uses data passed as arguments and returns a result. A script file is simply a collection of MATLAB commands.

All functions have a similar syntax, whether they are built-in functions or user-defined functions

```
Name
Input
Result
```

For example: `A=cos(x)`

Defining a function – the first line

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

```
    Comments explaining the purpose, input, and output variables.
```

```
    Matlab statements. All output variables must be assigned values.
```

```
end (optional)
```

Example: Function to compute area and volume of a cylinder.

The file must be saved as `cylinder.m` in the current working directory. Because of the use of array operations, the function will work when heights and radius are arrays.

```
function [area, volume] = cylinder(height, radius)
% function to compute the area and volume
% of a cylinder
% usage: [area, volume]=cylinder(height, radius)
    base = pi * radius.^2;
    volume = base .* height;
    area = 2 * pi * radius .* height + 2 * base;
end
```

Compute area and volume of a cylinder with height = 2 and radius = 1.

```
[a, v] = cylinder(2, 1)
```

```
a =
    18.8496
v =
     6.2832
```

If no output variable is specified, it returns only the area.

```
cylinder(2, 1)
```

```
ans =
    18.8496
```

If only one output variable is specified, it returns only the area. Variable name does not matter. We can say v (or even volume) but it still returns area.

```
v = cylinder(2, 1)
```

```
v =
    18.8496
```

We can compute areas and volumes of several cylinders by passing arrays to the function.

```
r = [1:.2:2];
h = 2;
[a, v] = cylinder(h, r);
disp(' Height(in) Radius(in) Area(in^2) Volume(in^3)')
disp([2*ones(1,length(r))', r', a', v'])
```

```
Height(in) Radius(in) Area(in^2) Volume(in^3)
    2.0000    1.0000    18.8496     6.2832
    2.0000    1.2000    24.1274     9.0478
    2.0000    1.4000    29.9080    12.3150
    2.0000    1.6000    36.1911    16.0850
    2.0000    1.8000    42.9770    20.3575
    2.0000    2.0000    50.2655    25.1327
```

Class Activity 1

Type and save the function cylinder.

(1) Use the function to compute areas and volumes of cylinders of radius 2 in and heights from 2 in to 10 in increments of 2. Display your results in a neat table with appropriate headings.

(2) Use the function to compute areas and volumes of the following cylinders.

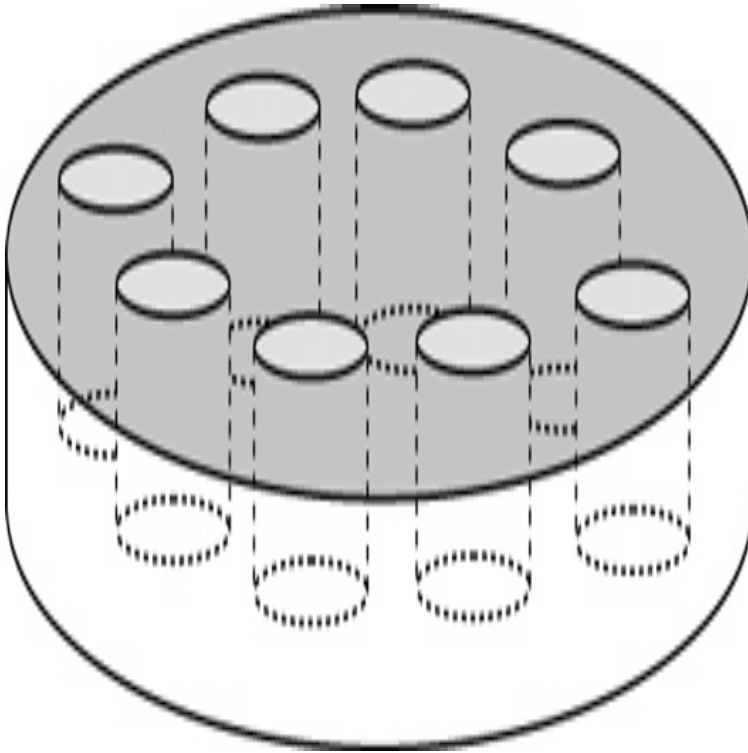
Heights = [2, 5, 10, 20]

Radii = [2, 4, 8]

Consider ALL combinations of heights and radii. [Hint: Use meshgrid function.] Display your results in a neat table with appropriate headings.

Example: Computing volume of a cylindrical solid with cylindrical holes

The cylinder function can be used in a loop as part of another function to compute volume of a cylindrical solid with cylindrical holes.



```
function v = cylSolidWithHoles(hc, rc, rh)
% function to compute the volume
% of a cylindrical solid with cylindrical holes.
% Input:  hc = height of the solid
%         rc = radius of the solid
%         rh = list with radii of holes
% Output:
%         v = volume of the solid with holes
%
[a, v] = cylinder(hc, rc);
for i=1:length(rh)
    [ah, vh] = cylinder(hc, rh(i));
    v = v - vh;
end
end
```

volume of a solid with $h = 25$ in, radius = 3 in, and four 0.5 in radius holes.

```
cylSolidWithHoles(25, 3, .5*ones(1,4))
```

```
ans =
    628.3185
```

Class Activity 2: Function for analyzing simply supported beams.

Create a function to compute maximum moment, shear & deflection of a simply supported beam. Recall that for a simply supported beam with span L subjected to distributed load w we have

$$\text{Max bending moment} = wL^2/8$$

$$\text{Max shear} = wL/2$$

$$\text{Max deflection} = \frac{5wL^4}{384EI} \text{ where } E \text{ is the modulus of elasticity and } I \text{ is the moment of inertia of the section.}$$

Call your function **simpleBeam**. The input arguments should be w (kip/ft), L (ft), I (in^4), and E (ksi). The function should return bending moment (kip-ft), shear force (kip), and deflection (in). Make sure your function performs appropriate unit conversions when performing computations.

Test your function by analyzing a beam with a span = 20 ft, distributed load = 2 kip/ft, $I = 2750 \text{ in}^4$ and $E = 29000 \text{ ksi}$.

Example: Function for extracting data from steel sections database.

The steel sections data is available as an excel file.

aiscmanua w	a	d	bf	tw	tf	kdes	kdet	k'
W44x335	335	98.5	44	15.9	1.03	1.77	2.56	2.625
W44x290	290	85.4	43.6	15.8	0.865	1.58	2.36	2.4375
W44x262	262	76.9	43.3	15.8	0.785	1.42	2.2	2.25
W44x230	230	67.7	42.9	15.8	0.71	1.22	2.01	2.0625
W40x593	593	174	43	16.7	1.79	3.23	4.41	4.5

Our main tasks are to read the data from the excel file, match the given section name with that in the first column of the file and then read appropriate columns from the matched row. Matlab functions **xlsread** and **strmatch** are useful in accomplishing these tasks.

`[num, txt] = xlsread(filename, 'range')` returns numeric data in array `num` and text data in cell array `txt`. 'range' defines the region to be read. For example, 'D2:H4' represents the 3-by-5 rectangular region between the two corners D2 and H4 on the worksheet.

`x = strmatch(str, strarray, 'exact')` compares `str` with each row of `strarray`, looking for an exact match of the entire strings.

In the following function we first use `xlsread` and read the entire first column (from A2:A275). There is no numerical data in this column. All text data is read as `secNames`. The `strmatch` function then matches the given section with these names and finds the row number (in variable `num`) that matches. If there is no match then `num` is returned as an empty variable (with length = 0). We display a message and return 0 values for the properties. If we do have match then we read all numeric data from the file and extract appropriate columns from the row that matches.

```

function [A, Ix, Iy] = AISCWshapeProps(sec)
% Extract A, Ix, and Iy for a given section from the AISC sections database
% Input = text string with section name (e.g. W18x143)
% Output: A = area, Ix & Iy = moment of inertias about x and y axes
%
% Assign default values
A=0; Ix=0; Iy=0;
% Read the section names from the first column
[num, secNames] = xlsread('AISCWShapesDatabase.xls','A2:A275');
% Find a match
num = strmatch(sec, secNames, 'exact');
if length(num)==0
    disp('No match for the given section name')
else
    data = xlsread('AISCWShapesDatabase.xls','B2:Z275');
    secData = data(num,:);
    A = secData(2); Ix = secData(12); Iy = secData(16);
end
end

```

We test the function by extracting properties for a W18x143 section.

```

[A, Ix, Iy] = AISCWshapeProps('W18x143')

A =
    42.1000
Ix =
    2750
Iy =
    311

```

Of course a message is returned if there is no match.

```
AISCWshapeProps('W180x143')
```

```

No match for the given section name
ans =
    0

```

Class Activity 3: Function for analyzing W shape simply supported beams.

The beam analysis function can be made much more useful if it can automatically extract the moment of inertia values from the steel database. Create a function called **WSectionSimpleBeam**. The input arguments should be w (kip/ft), L (ft), and section name (text string). The function should make use of the **AISCWshapeProps** function to get the moment of inertia values and then call the **simpleBeam** function to perform the moment, shear, and deflection calculations. Note that for all steels $E = 29\,000$ ksi.

Test your function by analyzing a beam with a span = 20 ft, distributed load = 2 kip/ft, and W18×143 section.

