Intro Learning Module - Plot sigma_x vs. inner radius (Take 4)

Author: Rajesh Bhaskaran, Cornell University

Plot $_x$ vs. r_i (Take 4: Functions)

Let's create a function to calculate the bending stress that outputs $_x$ given (M, r_i, r_o). Functions are really useful to break down your code into modules and also reuse parts of your code.

Let's first pick MATLAB's brain on how to create functions in MATLAB. Bring up the following page from Documentation.

Home (tab) > Documentation (icon) > MATLAB > Programming Scripts and Functions > Functions > Function Basics > function

Take a couple of minutes to peruse the examples for functions with one output and functions with multiple outputs. MATLAB has extensive documentation on the use of functions; however, one has to poke around a bit before finding the most useful information. I personally go for the examples first.

The correct syntax for creating a function is:

```
function return_value = function_name(parameter_1, parameter_2,...)

%function description - MUST  be in a comment

code...

return_value = value
```

A few noteworthy points to ponder:

1. *return_value* is the only data that gets passed back to the main code.
2. You do not need to have a function description but it is good programming practice to add a comment on each function that describes what the function does. Also, if you add a description of the function, MATLAB will be able to index it and return a description of your function if you type `help function_name` in the Command Window.
3. Once you have created a function you MUST name the .m file with the same name the function has. Otherwise MATLAB will not be able to access your function when you call it.
4. You need brackets around your outputs if you have multiple output arguments. For only one output, the convention is to not include brackets.

We will start by creating the bending stress function that outputs $_x$ given (M, r_i, r_o).

Editor (tab) > New > Function

Note that the main code for creating a function is already built-in. We could have also selected New > Script and get a blank page, but it's nice to have the function template right away. What's important to understand here is that both the script files and function files have the file extension **.m**. A function file is really just a script file but with a function statement in its first line and with its function name matching its file name.

Replace the default template by the following statements.

```
function sigma_x = bending_stress(M,ro,ri)
I = pi*(ro^4 - ri.^4)/4;
sigma_x = 1e-6*M*ro./I;
```

You can be lazy like me and copy-and-paste the last two statements from your previous code. Save this file as *bending_stress*, which is the name that MATLAB automatically assigns the file. Thus, the function name and the file name have the same name.
Bring up *beam3.m* in the MATLAB editor. Make a copy of *beam3.m* using ***Save As ...*** and call the new file *beam5.m*. In this file, comment out the lines below since this calculation is now done within the function.

```
I = pi*(ro^4 - ri.^4)/4;
sigma_x = 1e-6*M*ro./I;
```

We'll replace these statements with a call to the *bending_stress* function. The following statement does this:

```
sigma_x = bending_stress(M,ro,ri);
```

Add this to *beam5.m*. Run the file and check the output. You should get the same plot you got with *beam3.m*.

Subfunctions

Functions can be called from within a function. You can put multiple functions in one M-file as shown in the example located in this documentation page.

Home (tab) > Documentation (icon) > Search: *Subfunctions* > Local Functions

Please go through the subfunction page (or local function as MATLAB calls it) in its entirety before watching the following tutorial video.

Your main_function file should look like this.

```
function main_function

% Program to solve toy beam
% bending problem
% Author: Anonymous
% Date: circa 2010

%Clear everything
clear all; %Clear
clc; %Clear command window

%Calculate reactions
A = [1 1; 0 12];
B = [400;  2400];
R = A\B; %Reaction vector

%Calculate sigma_x
M = -600;
ro = 2e-2;
ri = 1e-2*(0.5:0.1:1.5);
sigma_x = bending_stress(M,ro,ri);

%Plot sigma_x vs. ri
figure(1); %Create figure #1
clf; %Clear current figure
h=plot(1e2*ri,sigma_x,'-r');
set(gca,'Box','on','LineWidth',2,...
    'FontName','Helvetica',...
    'FontSize',14);
    %Set axis properties
set(h,'LineWidth',2);
    %Set linewidth for curve
xlabel('r_i (cm)');
ylabel('\sigma_x at O (MPa)');
title('Bending stress');
axis square; %Make axis box square
end

function sigma_x = bending_stress(M,ro,ri)
I = pi*(ro^4 - ri.^4)/4;
sigma_x = 1e-6*M*ro./I;
end
```

Subfunction workspace

The last paragraph of the Local Functions documentation page says that "All functions, including local functions, have their own workspace that are separate from the base workspace. Local functions cannot access variables used by other functions unless you pass them as arguments" and vice-versa. The following video covers this very important concept.

That brings us to the end of this tour. Before we part, let's remind ourselves of some important programming guidelines that we have followed in this tour:

- Develop code incrementally, testing obsessively at each stage. Develop a plan for how you are going to build your code before you sit at the computer.
- Dig through the MATLAB help diligently to figure out how to use specific functions etc. Usually, the examples are the best place to start. This is a better strategy than desperately hunting for the TA every time you need help with your code.
- Comment your program liberally.

In addition to this tutorial, there are several other references to use when learning Matlab. This link here will take you to Mathworks' video tutorials for Matlab.

Go to Tips

Go to all MATLAB Learning Modules