# Intro Learning Module - Plot sigma_x vs. inner radius

Author: Rajesh Bhaskaran, Cornell University



## Plot $_x$ vs. $r_i$

### Calculate $_x$ at *O*

Essentially, we need to repeat our previous calculation of $_x$ at *O* for a range of $r_i$values. One way to do this is to loop over the previous calculation. There are two types of ***loops***: ***for*** and ***while***. Let's check the documentation to see what MATLAB provides for loops. Enter *for loop* in the search bar and then select *for* which is the first entry up top. Look at the first example. This shows the format of the *for* loop. You can often speed up the execution of MATLAB code by replacing for and while loops with vectorized code. That is what we'll do in Step 4. Let's stick it out with slowpoke *for* loop for now.

We'll create a new script file from *beam.m* and implement the *for* loop in that file.

**Editor (tab) > Save > Save As**

Enter `beam2.m` for filename. Confirm that you are editing *beam2.m* (*beam.m* remains intact).

We need to put the following three statements inside of our loop.

```
 9 - ri = 1e-2;
10 - I = pi*(ro^4 - ri^4)/4;
11 - sigma_x = 1e-6*M*ro/I
```

A little exercise to keep you awake:

1. Create a counter *k* around these three statements using the format in the documentation example. We'll create 11 $r_i$ values, so your counter should go from 1 to 11.
2. Indent the statements inside the loop using the *Tab* key. This makes it clear what's being looped over.

Don't cheat by looking at my code snippet below.

```
 9 - for k = 1:11
10 -     ri = 1e-2;
11 -     I = pi*(ro^4 - ri^4)/4;
12 -     sigma_x = 1e-6*M*ro/I
13 - end
```

Next, we replace *ri* and *sigma_x* with *ri(k)* and *sigma_x(k)*, respectively. Modify the $r_i$ statement such that we get a linear variation between $r_i$(k)=0.5e-2 m when *k*=1 and $r_i$=1.5e-2 m when *k*=11.

```
for k=1:11
    ri(k) = 1e-2*( 0.5 + 0.1*(k-1) ) ;
    I = pi*(ro^4-ri(k)^4)/4;
    sigma_x(k) = -1e-6*M*ro/I
end
```

Note that we did *not* replace *I* with *I(k)*. Essentially, *I* is recalculated during each iteration but its value is not stored, saving us some memory. It's always a good idea to be stingy with memory usage since this will make the program run faster. Save your program by clicking on the *Save* icon but don't run it yet.

Since we have replaced some scalar variables with arrays, it's a good idea to clear the *Workspace* before running the program. At the command prompt, type

```
clear all
```

Check that your *Workspace* is cleared. Type *help clear* at the command line for help on this command. The command line help is a quick-and-easy way to check the documentation. But the information is not as extensive as the documentation browser.

Clear the command window by entering `clc`

Now run the program from the command line: `beam2`

Our program reports sigma_x at the end of each iteration (since we didn't add a semi-colon after the *sigma_x* statement). From the output, we can deduce that *sigma_x* is a row vector, with the columns getting built up at each iteration. Note that a different way to look at the sigma_x array is to double-click on *sigma_x* in the Workspace.

How do we know if the results are right? Let's spot-check the result for *ri*=1e-2 m against the previous step; this value of *ri* corresponds to *k*=6. What is your value of *sigma_x(k)* for *k*=6? You can check this from the output or by typing `sigma_x(6)` at the command line. My value is exactly the same as in Step 2 ... nailed that one! How about you?

Now that we have spot checked our result, add a semi-colon after the *sigma_x* statement to avoid cluttering up the output.

## Plot $_x$ vs. $r_i$

Let's pick MATLAB's brain on how to make a plot.

**Home (tab) > Documentation (icon) > Search: *Plotting Functions* > Plotting Functions**

Note that you can access this exact page without using the search field. Doing this will give you a sense of how to navigate the documentation.

**Home (tab) > Documentation (icon) > MATLAB > Graphics > 2-D and 3-D Plots > Plotting Basics > Plotting Functions**

This page contains a lucid description of the extensive plotting capabilities available in MATLAB. When looking for plotting help, this would be the first place to turn to.

In your program, leave a blank line and start a new section for plotting by adding a comment line. Then, plot *sigma_x* (MPa) vs. *ri* (cm).

```

%Plot sigma_x vs. ri
plot(1e2*ri,sigma_x,'-r');
```

Click the *Run* icon in the Editor to create the plot. Things to note: the factor `1e2` is used to convert `ri` to cm. The `'-r'` is the curve style specification with `-` and `r` specifying a solid line and red color, respectively. Check the documentation for all the line styles and colors available.

Check the plot: Does the $_x$ value at $r_i$=1 cm look right? Does the trend for $_x$ in the plot look right?

Add axis labels:

```
xlabel('r_i (cm)');
ylabel('\sigma_x at O (MPa)');
```

Click the *Run* icon in the Editor. The underscore symbol (_) gives subscripts and `\sigma` gives the corresponding Greek symbol.

See my entire program here (right click and select save target as, or just left-click and copy-paste in the editor).

See how we are building up the program slowly and checking the results obsessively at each stage? This is a good programming approach and ends up being the most time-efficient especially for larger programs. So I'd highly recommend this approach.

Recall that "you can often speed up the execution of MATLAB code by replacing for and while loops with vectorized code". We are into speed, so let's vectorize our program.

**Go to Step 4: Plot $x$ vs. $r_i$ (Take 2)**

Go to all MATLAB Learning Modules