How to succeed in Math 365

Table of Contents

These are the homework tips for doing Math 365 homework

Introduction

Below are several tips that will help you do your homework.

```
function main()
format short;

% Run tips that have actual code for demonstration.
tip1()
tip2()
tip3()
tip7()
tip8()
tip9()
tip11()

end
```

Tip #1 : Physical constants

Do not "hardwire" approximations when Matlab can compute them to full precision. A rule of thumb for this course is : **You should never need your calculator in this course**!

```
function tip1()
```

Example: Compute the area of a triangle wth sides A=5.1 and B=3.2 and enclosed angle theta = 20 degrees.

```
% This is the correct way to solve this problem :
theta = 20*pi/180;
sin_theta = sin(theta);   % Good : Let Matlab compute the sin(th) for you.
A = 5.1;
B = 3.2;
Area = 0.5*A*B*sin_theta;
fprintf('Area = %24.16f\n',Area);
```

```
        Area =       2.7908843695374568
```

```
% This is the wrong way!
sin_theta = 0.3420;   % Don't approximate the sin(x)
A = 5.1;
B = 3.2;
Area = 0.5*A*B*sin_theta;
fprintf('Area = %24.16f\n',Area);
```

```
        Area =       2.7907200000000003
```

Do you notice the difference in the values?

Especially don't "hardwire" numbers if

- You don't tell me where your hardwired value came from.
- You supply very few digits of a value that we know to full precision.
- You don't indicate how accurate your value is. For example, you might get values from a CAD package, and report them to full precision (e.g. 16 digits). But in fact, it is unlikely that a CAD package can give you that much precision. Or if it does, you may well have been able to compute the same values in Matlab.

Examples of the above errors are below.

```
% The exact area is
area_exact = 1.55678393345;   % where did this come from?


% The value of e, obtained from my calculator is
e_const = 2.7183;     % Much better to use e_const = exp(1)

% These numbers appear to be very "accurate", but who really knows?
x = [1.2348454366677766;  4.556667841225998];    % Really?
y = [4.5677445639358711; -2.345664448941219];
```

The obvious exceptions to this rule are when you are asked to supply phyiscal constants, such as the acceleration of gravity, a thermal conductivity, or some other physical constant in the problem that is only given to a limited number of digits. In the above examples, we were given the length of the two sides of the triangle to only a single digit. As far as this problem is concerned, then, these are the "exact" values for these quantites. Here are a few more examples.

```
g = 9.81;    % m/s acceleration due to gravity

beta = 1.2e-4;  % Thermal conductivity.

end  % end of tip 1
```

Tip #2 : Formatting output

Don't use the 'disp' command to show your results. Most likely, you won't show enough digits.

```
function tip2()

two_times_pi = 2*pi;
```

This will most likely not produce the right number of digits for display

```
disp(two_times_pi)
```

The formatting will depend on the current setting for the last format statement you issued, which may or may not be precisely what you want. That said, it is rarely a good idea (and generally considered lazy!) to use the 'format' command anywhere other than at the command line prompt.

```
    6.2832
```

Instead, use the 'write_file' function to save your results to a file. You can check the results in the file by using the 'type' command.

```
write_file(two_times_pi,'two_times_pi.out');
type two_times_pi.out
```

```
   6.2831853071795862e+00
```

Or, use 'fprintf'

```
fprintf('2*pi = %24.16f\n',two_times_pi);
```

```
2*pi =       6.2831853071795862
```

```
end % end of tip 2
```

Tip #3 : Line continuation '...'

Don't let your homework text exceed the 76 character limit. You can see this limit as a thin line near the right edge of the Matlab editor. Don't let your code or comments go past this line. To continue your code from one line to the next, use the '...' line continuation characters, as the example below shows.

```
function tip3()
```

Don't let your code run off the edge of the page like this...

```
x = [1,2,3,4,sin(10), linspace(1,100), rand(1,10), 100004, sin(56), 67.5678, atan(
```

This is much nicer, easier to read, and shows that you are not hiding anything.

```
x = [1,2,3,4,sin(10), linspace(1,100), rand(1,10), 100004, sin(56), ...
    67.5678, atan(-1), sin(456.7)];
```

To allow a longer string to continue across more than one line, use the concatenation operator []. For example,

```
str = sprintf(['This is a very long string.  In fact, it is so ',...
    'long that \nit runs off the end of the page.']);
fprintf('%s\n',str);
```

Notice that this string is broken into two pieces, which are then concatenated horizontally. Also we have included a 'line feed' character '\n' in the middle so that the long line, when printed, will not run off the page.

```
This is a very long string.  In fact, it is so long that
it runs off the end of the page.
```

```
end  % end of tip 3
```

Tip #4 : Typeset any explanatory text

Typeset any explanatory text that explains your method, or a particular approach that you took.

This text is typeset (using, for example, a Times Roman font). Notice that by typesetting, I can use **bold** and *italics* in my explanations. You can also create lists of items, either ordered, as in

How to succeed in Math 365 (in an ordered list)

1. Always typeset your code
2. Use *italics* for emphasis
3. Use **bold** when you want more emphasis
4. Indent your code - see Tip #9

These tips are good to follow too (in an unordered list)

- Always follow the tips in this homework
- Don't truncate decimals values when you know the full expansion
- Don't hardwire constants.
- Always publish your code to make sure it runs!
- Add comments to your code when specifically discussing particular code implementations choices.

```
% And this is how a code comment gets typeset.
```

Tip #5 : Don't cut and paste from a PDF

Do not cut and paste PDF text into your homework script. You run the risk of introducing hidden characters into your code or comments, and Publish may not work.

Tip #6 : Writing scripts and functions

Generally, it is not a good idea to include 'clc' or 'format' inside of scripts or functions. These are used mostly at the command line, and unless you have a good reason for including them they should be reserved for command line use only. This holds for inside of functions as well. It is also not a good idea to include a 'clear all' statement at the top of functions. The reason for this is that you may end up clearing any input arguments that are passed into the function. Even if you don't have any input arguments, there is no need

for clear all, since you have essentially a 'clear' workspace inside of the function. On the other hand, the reason for including these commands in scripts is to clear out any workspace variables that may interfere with the variable names in the script. What you may want to include in a script or function is the 'close all' command. This will close all figure windows. Note the difference between this and 'clf', which only clears the current figure window, but does not close it.

Tip #7 : Scientific notation

Be sure to take advantage of scientific notation and use it to enter physical constants that are most conveniently expressed as powers of 10. For example,

```
function tip7()

% This is the correct (and *easier*) way to input powers of 10.
S = 1e4;
b = 1e-3;
e = 5e-3;
c = 1.35e-9;
d = 0.45e10;

% instead of
S = 10^4;
b = 1*10^-3;
e = 5*(10^(-3));   % Too many parentheses!
c = 1.35*10^(-9);
d = 0.45*10^10;
```

The first method literally assigns the constant values, whereas the second method has to actually do a calculation to get the constant value.

```
end  % End of tip 7
```

Tip #8 : Indent your code!

Matlab makes it very easy to make your code readable. Use the 'indent' button on your tool bar to properly indent any selected code. This has several advantages

- It makes it easier to organize your code
- You will find bugs faster because your code will be readable.
- Logic and program flow are easier to read
- Anyone grading your code will be very appreciative.

```
function tip8()
```

Here is an example of code that is nicely indented.

```
s = 0;
for i = 1:100,
    s = s + rand(1,1);   % Sum up 100 random numbers
end
fprintf('The average value is %6.2f\n',s/100);
```

```
        The average value is   0.50
```

This code is much harder to read

```
    s = 0;
    for i = 1:100,
 s = s + rand(1,1);   % Sum up 100 random numbers
         end
       fprintf('The average value is %6.2f\n',s/100);

        The average value is   0.50

end  % end of tip #8
```

Tip #9 : Avoid code duplication

Rather than using cut and paste to duplicate large chunks of code, reorganize your code so that you can use a function instead. Anonymous functions are very handy for this.

```
function tip9()

area = @(R) pi*R^2;

% Area of a penny
area_penny = area(0.52);   % cm^2

% Area of a large pizza
area_pizza = area(30);      % cm^2

% Area of the cross section of a hot air balloon
area_balloon = area(20);    % m^2

end
```

Tip #10 : Avoid header lines that are longer than a single line (like this one) since they are hard to read (and ugly)!

Tip #11 : Use sqrt() and '\' whenever possible

Avoid using powers for simple roots and inverses.

```
function tip11()

a = 4.5;

% Square roots : Do this :
sqrta = sqrt(a);

% Not this :
sqrta = a^(1/2);   % Too many parentheses
```

```
% Inverses : Do this
b = 34.7;
invb = 1/b;

% Not this :
invb = b^(-1);

% And really avoid this!
invb = b^-1;    % even though Matlab allows it - it just looks bad!
```

For more general roots, you can use the ntthroot command. This command will generally be more accurate than using exponentiation.

For example, to get cube root of 64, do this

```
c = 64;
rootc1 = nthroot(c,3);
```

not this

```
rootc2 = c^(1/3);
fprintf('Error in nthroot : %16.8e\n',abs(rootc1-4));
fprintf('Error in c^(1/3) : %16.8e\n',abs(rootc2-4));
```

```
        Error in nthroot :   0.00000000e+00
        Error in c^(1/3) :   4.44089210e-16
```

Here is an even more dramatic example :

```
c = 125e12;
rootc1 = nthroot(c,3);   % Correct way

rootc2 = c^(1/3);    % not this way!
fprintf('Error in nthroot : %16.8e\n',abs(rootc1-5e4));
fprintf('Error in c^(1/3) : %16.8e\n',abs(rootc2-5e4));
```

```
        Error in nthroot :   0.00000000e+00
        Error in c^(1/3) :   2.91038305e-11
```

```
end
```

Tip #12 : Structure of a scientific program

A scientific program often consists of at least four main parts :

1. A section in which physical parameters and constants, i.e. radius, density, temperature, etc. are set up. ,

2. A section in which any numerical parameters, such as discrete mesh points (using, for example, linspace), convergence tolerances, time step size, etc. are set,

3. A section in which the problem at hand is solved, and

4. A section in which the results are displayed, either in a table, on a graph, or by writing the results out to a file for later processing.

The first step should be the only place where numerical values for the physical constants appear. One exception to this is when using a formula involving pi, or other well known constants that are not set by you, but somehow obtained from the computing environment.

The second step should be the only place in which numerical parameters appear. If you set N=100, use the values 'N', 'N+1', 'N-1' throughout the rest of your program, in place of 100, 101 or 99.

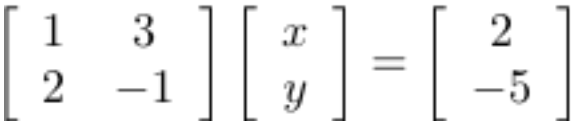
When you write the solution step for your program, you should imagine that you could run your program (within reason) with a different set of parameters by simplying changing the one place in your code where you set the value of density 'rho', or the number of mesh points 'N'. Also, the code will be much clearer if there are not hardwired constants mixed in with your solution method.

Tip #13 : Rendering equations in Publish

You can include math equations in your published documents by enclosing Latex commands between '$... $' or '$$ ... $$'. For example

$$\left[\begin{array}{cc} 1 & 3 \\ 2 & -1 \end{array}\right]\left[\begin{array}{c} x \\ y \end{array}\right] = \left[\begin{array}{c} 2 \\ -5 \end{array}\right]$$

Alternatively, you can create a PNG image by some other means (using, for example, http://www.chachatelier.fr/latexit/ on the Mac, or something similar on Windows) to produce a PNG image file, and then display it directly. For example,



Tip #14 : Plots in Publish

If you have problem getting plots to show up where you want when publishing your code, consider the following :

- Do not use '%%' inside of function definitions. Instead, use '%%%' to start a typeset section.
- You can also try using the 'snapnow' command to force Publish to produce a plot
- Use 'close all' at the beginning of each problem function to avoid plots from one problem being over-written in a second problem.

Final Tip : Publish your own code!

You can make sure that your code runs and publishes using the command

```
file = publish('homework_tips.m','pdf');
open(file);
```

I will run your program using essentially this command, and this is what I will be grading. Warning : Do not try to include this command in the file you are publishing - this will likely crash Matlab!

Published with MATLAB® R2014a