MATH 52: MATLAB HOMEWORK 3

1. M-files

So far your MatLab assignments have focused on built-in functionality. However, one of the strengths of MatLab is that you can write your own functions to solve whatever computational problem is interesting to you. The typical way of writing such a function is to store it in an m-file. M-files are simply text-documents which contain standard MatLab commands.

Example.

On the Math 52 homepage you should be able to download a file called `hello_world.m`. Save this file to the computer you use to run MatLab and open it (double-clicking it should bring up MatLab's m-file editory which will allow you to see the file). You can see that the program is fairly simple: when you run the program, it will print the phrase 'Hello World!' to your computer. Let's run the program to get a feel for how one goes about calling an m-file.

Begin by changing the current directory to the directory to which you saved the `hello_world.m` file. You can do this in lots of different ways. For instance, you can use the `cd` command as follows.

```
>> cd C:\Documents and Settings\Public\Desktop
```

Of course you'd want to replace `C:\Documents and Settings\Public\Desktop` to the directory in which you saved `hello_world.m`. You can also change the current directory in the current directory window, which is enabled by selecting 'Current directory' from the 'View' menu.

Now that you are in the directory which contains `hello_world.m`, you may run the m-file by simply typing `hello_world` into the command line. This will run the program written in the `hello_world.m` file, and your output should be:

```
>> hello_world
  Hello World!
```

This example was very simple, but even the most basic programming tools will allow you to write a number of powerful mathematical programs. Aside from the typical programming commands (such as `for`, `if`, `while` and others), there are a few other commands which are quite useful. In the `hello_world.m` file we saw the command `disp`. This command prints its argument to the screen. Another useful command is `input`, which prompts the user for information and stores her response.

One final piece of information which is useful to know is that MatLab uses the % key for comments. This means that any text which follows % on a line will be ignored by MatLab. This is typically used to insert comments into a program. For instance, the `hello_world.m` file begins with a commented section that explains what the program `hello_world` does.

Example.

Here is a slightly more complicated program which computes the sum

$$\sum_{i=1}^{n} i,$$

where the integer n is given by the user.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  This program adds the first n integers
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('This program computes the sum 1 + 2 + 3 + ... + n');
n = input('What would you like n to be? ');

sum = 0;
for i=1:n;
    sum = sum + i;
end
disp('Your sum is...');
disp(sum);
```

You can copy this code and save it as an m-file by opening MatLab's m-file editor: in the File menu, select New → M-file. After you have copied the code, save the file with a *.m extension. For the purposes of this example we will assume the program is saved to `C:\Documents and Settings\Public\Desktop\summing_integers.m` and that the current directory is set to `C:\Documents and Settings\Public\Desktop`.

To run the program we type `summing_integers`. This will prompt us to specify which value of n we would like to use. For example if I choose $n = 10$ then my output will appear as follows

```
>> summing_powers
 This program computes the sum 1 + 2 + 3 + ... + n
 What would you like n to be? 10
 Your sum is...
      55
```

You can experiment with this function by putting different values of n and m into the program. Beware: if you put in an non-integer value for n, bad things might happen!

Exercise. Modify `summing_integers` to create a function `summing_powers` that computes the sum

$$\sum_{i=1}^{n} i^m,$$

where the user provides an integer n and a real number m. Use your program to compute the sum when $n = 100$ and $m = .5$.

2. Another way to make a million dollars using Mathematics

An eccentric billionaire tells you he has buried a treasure chest full of money somewhere in Colorado. He tells you that he will answer (truthfully) as many questions as you'd like to ask about the treasure, but only if they are questions which require either a 'yes' or 'no' answer. How do you go about finding out where the loot is hidden?

Here's one clever solution. You pull out a map of Colorado and draw a straight line down the middle of the state, calling one half of the state A and the other half of the state B. You ask him 'Is the treasure buried in region A?' If he answers yes, then you know the treasure is in region A. If he answers no, you know the treasure is in region B. No matter how he answers, you now have a region which is half as large as before in which you know the treasure is buried. In a sense, you have gotten him to answer the question 'Which of these two regions contains the treasure?' with a simple yes-or-no question. Clever!

Where do you go from here? If you want to find that treasure, you should take your new half of the state and cut it in half again (we'll call these pieces C and D), and ask that rich loon 'Is the treasure buried in region C?' If he answers yes then the treasure is buried in region C, and if he answers no you know the treasure is buried in region D. Either way, you now have a region which is one-quarter as large as the state of Colorado in which the treasure is buried.

The pattern should now be clear: each time you have a region in which you know the treasure is buried you should cut that region in half and find out which of the subregions contains the treasure. If you do this long enough you will have a region which is very small (as small as you want! just keep asking questions), and you can then buy a shovel and go find the treasure.

While the scenario described above doesn't happen too frequently, this same strategy is often useful for finding equally-desirable treasures. For example, suppose you want to find a solution to $f(x) = 0$ for some crazy function $f(x)$. When $f(x)$ is a quadratic polynomial you can find these solutions using the quadratic formula, but for a random function there is no simple way for exactly computing these solutions. Instead, you must find approximate solutions by being clever. One way to find such solutions is to use Newton's Method, a technique you probably learned in single variable calculus. Another way is to use the technique described above, where the role of buried treasure is played by a solution to $f(x) = 0$ and the crazy billionaire is played by the Intermediate Value Theorem.

2.1. **Finding zeros of a function using the Intermediate Value Theorem.** The intermediate value theorem says that if $f(x)$ is a continuous function on an interval $[a, b]$, and if either

- $f(a) \geq 0$ and $f(b) \leq 0$, or
- $f(a) \leq 0$ and $f(b) \geq 0$,

then there is some value $c \in [a, b]$ so that $f(c) = 0$. Sadly, the theorem does not provide us with a way to find this magical value c. We'll use our ideas from above to use the Intermediate Value Theorem to actually find (or, more accurately, approximate) the value c which satisfies $f(c) = 0$.

Begin by finding computing the midpoint $(a+b)/2$ of the interval $[a, b]$. Since the Intermediate Value Theorem has told us there is a number $c \in [a, b]$ so that $f(c) = 0$, this number c must either

(1) be one of the endpoints of the interval or the midpoint of the interval;
(2) be in the left-hand subinterval $(a, \frac{a+b}{2})$;
(3) be in the right-hand subinterval $(\frac{a+b}{2}, b)$.

We can check to see if the first case happens by just computing the values $f(a), f(\frac{a+b}{2}), f(b)$ and seeing if they are zero or not. If they are, we are happy because we will have found a solution to $f(x) = 0$. Otherwise, we know that a zero for $f(x)$ lurks in either the left- or right-hand subinterval. How do we know which subinterval to look in? We'll use the Intermediate Value Theorem again, except this time for the subintervals $[a, \frac{a+b}{2}]$ and $[\frac{a+b}{2}, b]$. Specifically, if $f(a)$ and $f(\frac{a+b}{2})$ have different signs, then the IVT tells us that there is a solution to $f(x) = 0$ in the left-hand subinterval. Otherwise $f(\frac{a+b}{2})$ and $f(b)$ must have different signs (why?), and so there is a solution to $f(x) = 0$ in the right-hand subinterval.

The magic in this process is that we have narrowed down the places where we know a solution to $f(x) = 0$ exists: previously we only knew that a solution existed on the (big) interval $[a, b]$, but now we know which of the two (small) subintervals contains a solution. If we iterate this procedure, we will find smaller and smaller subintervals which we know contain a solution to $f(x) = 0$. Eventually these subintervals will be so small that we can just choose any point in the subinterval and it will be a good approximate solution to $f(x) = 0$.

Example.

The m-file `zero_finder.m` (available on the course webpage) implements the strategy above to approximate zeros of an input function. Here's an example where we approximate a solution to $f(x) = 0$ on $[0, \pi]$, where $f(x) = sin(x^2 + 1) - exp(x - 4)$. Notice that when putting the function into the prompt we placed the function description in single quotes. This is essential for MatLab to handle the function correctly, so don't forget to do this if you'd like to experiment with this program.

```
>> zero_finder
 Enter the function whose zeros you want to approximate: 'sin(x^2+1)-exp(x-4)'
 Enter the left-hand endpoint of your domain:  0
 Enter the right-hand endpoint of your domain:  pi
 Enter the number of decimals of accuracy:  7
 Your zero is approximately
    1.blahblahblah
```

When you read the m-file you will see that we begin with the command `format long` which ensures that answers will be displayed with more decimals than MatLab normally shows. This is just to make the output look a little nicer.

When reading the m-file you will also see that we have taken advantage of MatLab's `inline` command. This allows us to define the user's function in such a way that MatLab can manipulate it (e.g., so that MatLab can evaluate the function at various inputs).

Exercise. In the last MatLab assignment you were introduced to complex functions. In particular you saw that if a function $f(z)$ had a pole at the complex value $a + bi$, and if C was a simple closed path which had $a + bi$ in its interior, then

$$\int_C f(z) \, dz = 2\pi i \, \text{Res}_f(a + bi).$$

In this exercise you will use this result to find the poles of a complex function in a rectangular domain D.

(1) Suppose that D is a rectangular array in the complex plane; i.e., D is the collection of all complex numbers $a + bi$ so that $x_0 \leq a \leq x_1$ and $y_0 \leq b \leq y_1$. The number $x_0 + y_0 i$ is the bottom-left hand corner of D, $x_0 + y_1 i$ is the top-left hand corner, $x_1 + y_0 i$ is the bottom-right hand corner, and $x_1 + y_1 i$ is the top-right hand corner (see Figure 1).
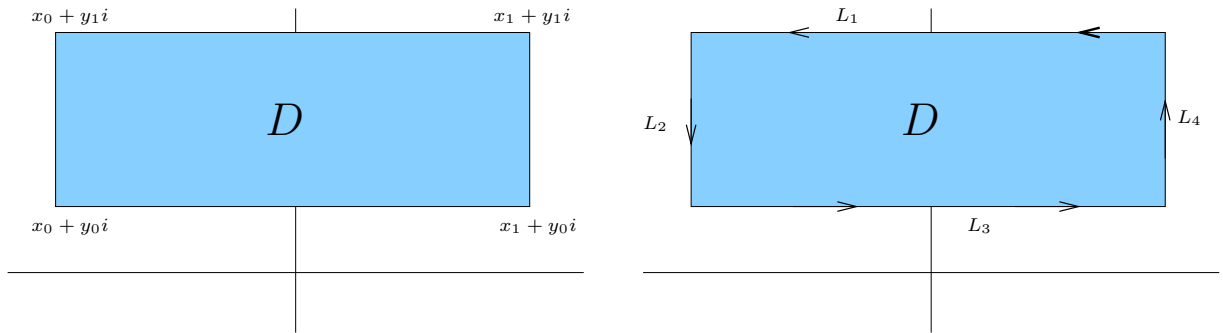
$x_0 + y_1 i$ $x_1 + y_1 i$ L_1

D L_2 D L_4

$x_0 + y_0 i$ $x_1 + y_0 i$ L_3

FIGURE 1. A rectangular domain in the complex plane

Write an m-file which computes the integral of given function $f(z)$ along ∂D, the boundary of D, in the counter-clockwise direction. You will want to use the fact that

$$\int_{\partial D} f(z) \, dz = \sum_{i=1}^{4} \int_{L_i} f(z) \, dz,$$

where the L_i are the 4 line segments which make up ∂D, parameterized in the counter-clockwise direction (again, see Figure 1). You will also want to use MatLab's built in integration function `int` which you learned in your last MatLab assignment.

(2) Write an m-file which approximates poles of a user-defined function $f(z)$ around the boundary of a user-defined rectangular domain D to a user-defined tolerance by making use of the identity

$$\int_{\partial D} f(z) \, dz = 2\pi i \sum_{z_0 = \text{poles of f}} \text{Res}_f(z_0).$$

Here's a way you might structure your program. Given the user-defined function $f(z)$ and a rectangular domain D, your program might first check that $\int_{\partial D} f(z) \, dz \neq 0$ (using, for instance, the code from your first m-file), since if this is the case the given identity will imply that there must be *some* pole in the domain D. Your program might then cut the rectangular domain into 4 equal pieces (top-left, top-right, bottom-left, bottom-right) and compute the path integral around each sub-rectangle. It might then select a sub-rectangle whose boundary gives a non-zero path integral, and thereafter replace the original rectangle with this new sub-rectangle. Iterating this procedure until the rectangle is sufficiently small (e.g., until the height and width of the rectangle are less than the prescribed tolerance) gives a good approximation for where the given function has a pole.

(3) Use your first m-file to compute the integral of $f(z) = somegrossfunction$ in the rectangular domain $D = \{a + bi : leftendpoint \leq a \leq rightendpoint, bottomendpoint \leq b \leq topendpoint\}$. (Your answer should be a non-zero number).

(4) Use your second m-file to find a pole of $f(z) = somegrossfunction$ in the rectangular domain $D = \{a + bi : leftendpoint \leq a \leq rightendpoint, bottomendpoint \leq b \leq topendpoint\}$.