Debugging in MATLAB

November 13, 2017

1 Introduction

The purpose of this worksheet is to introduce you to Matlab's debugger and give you some practical tips about debugging in general. Once you have completed this worksheet, you will be able to use Matlab's debugger and editor to find common errors in code.

2 Preliminaries

Sooner or later, you will need to use MATLAB's debugger to fix a program that you, or perhaps a colleague, has written. If the program is not yours, hopefully your colleague has commented the code liberally. Writing code comes with it the understanding that more likely than not, the program will not run correctly the first time. That said, there are ways to ensure that the time you spend debugging a program is minimized.

3 Minimizing debugging

Following are a few key pointers to get your program working correctly the first time:

1. Write down on paper what you want to program
   Too often people will begin a program by sitting down in front of the computer and typing. For most of us, that just won't work! Before you even sit down at the computer, plan your program. Make an outline. In it you should write down the final objective and the steps that you must take to accomplish the objective.

   (a) Objective: I am writing a program to calculate a multiplication table as large as the user wishes.

   (b) Preallocate memory for the table.

   (c) Loop to calculate the entries of the table.

   (d) Return.

2. Start small
   You wouldn't build a computer from start to finish and *then* check to see if all of the components were working. Write one section of code at a time, and test it to make sure that it does what you expect. Once you are satisfied with it, move on to the next section of code.

3. Write your comments while you are coding
   Do yourself a favor, and *begin* to comment your code from the *begin*ning. This will help to keep you from straying from your outline. Also, if you comment a section of code before you write it, it can help you focus on writing that section error-free.

4 Debugging

Okay. You have made a plan, tested each part, and placed comments in your code. But, when you test your program it doesn't work the way that it should. PANIC! No, just kidding. Consider yourself the newest member of a very unselective club. How you proceed now depends on the types of errors that the program has.

Syntax errors vs. Run-time errors

If your program won't even run, then you need to resolve a *syntax* error. This is usually a misspelling or a mismatched pair of parentheses or brackets. MATLAB will alert you to the name of the m-file (and corresponding function) and the location of the error. If you click on the underlined part of the error specifying the location of the problem, MATLAB will take you to that part of the code in the editor. Syntax errors are easy to fix. If you find that you have misspelled something multiple times, you can use MATLAB's *Find/Replace* utility - it is a pair of binoculars on the editor toolbar (or CTRL-H). A note on MATLAB's error map: if the error occurs inside of a function that was called by another function (for example, if the function that stores a differential equation is incorrect and you call on `ode45` to solve that equation), MATLAB will give you more than one location in which to look for the error. If the error is in a MATLAB library function and in one of your functions, chances are pretty good that you'll want to look for the error in your own code.

If your program runs but does not do what you expected, you have a *run-time* error. Run-time errors are more difficult to fix. Before you use the debugger, try the following:

- Remove a few of the semicolons in you code. This will cause those statements to print onto the command line so that you can monitor those values as they are calculated.

- Try to narrow down where the error is occurring by commenting out blocks of potentially troublesome code. This is most easily done by highlighting a section of code and selecting *Comment* under the *Text* tab. Also, for functions, consider commenting out the function declaration and temporarily adding code at the beginning to supply the necessary variable values for the function.

Using MATLAB's Debugger

You tried all of that, and the program still won't run. It's one of **those** types of problems. Okay, time to use the debugger.

You probably have some inkling of where the problem lies. If not, refer to the above suggestion to "comment out" sections of your code. Now, it's time to figure out where to target your efforts. If only you could stop your program at a certain point to "peek" inside.... Well, you can. Go to the "Editor" (this is where your M-File code is written). Now, by clicking on the edge of your editor (next to any of the line numbers that have −), the − will change to a red dot, indicating that you have set a "breakpoint" at that point in your code. When you run the program now, it will stop before it executes that line of your code. Notice that the *Command Window* prompt will change from >> to K>> to indicate that you are in Debug mode.

Once your program has been suspended, you can look inside the *Workspace* window to see your variable values. If you can't find the workspace, click on the *Desktop* tab and select *Workspace*. If you would like to run through the program line-by-line from this point, select "Step" under the *Debug* tab. If you would like to continue running the program until the next breakpoint, select "Continue" under the *Debug* tab. Once you have found the problem and wish to exit Debug mode, select the *Debug* tab and click on "Exit Debug Mode." Note that there are *hot keys* available for these tools.

5 Homework

Your task is to find and fix the errors in the file *debug1.m* provided on the APPM 2460 D2L page. Do not try to "rewrite" the entire program! Just fix the errors. Please use the MATLAB naming convention of calling the corrected files by the name of the function in the file. Use the techniques discussed in the worksheet above, and have fun! Once you have corrected the mistakes, submit a .pdf of the corrected code to D2L. Make sure to show that your code is working with an example or two.