Debugging MATLAB for beginners

Matthew Roughan

September 9, 2016

> There are two ways to write error-free programs; only the third one works.
>
> *Epigrams in Programming, 40., Alan Pearlie*

Debugging is hard to do, but even harder to learn.

MATLAB is an easy language to learn, but debugging is hard in any language.

The goal here is to get students started with the ideas of debugging. So I won't talk (much) about MATLAB's ID'S, or its built in debugger. The goal is to talk about the concepts, which can be supported by the tools, and the common errors starting students often have, which don't need a formal debugger, just some care.

First things first – the checklist

There are many tools for debugging, but they are aimed at experts. The beginner needs to learn some basic strategies before they can make good use of these tools. This short note is intended to be a kind of check list to get you started. If you have a problem in your code, go through this first, and see if you can find the problem.

1. **Read the error message.**

   Lots of students seem to find error messages hard, and so just don't read them. They may be arcane, but they contain lots of information.

   - The message will tell you which function the error was in, and which function(s) called this. That helps you trace the path to the error.
   - It will tell you what type of error it was. You can cut-and-paste key parts of the error text into Google to help decipher it.
   - Some common cases are listed below.

2. **Check spelling and typos.**

   Spelling is even more important in programming than in normal life. Computers can't tell what you intended, only what you wrote. Many errors are just simply misspellings or other typos. So check

   - Spelling (it is less important that your spelling is correct, as long as it is consistent).
   - Capitalisation (MATLAB is *case-sensitive* – you have been warned).
   - Check names haven't run together, or been broken apart by spaces.
   - Check that you are using "==" for comparison and "=" for assignment.
   - Use `||` for OR and `&&` for AND. Note the symbols are doubled.
   - Check whether you need `.^` or `^`, and similarly for other operators.
   - Check you haven't omitted operators, *e.g.,* you should replace `5x` with `5 * x`.

3. **Brackets and quotes:**

   Brackets and quotation marks need to match up *exactly*.

   - Get an editor that can show you matching brackets.
   - Use spaces to make it easy to see the matching brackets.
   - Don't overuse brackets: its often clearer, and certainly helps with debugging, to *break out* a sub-expression into a new variable, so that we can calculate (and check) it by itself, making the bracket matching must easier to see, *e.g.,* instead of

     ```
     x = sin((2*pi+phi*(1+1/y))+0.2)
     ```

     write

     ```
     tmp = 2*pi + phi*(1 + 1/y)
     x = sin( tmp + 0.2 )
     ```

     I often use `tmp` or `temp` for such variables, but only if they are *really* temporary, *i.e.,* I only need them this once.
   - If you cut-and-pasted your command from a PDF, often the quote marks (and other characters) are not ASCII quote marks – they are fancy, but they don't work.

4. **Floating point numbers**

   Remember, treat these as approximations, except in very rare cases when you are really, really sure they don't need to be. Common problems are comparisons, *e.g.,*

   - Don't test `if x == y`, instead use tests like `if abs(x-y) < small_number`

5. **Check the help file.**

   MATLAB has an extensive help system. Use it.

   - Look up a function you are using, and check you are calling it correctly. Particularly, check inputs are in the correct form.
   - Check what diagnostics the function gives you. Sometimes they give extra information through optional outputs.

6. **The error may not be where you think it is.**

   A common source of confusion is that MATLAB errors will tell you a line number. This is where MATLAB detected a problem, but the error may be many lines above, *e.g.,*

   - You have redefined one of the common MATLAB functions, which you later want to use, *e.g.,*

     ```
     sin = sin(2*pi)
     ```

     This code redefines the "sin" function, and forever afterwards calls to this function will return the variable, not the function, or cause an error.
   - Check you haven't used the same name twice for different variables. I find this happens often for indices like "i" or "j", that are being used for a `for` loop, when I try to use them for something else inside the loop.

7. **Check that the function/variable/file is what you think it is.**

   A common source of errors is calling a function called, *e.g.,* "simplex," which is meant to be the one you built. But actually, MATLAB is finding a different function or variable.

   - You can test this easily: change something simple, *e.g.,* print an extra value, and check that the change is reflected in the results when it is run.
   - Use `which` to test which it is.
   - Check your "path", and the current directory, are correct, and contain the code/data you need.
   - See the point above about redefinition as well.

8. Check version numbers: MATLAB changes over time. Make sure your version supports the things you are doing. Make sure you are looking at the correct version of the documentation.

If none of this works, you might have to be a little more clever. See below.

Common Errors

The following is a list of common error messages, and what I would do about them

- `Index exceeds matrix dimensions.`

  You have entered something like `x(i)`, where **x** is a vector of length n, and $i > n$.

  What to do: print out **x** and i and check what they are, and which one isn't what you expected.

- `Too Many Input/Output Arguments` or `Not enough Input/Output Arguments`.

  You have called a function with the wrong number of arguments. Common causes (part from the obvious) are that you aren't calling the function you think you are

  What to do: read the help on the function to check the calling sequence, and check using `which`, that you are calling the write function.

- `Undefined Function or Variable`

  The variable you are trying to use isn't defined. Often this is caused by a typo, or because you are in the wrong working directory, and the function file can't be found. A harder error is when you have a variable by the correct name, but you need a function or the other way around.

  What to do: check your spelling, and use `which` to make sure which one you are actually using.

- `Subscript Indices Must Be Real Positive Integers or Logicals`

  The index you are using to the array is not an integer, or is negative. Common causes are when we calculate an index, we sometimes aren't careful to round it off so we get a fraction, or a miscalculation gives you a non-positive number, *e.g.,* 0. Another common cause is that you try to use "logical" indexing, but the values are interpreted as zeros.

  What to do: print out the index(es) immediately before using it (them).

More advanced stuff – just a bit at least

If you follow good programming practices from the start, debugging your code will be much easier. For instance, choose good variables names, indent code appropriately, provide useful comments, modularise your code well, make sure your functions test their inputs, and so on.

> Thus spake the master programmer: "A well-written program is its own heaven; a poorly-written program is its own hell."
>
> *The Tao Of Programming*

Also, it is often better to write your (first draft) of code in the simplest way possible. Don't try to be clever when you are just learning to get it to work. Avoid weird tricks.

> Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.
>
> *Brian Kernighan*

Let's start to think about debugging more systematically. Firstly, the different types of error:

1. **Syntax errors:** these will cause your MATLAB code to fail before it even really starts. Common examples include missing, or unmatched brackets, and invalid mathematical expressions. These are usually pretty easy to debug (in MATLAB).
2. **Run-time errors:** these occur part way through your program, and often cause it to "crash". These are, perhaps, the most common obvious bugs.
3. **Logical errors:** these are the hardest, and can involve arbitrarily complex cases. The most severe cause problems like infinite loops. Others are more subtle, and might only occur in rare "corner cases". These can only be checked for using extensive testing!
4. **Warnings:** these won't stop your code, but they are worth checking, as they may indicate the cause of an error later on, or may indicate that although your code runs, the results are garbage.

The type of error determines your strategy. Syntax errors can usually be diagnosed on the spot, and run-time errors often don't take much longer. Logic errors are much harder, and require a strategic approach. The general strategy is

- reproduce it - find cases that make the problem happen
- refine it - reduce the bug to its smallest pieces (divide and conquer)
- diagnose it
- fix it
- really fix it – build test cases, so that you can make sure it never happens again.
- make sure the "fix" didn't create a new bug! Run your tests again.

Diagnosis is arguably the hardest part of this. Strategies

1. Check simple stuff first. For instance, a spelling error may not cause a run-time mistake if you manage to misspell a variable name into another variable name. Likewise, overwriting a variable (using it for two different things) may not cause a program to crash. So check this stuff (see the list above). Check it twice.

2. Decompose your code into pieces, and check each piece carefully.

3. Run the code diagnostically

   You could use a debugger to do this, but sometimes its easier just to try it out.

   - If the error is inside a function, when the code crashes, you won't be able to see the local-scope variables. So, take the code from inside the function, and put it inside a script-file (just comment out the "function" line, and define the input arguments by hand).
   - Get it to print out values as it goes (in MATLAB its easy – just leave semi-colons off the end of lines).
   - Check which code is executing. I find that one of my most common errors is when an `if-then-else` statement goes the wrong way.
   - Force your code to terminate early to check its state. You can do this with a debugger, but its easy enough to cause code to crash as well – just write "crash".
   - Count loops. When running a loop, print out a counter (this requires one extra variable in a `while` loop), and print out the variable each run through the loop. Often a problem will manifest when the loop runs the wrong number of times.

4. There is one other possibility to consider. I have had code that was correct (I believe) but caused MATLAB to crash. Thankfully that is very rare these days. However, it is very possible that a function you use has some unexpected behaviour, particularly if you get it from a third party. *Never assume that code works, just because someone else wrote it. Test it too!*

A critical part of debugging is test cases. These are crucial to debugging, even before you think your code has bugs. Good tests

1. test a single functionality in as close to isolation as possible, so when it fails, you know what is wrong;

2. each test something different; and

3. test every possible case

The last is almost impossible, so we try our best. The strategy is often to start with a few small cases, and then build up. When you find a bug, and fix it, add the new test cases to your suite.

More stuff

There is a vast array of advice for good programming practice in MATLAB, and for debugging. MATLAB has its own built in debugger to help you, for instance.

I have concentrated on the simple parts. The things that cause most starting students the most pain.

Below you will find some links to go further:

- Programming Patterns: Some Common MATLAB Programming Pitfalls and How to Avoid Them, by Loren Shure.
- Techniques for Debugging MATLAB M-files
- How to deal with errors in MATLAB
- My Top 5 Most Painful MATLAB Bugs
- Debugging for beginners, by Brian McDonald.
- Debugging matlab m-files
-

Some of them a little old, so the exact technical details may be obsolete, but the concepts are still valid.

Finally, always remember, the computer is not your friend, but it isn't your enemy either!