Debugging

Beyond the Mouse

Software Development Cycle

1. Design
2. Coding
3. Testing
4. Debugging
5. Repeat 1, 2, 3 and/or 4

What is Debugging?

- Debugging is the **art** of finding and fixing mistakes in computer programs. To be successful you need insight, creativity, logic, and determination.

> *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

Truths about Debugging

- Bugs are static – they won't run away.
- Often, the problem is simple.
- You created the bug! It's nobody else's fault - suck it up!
- Debugging is a great way to learn being self-critical. Good luck!
- Be critical – did you mean '<', '<=', '>', '>='?
- Don't panic – be systematic!
- Sleep, go for a walk, come back later.

Types of Bugs/Errors

- Syntax errors – statements that are just plain wrong. These cause fatal errors.
  - Incompatible vector sizes
  - Name hiding (redefining i, etc).
- Logic errors
  - Annoying: program crashes in some situation
  - Evil: program runs but gives wrong answer
  - Design/conceptual errors are lumped in here
- Numerical (rounding) errors

Incompatible Vector Sizes

- This happens to me all the time. Fortunately, in MATLAB you get a crash and the red text tells you where to look. In other languages, bad things might happen.
- Sometimes, you can get the sizes you need from the workspace panel or using `whos`. This lists all variables in your workspace and their sizes
- Usually, the next step is to go back to the code and insert some "size" statements to see what size the arrays really are.
  - Did you need to transpose it?
  - Why is it that size? It shouldn't be that size!

Function Name Resolution

- When MATLAB encounters a name, it resolves it via these 4 steps (in order)
  1. Checks if the name is a variable
  2. Checks if the name is a subfunction of the calling function.
  3. Checks if the name is a private function
  4. Checks if the name is a function in the directories specified by the search path
- If you define a variable with the same name as a function, you "hide" the function and cannot call it until the variable is cleared.

Logic Errors

- Many, many varieties, and some can be hard to track
  - Test errors: you used < but should have used <=
  - Indexing errors: you asked for element i but should have used i+1
  - Misunderstanding what a function does
  - Typos on assignment: you meant to type `j2 = …` but accidentally typed `j23 = …`
  - Conceptual or design errors: your plan is wrong
  - Use the variable `d` for dip angle, and also for distance (in the same program, oops)

Debugging Styles

- **echoing**: place print statements at useful points in a program (function entry, exit)
- **unit testing**: write calls to particular function, throw artificial values at it
- **exception handling**: in high level languages: sources of mistakes easier to spot
- **inline debuggers**: useful if you want to step through your code, or for memory problems
- **version control**: have a tool keep track of changes you make; roll back to bug-free code is simple (not covered here)

Debugging Styles: echoing

. . . we find stepping through a program less productive than thinking harder and **adding output statements and self-checking code at critical places**. *Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that is. More important,* **debugging statements stay with the program; debugging sessions are transient**.

From: Brian Kernighan, Rob Pike "The Practice of Programming"

Example of echoing

- When in doubt, print it out.

Example of echoing: t_debug

- Ronni Grapenthin wrote a little library called `t_debug` that lets you include define a debugging mode and print stuff out when you are in debugging mode.
- Two functions: `t_debug` and `print_debug`.
- It stores the debugging setting as a ***preference***, which can be accessed anywhere (it is global).

t_debug

```
% To allow for debugging statements that stay with the program and can be switched on and off
% (t_debug.m, print_debug.m). Here is how they work:
%
% >> t_debug on
% >> print_debug('t_debug is on');
% DEBUG: t_debug is on
%
% >> t_debug
% DEBUG state: 1
%
% >> t_debug off
% >> print_debug('This should not print');
%
% >> t_debug offff
% Warning: USE: debug(mode) with mode='on', 'off', or a numeric value to
% set debugging to a certain level. You used 'offff'.
% > In t_debug at 54
%
% >> t_debug(10)
% >> print_debug('t_debug prints up to level 10', 9)
% DEBUG: (L=9)  t_debug prints up to level 10
% >> print_debug('t_debug prints up to level 10 (this won''t print)', 11)
% >> print_debug('t_debug prints up to level 10')
% DEBUG: t_debug prints up to level 10
%
% >> t_debug
% DEBUG state: 10
```

`t_debug` demo

- Go to MATLAB…

Debugging: Unit Testing

- At the simplest:
  - write calls to your functions with artificial values
  - execute these calls at the beginning of your code, check function results
  - this helps to detect errors due to changes in functions immediately
- Many complex software packages maintain a series of automatic daily unit tests on the code as of today, so that any changes in output can be flagged immediately.

Debugging styles: Use a debugger

- A debugger is a program that lets you execute your program up until a certain point, or line by line, and then stop and examine all variables, etc.
- The MATLAB editor has a built-in debugger.
- Advantage:
  - You can examine any variable, not just the ones you thought about in advance.
- Disadvantage:
  - You can waste a lot of time stepping through code that works perfectly well.

MATLAB Debugger Demo

- Go to MATLAB…