

MATH 3341: Introduction to Scientific Computing Lab

Libao Jin

University of Wyoming

March 10, 2021



The background features a large, faint watermark of the University of Wyoming seal. The seal is circular with a rope-like border. Inside the border, the words "UNIVERSITY OF WYOMING" are written in an arc at the top, and "1886" is at the bottom. In the center, there is an open book and the word "EQUALITY" is written below it.

Lab 07: Debugging & Good Coding Practices



The background features a large, faint watermark of the University of Wyoming seal. The seal is circular with a rope-like border. Inside the border, the text "UNIVERSITY OF WYOMING" is at the top, "EQUALITY" is in the middle, and "1886" is at the bottom. In the center of the seal is an open book.

Debugging



MATLAB Debugger Commands

- `dbtype`: Display file with line numbers
- `dbstop`: Set breakpoints for debugging
- `dbstatus`: List all breakpoints
- `dbclear`: Remove breakpoints
- `dbstep`: Execute next executable line from current breakpoint
- `dbcont`: Resume execution
- `dbstack`: Function call stack
- `dbup`: Shift current workspace to workspace of caller in debug mode
- `dbdown`: Reverse `dbup` workspace shift
- `dbquit`: Quit debug mode
- `keyboard`: Input from keyboard
- `checkcode`: Check MATLAB code files for possible problems
- `mlintrpt`: Run `checkcode` for file or folder



Example: factorialIterativeBuggy.m

Given a function file factorialIterativeBuggy.m:

```
function f = factorialIterativeBuggy(n)

m = 0;
for i = n
p = p + i;
end

end
```

Issue the dbstop command and run factorialIterativeBuggy.

```
dbstop in factorialIterativeBuggy
factorialIterativeBuggy(5)
dbstep
```



Reference

Debug a MATLAB Program:

https://www.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html

or doc dbstop and then go to the bottom “Topics: Debug a MATLAB Program”.



The background features a large, faint watermark of the University of Wyoming seal. The seal is circular with a rope-like border. Inside the border, the words "UNIVERSITY OF WYOMING" are written in an arc at the top, and "1886" is at the bottom. In the center, there is an open book with a banner across it that says "EQUALITY".

Good Coding Practices



Consistent Programming Style

A consistent programming style gives your programs a visual familiarity that helps the reader quickly comprehend the intention of the code.

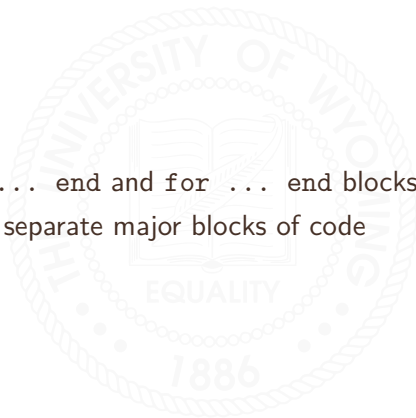
A programming style consists of

- Visual appearance of the code
- Conventions used for variable names
- Documentation with comment statement



Use Visual Layout to Suggest Organization

- Indent `if ... end` and `for ... end` blocks
- Blank lines separate major blocks of code



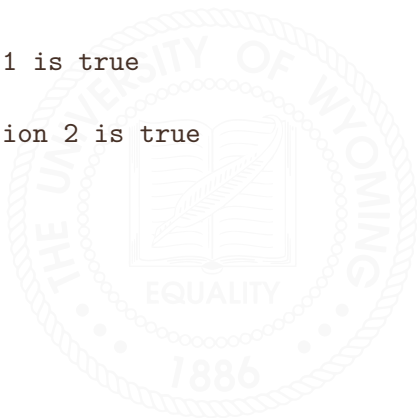
Example: Indent code for conditional structures and loops

Conditional structure:

```
if condition 1 is true
    Block 1
elseif condition 2 is true
    Block 2
else
    Block 3
end
```

Loop structure:

```
for i = 1:length(x)
    Body of loop
end
```



Shortcuts

- Windows shortcuts

- Press `Ctrl` + `A` to select all
- Press `Ctrl` + `I` to adjust indentation
- Press `Ctrl` + `R` to comment
- Press `Ctrl` + `T` to uncomment

- macOS shortcuts

- Press `command` + `A` to select all
- Press `command` + `I` to adjust indentation
- Press `command` + `/` to comment
- Press `command` + `T` to uncomment

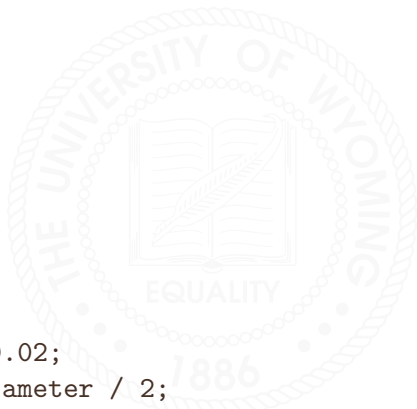


Use Meaningful Variable Names

```
d = 100;  
t = 0.02;  
r = d / 2  
r2 = r + t;
```

vs.

```
diameter = 5;  
thickness = 0.02;  
radiusIn = diameter / 2;  
radiusOut = radiusIn + thickness;
```



Follow Programming and Mathematical Conventions

Variable Names	Typical Usage
i, j, k	Array subscripts, loop counters
$i, 1i, j, 1j$	$\sqrt{-1}$ with complex arithmetic
m, n	number of rows (m) and columns (n) in a matrix.
A, B	generic matrix
x, y, z	generic vectors

Note: Consistency is more important than convention.



Document code with comment statements

- Write comments as you write code, not after
- Include a prologue that supports “help”
- Assume that the code is going to be used more than once
- Comments should be short notes that argument the meaning of the program statements. Do not parrot the code
- Comments alone do not create good code



Example: Comments at beginning of a block

```
% Evaluate curve fit and plot it along with original data
tfit = linspace(min(t), max(t));
pfit = polyval(c, tfit);
plot(t, p, 'o', tfit, pfit, '--');
xlabel('Temperature (C)');
ylabel('Pressure (MPa)');
legend('Data', 'Polynomial Curve Fit');
```



Example: Short comments at side of statement

```
cp = 2050; % specific heat of solid paraffin (J/kg/K)
rho = 810; % density of solid paraffin (kg/m^3)
k = 0.23; % thermal conductivity (W/m/C)
L = 251e3; % latent heat (J/kg)
Tm = 65.4; % melting temperature (C)
```



Defensive Programming

- Do not assume the input is correct. Check it.
- Provide default condition for a `if ... elseif ... else ... end` construct.
- Include optional (verbose) print statement that can be switched on when trouble occurs
- Provide diagnostic error messages



Preemptive Debugging

- Use defensive programming
- Break large programming projects into modules
 - Develop reusable tests for key modules
 - Good test problems have known answers
 - Run the tests after changes are made to the module

