Organizing and Debugging MATLAB Programs

Gerald Recktenwald
Portland State University
Department of Mechanical Engineering

The latest version of this PDF file, along with other supplemental material for the book, can be found at `www.prenhall.com/recktenwald`.

Version 0.9 October 10, 2000

Overview

- Rationale
- Programming Style
- Why and How of Modular Code
- Top down program design
- Basic Debugging

Rationale

Organized programs are. . .

- easier to maintain
- easier to debug
- not much harder to write

Debugging. . .

- is inevitable
- can be anticipated with good program design
- can be done interactively with MATLAB 5.x

Programming Style (1)

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 statements

Programming Style (2)

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*

```
if condition 1 is true

    Block 1

elseif condition 2 is true

    Block 2

end


for i=1:length(x)

    Body of loop

end
```


Programming Style (3)

Use meaningful variable names

```
d = 5;                  d_in = 5;
t = 0.02;               thick = 0.02;
r = d/2;                r_in = d_in/2;
r2 = r + t;             r_out = r_in + thick;
```

Follow Programming and Mathematical Conventions

Variable names	Typical usage
`i`, `j`, `k`	Array subscripts, loop counters
`i`, `j`	$\sqrt{-1}$ with complex arithmetic
`m`, `n`	End of a sequence, $i = 1, \ldots, n$, 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.

Programming Style (4)

Note: I prefer to avoid use of lower case "L" as a variable name. It looks a lot like the number "1". Which of the following statements assigns the value "1" to the lower case version of the variable "L"?

```
l = 1;        (or)        1 = l;
```


Programming Style (5)

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 augment the meaning of the program statements: Do not parrot the code.
- Comments alone do not create good code.
  - ▷ You cannot fix a bug by changing the comments

Programming Style (6)

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 statements*

```
cp = 2050;    %  specific heat of solid and liquid paraffin   (J/kg/K)
rho = 810;    %  density of liquid or 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)
```


Supporting On-line Help

- First line of a function is the definition
- Second line must be a comment statement
- All text from the second line up to the first non-comment is printed in response to

  `help functionName`

Prologue Used in the NMM Toolbox

Summary: One line description of what the function does.

Synopsis: Lists the various ways in which the function can be called.

Input: Describes each input variable.

Output: Describes each output variable.

Function Prologue

First line of the file must be the function definition.

No blank lines between function definition and first comment statement in the prologue

First line of the prologue is a terse but complete description of the function.

T and units are optional input variables as indicated by the synopsis.

```
function  rho = H2Odensity(T,units)
% H2Odensity  Density of saturated liquid water
%
% Synopsis:   rho = H2Odensity
%             rho = H2Odensity(T)
%             rho = H2Odensity(T,units)
%
% Input:   T     = (optional) temperature at which density is evaluated
%                  Default: T = 20C. If units='F' then T is degrees F
%          units = (optional) units for input temperature, Default = 'C'
%                  units = 'C' for Celsius, units = 'F' for Fahrenheit
%
% Output:  rho = density, kg/m^3 if units = 'C', or lbm/ft^3 if units = 'F'

% Notes:  Use 4th order polynomial curve fit of data in Table B.2
%         (Appendix B) of "Fundamentals of Fluid Mechanics",
%         B.R. Munson, et al., 2nd edition, 1994, Wiley and Sons, NY
```

This comment will not be printed when the user types "help H2Odensity" because it is separated from the prologue by a blank line.

Modular Code (1)

A module should be dedicated to *one* task

- Flexibility is provided by input/output parameters

General purpose modules need. . .

- Description of input/output parameters
- Meaningful error messages so that user understands the problem

Modular Code (2)

Reuse modules

- Debug once, use again
- Minimize duplication of code
- Any improvements are available to all programs using that module
- Error messages must be meaningful so that user of general purpose routine understands the problem

Organization takes experience

- Goal is *not* to maximize the number of m-files
- Organization will evolve on complex projects

Example: Built-in Bessel functions (1)

The Bessel functions are solutions to

$$z^2\frac{d^2y}{dz^2} + z\frac{dy}{dz} - (z^2 + \nu^2)y = 0$$

The Bessel function of the first kind is

$$J_\nu(z) = \left(\frac{z}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{\left(\frac{z^2}{4}\right)^k}{k!\,\Gamma(\nu + k + 1)}$$

where ν is a real number, z is complex, $i = \sqrt{-1}$ and

$$\Gamma(z) = \int_0^\infty e^{-t}t^{z-1}\,dt$$

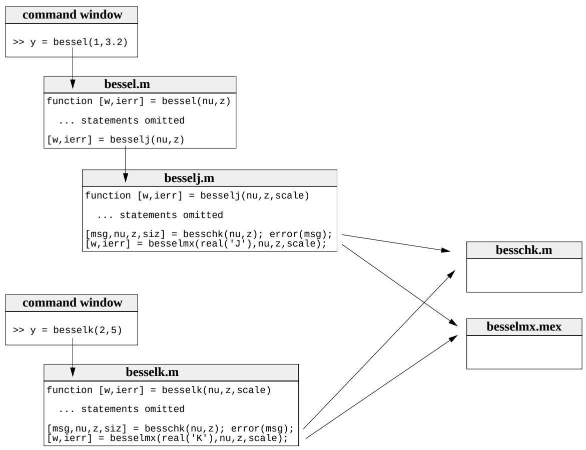
Other Bessel functions (which are also solutions to the ODE) are defined in terms of $J_\nu(z)$.

Example: Built-in Bessel functions (2)

Rather than repeat the code that computes $J_\nu(z)$ and $\Gamma(z)$, these fundamental functions are part of a core routine that gets evaluated via an interface function.

```
>> lookfor bessel
BESSCHK  Check arguments to bessel functions.
BESSEL   Bessel functions of various kinds.
BESSELA  Obsolete Bessel function.
BESSELH  Bessel function of the third kind (Hankel function).
BESSELI  Modified Bessel function of the first kind.
BESSELJ  Bessel function of the first kind.
BESSELK  Modified Bessel function of the second kind.
BESSELY  Bessel function of the second kind.
BESSLDEM Driver function for Bessel zero finding.
BESSLODE Bessel's equation of order 0 used by BESSLDEM.
```

Example: Built-in Bessel functions (3)



Defensive Programming

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

Example: H2Odensity.m

```
function  rho = H2Odensity(T,units)
% H2Odensity  Density of saturated liquid water
%
% Synopsis:   rho = H2Odensity
%             rho = H2Odensity(T)
%             rho = H2Odensity(T,units)
%
% Input:   T     = (optional) temperature at which density is evaluated
%                  Default: T = 20C. If units='F', then T is degrees F
%          units = (optional) units for input temperature, Default = 'C'
%                  units = 'C' for Celsius, units = 'F' for Fahrenheit
%
% Output:  rho = density, kg/m^3 if units = 'C', or lbm/ft^3 if units = 'F'

% Notes:  Use 4th order polynomial curve fit of data in Table B.2
%         (Appendix B) of "Fundamentals of Fluid Mechanics",
%         B. R. Munson, et al., 2nd edition, 1994, Wiley and Sons, NY

if nargin<1
  rho = 998.2;  return;    %  Density at 20 C w/out evaluating curve fit
elseif nargin==1
  units='C';               %  Default units are C
end

% --- Convert to degrees C if necessary
if upper(units)=='F'
  Tin = (T-32)*5/9;        %  Convert F to C; don't change input variable
elseif upper(units) == 'C'
  Tin = T;
else
  error(sprintf('units = ''%s'' not allowed in H20density',units));
end

% --- Make sure temperature is within range of curve fit
```



```
if Tin<0 | Tin>100
  error(sprintf('T = %f (C) is out of range for density curve fits',Tin));
end

% --- Curve fit coefficients
c = [ 1.543908249780381441e-05  -5.878005395030049852e-03 ...
      1.788447211945859774e-02  1.000009926781338436e+03];

rho = polyval(c,Tin);     % Evaluate polynomial curve fit
if upper(units)=='F'
  rho = rho*6.243e-2;     % Convert kg/m^3 to lbm/ft^3
end
```

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
- Include diagnostic calculations in a module
  - ▷ Enclose diagnostics inside `if...end` blocks so that they can be turned off.
  - ▷ Provide extra print statements that can also be turned on and off

Debugging Tools

- MATLAB version 5 (and later) has an *interactive* debugger
- The `type` and `dbtype` commands are used to list contents of an m-file.
- The `error` function prints a message to the screen, and stops execution. This provides for graceful failure, and the opportunity to inform the reader of potential causes for the error.
- The `warning` function prints a message to the screen, but does not stop execution.
- `pause` or `keyboard` commands can be used to temporarily halt execution.

Use of `keyboard` command

```
function r = quadroot(a,b,c)
% quadroot   Roots of quadratic equation and demo of keyboard command
%
% Synopsis:  r = quadroot(a,b,c)
%
% Input:     a,b,c = coefficients of  a*x^2 + b*x + c = 0
%
% Output:    r = column vector containing the real or complex roots

% See Chapter 4, Unavoidable Errors in Computing, for a discussion
% of the formula for r(1) and r(2)
d = b^2 - 4*a*c;
if d<0
  fprintf('Warning in function QUADROOT:\n');
  fprintf('\tNegative discriminant\n\tType "return" to continue\n');
  keyboard;
end
q = -0.5*( b + sign(b)*sqrt(b^2 - 4*a*c) );

r = [q/a; c/q];  % store roots in a column vector
```