2E11 Programming Style

Prof. Anil Kokaram

1 Jan 2017

See **here** for another example Matlab Style guide.

Software is one aspect of any product in the modern age. In many product development exercises, there can be large teams of Engineers contributing to the same deliverable. Whether you may be having to write control software for some mechanical device, or the software itself implements some service to the consumer, a consistent style helps your team members read your code and helps you debug as well. A good style also helps you avoid common mistakes that could lead to gnarly errors that are hard to debug. In Google for instance there are more than 20,000 engineers contributing code to the same codebase all the time. If the code that they exchange was not *readable*, teams would rapidly be unable to build on the work of others.

Many companies have agreed *styles* for their software whether it be C++, Python or Matlab. In this course we will use a simple version of the Matlab style used by many around the world and also Google. Your submitted code must obey these guidelines. There are two points of interest: programming itself and syntactical form.

Matlab Programming best practices

Programming Rules

Scripts	Allowed if the first line is `clear`
Subfunctions	Allowed
Nested functions	Not Allowed
Figure handles	Always use
Overloading keywords	Not allowed
Global variables	Not allowed

Scripts

Scripts are programs composed of sequences of MATLAB commands stored in an M-file.

Scripts are allowed only if the first line of the script starts with `clear` ***and*** `close all`***.***

Reason: Scripts are the best way to avoid having to retype many commands from the command line, and are a fast mechanism for prototyping. However, if you cascade the execution of one script with another, variables in one invocation may clobber the variables in your previous invocation. This rule ensures that scripts are self contained and start from a known state.

Subfunctions

Subfunctions are functions defined in an m-file after the first or primary function.

Subfunctions are allowed.

Reason: Subfunctions are file-local. They can be used only within the m-file they are declared in. So it enhances readability because there is no magic dependency to other files including functions you have in your directory.

Nested Functions

Nested functions are functions defined in an m-file within the first or primary function.

Nested Functions are not allowed.

Reason: Although nested functions are file-local, with scope limited to the local function around it, it encourages code that is very difficult to read. They can pollute the local namespace with variables that mean different things in different nested functions but have the same name.

Figure Handles

Figure handles are simply the argument to the figure() function. `figure(1);` creates a Figure window and subsequent graphics calls render to the handle 1. `figure` on its own creates another figure window and increments the handle index automatically.

Do not use `figure` ***without an accompanying argument.***

Reason: If you are rendering multiple GUI items into a figure window it is possible that you render into the wrong window if you do not use an argument to the function. Specifying which figure you are rendering into helps keep code readable and makes it easier to debug when graphics do not render into the correct figure window.

Overloading keywords

There are almost no reserved keywords in the Matlab language. So for instance the following code

```
>> zeros = 23.4;
```

actually makes a variable called *zeros* which has a value of 23.4.

Do not overload Matlab keywords.

Reason: Normally, *zeros(N)* creates a matrix, $N \times N$, filled with zeros. Since the code above is valid, you would have overloaded the *zeros* function to act as a variable and not as a

function! This obfuscates code and makes it quickly unreadable.

Matlab Style Rules

Style Rules

Line Length	Maximum 80 characters. Use ... to break lines.
Statements	One statement per line
Indentation	Use 2 chars of whitespace to indent code blocks and nested functions
Whitespace	Use one char around operators but not : ;) (
Naming	Use descriptive words for variables, separated by _. Use camel case for function names.
Blank lines	Use two blank lines between function definitions.
Comments	Use them

Line Length

Maximum line length is 80 characters. Use ... to break lines if you need to.

```
s = ['A very long and  alarmingly, boring string that must be ' ...
     'split in two pieces.'];
```

One statement per line

Only write one statement per line. The following code is allowed.

```
x = 1;
while x < y
  x = x + 1;
end
```

The following is NOT allowed..

```
x = 1; while x < y, x = x + 1; end;
```

Indent nested functions and code blocks

Use 2 character spaces to indent. The following code is allowed.

```
for p = 1:100,
  fprintf('p is %f\n',p);
  m = p/2;
  fprintf('m is %f\n',m);
  if (m > 20)
    fprintf('m is bigger than 20\n');
  end;
end;
```

The following is NOT allowed.

```
for p = 1:100,
fprintf('p is %f\n',p);
m = p/2;
fprintf('m is %f\n',m);
if (m > 20)
fprintf('m is bigger than 20\n');
end;
end;
```

Whitespace

Use one character of whitespace around operators, but not) (: ; . This allows your mathematical manipulations to be more easily read. The follwing is allowed.

```
x = x + 1;
y(1:10) = Spam([1, 2, 3, 4]);
q = ~p;
```

```
x == 1;
A .* B;
```

The following is NOT allowed

```
x = x+1;
y(1 : 10)=Spam([1,2,3,4]);
q=~ p;
x==1;
A .* B;
```

Naming

Variable names should be descriptive with underscores between words. Use camel case for function names. The following is good.

```
computed_velocity = pi * 16.4 - c * velocity;
new_velocity = dragCalculation(computed_velocity);
```

The following is not allowed.

```
computedvelocity = pi * 16.4 - c * velocity;
newVelocity = drag_Calc(computedvelocity);
```

Blank Lines

Use two blank lines between function definitions.

Comments

Commments should be used wherever the rationale for a line of code is not obvious. Comments must go into their own line above the line being explained. The following is a good example of the comment lines at the start of a function.

```
function [bark, woof] = DogSong(breed, pet, pat, good_dog, sit, scratch);
% DogSong -- Sing like a dog.
%
```

```
% [bark, woof] = DogSong(breed, drink, pat, good_dog, sit, scratch)
% Simulates the sequence of barks and woofs produced by a dog of
% given breed under various owner-interaction scenarios.
%
% Arguments
%
% breed: Character string identifying the dog's breed.
% pet: Optional. A 1x3 double array of petting components. Specify
%      the empty array [] to use the default value [0, 0, 0].
% pat: Logical scalar, indicating whether the dog is being patted on
%      the head
% good_dog: A 1xN cell array of character strings, each containing
%           an appreciative sentence for the dog's behavior.
% sit: Optional. Logical scalar indicating whether the dog should
%      be sitting. Default is true.
% scratch: Optional. Logical scalar indicating whether the dog is
%          allowed to scratch while singing. Default is false
%
% Return values
%
% bark: An MxN double array of barf spectra, each row representing
%       the spectral distribution in the human-audible range of one
%       dog utterance.
% woof: An Mx1 cell array of character strings, each containing a
%       human-readable rendition of the corresponding row in barf.
%
% Required toolboxes: None.
```