MATLAB Coding Style

To stipulate the provision of spaces, we use the following terminology. *Space-followed* means "followed by a space." For example, "commas are space-followed" means that every comma must be followed by a space. *Space-padded* means "followed and preceded by a space." For example, "`=` is space-padded" means that every `=` must be preceded and followed by a space.

The acronym ASAP stands either for "as short as possible," or "as small as possible," or both, depending on context.

Submissions

• Unless otherwise indicated, hard and soft copies of a submission have identical contents.

Hard copies

• Hardcopy submissions are stapled.

• Feel free to submit the printed version of just one file (e.g., a Word document) that contains all text, code, and figures.

• The order of presentation must be logical. In particular:

(a) Solutions must be presented in the same order as the questions.
(b) If `file2.m` calls `file1.m`, then `file1.m` must be presented before `file2.m`. (Mutually recursive functions are the exception; they are rare.)

Soft copies

• One directory contains all files and subdirectories. This top directory is named after your Cornell NetID, e.g., `akf6`. Compress it with `zip` or `gzip` (and possibly also `tar`).

• Figures should be in `jpeg` or `gif` format, not MATLAB `fig` format.

• When submitting work by email, the body of the email contains the `zip` file as an attachment and contains no text and no other matter. The subject line template is

```
CS<cnum>_<year><semester>_<submtype><number>
```

where `<cnum>` is the course number, `<year>` is the year in four digits, `<semester>` is `fa` for fall and `sp` for spring, `<submtype>` is `HW` for homework and `P` for projects, and `<number>` is the number of the homework of project. For example, to submit the first homework of CS 321 in Fall 2003, the subject line is `CS321_2003fa_HW1` and to submit the second project it's `CS321_2003fa_P2`. Every part of this string is case-sensitive.

Font

• Code is written in single-spaced 12-point plain Courier font (no bold, italic, etc.).

Comments

• The comment character (%) is space-padded except when it is the first character of a line, in which case it is space-followed.

• Code is not over-commented.

• Comments are ASAP. To that end, they are not necessarily full and proper sentences. When convenient, use abbreviations and contractions to make them ASAP. Articles (definite and indefinite) are to be omitted as much as possible inasmuch as the omission does not obscure the comment's meaning.

• Multi-line comments have matching indentation. For example, change

```
[x y] = f(a, b, c, d, e, g, h); % this is a very long comment
% with bad indentation
```

to

```
[x y] = f(a, b, c, d, e, g, h); % this is a very long comment
                                % with correct indentation
```

• Avoid alternating code-comment lines, as in

```
% do this
c1 = a1 + b1;
% do that
c2 = a2 + b2;
% do the other
c3 = a3 + b3;
```

Instead, write

```
c1 = a1 + b1; % do this
c2 = a2 + b2; % do that
c3 = a3 + b3; % do the other
```

(This is only an illustration; simple addition needs no comment.)

• Unless there is a compelling reason to do so, do not comment out lines of code and include them in your submission. Instead, delete them.

Expressions

Variables

• Unlike Java, C, C++, Scheme, and many other programming languages, the readability of MATLAB is diminished, not enhanced, by the use of long, self-descriptive variables. The reason is that a piece of MATLAB code is very often the implementation of a *numerical* algorithm. Such an algorithm is almost always described, if not created, on paper and in the langauge of mathematics, whose "variables" are just one character long. In light of the preceding paragraph, MATLAB variable names are not self-descriptive but are ASAP and reflect the mathematics they help implement. Break this rule if you think it would improve clarity.

• Global variables *are* long and self-descriptive. They are also fully capitalized, as in `MYGLOBAL1`.

Operators

• All arithmetic and boolean operators are space-padded, except for the exponentation operator (`^`).

• When not used to subscript arrays, commas are space-followed. The exception is when commas separate a list consisting entirely of single characters. For example, change

```
zeros(dim1,dim2,dim3)
```

to

```
zeros(dim1, dim2, dim3)
```

but

```
zeros(1,d)
```

is fine. Break this rule when you think there is a compelling reason.

• The colon operator (`:`) is neither preceded nor followed by a space.

Parentheses

• Parentheses are not used gratuitously, except when clarifying operator precedence. For example, change

```
if(a < 2)
```

to

```
if a < 2
```

and change

```
a = (b + c);
```

to

```
a = b + c;
```

But

```
a = (b + c) * d;
```

is fine.

• Expressions are not reevaluated. Create a variable instead. For example, change

```
[X Y] = meshgrid(linspace(1,n), linspace(1,n));
```

to

```
x = linspace(1,n);
[X Y] = meshgrid(x,x);
```

• Introduce extraneous variables only to improve clarity or to satisfy the previous rule. For example, change

```
g = [d a t];
ic{i} = g;
```

to

```
ic{i} = [d a t];
```

Statements

• Unless explicitly requested, MATLAB output is suppressed (i.e., terminate statements with semicolons).

• The assignment operator (`=`) is space-padded.

• Statements are appropriately indented. For example, code inside an `if` statement or a `for` loop is indented. An indentation is no less than 3 spaces.

• Break long lines and indent for clarity. For example, change

```
dt = dt + (gem .* (dum – duem .* em) + duem .* (gm – gem .* em)) ./ Dm;
```

to

```
dt = dt + (gem .* (dum – duem .* em) + ...
           duem .* (gm – gem .* em)) ./ Dm;
```

• At most one statement appears per line. The only (rare) exception to this rule is when clarity is improved by vertically aligning blocks of code. For example, change

```
[v d] = eig(x); v = normc(v);
```

to

```
[v d] = eig(x);
v = normc(v);
```

But the following is acceptable:

```
a1 =  3;  a2 = -52;
b1 = 28;  b2 =   4;
```

• The redundant else is forbidden. For example, change

```
a = 1;
if b
    a = 2;
else
    a = 1;
end
```

to

```
a = 1;
if b
    a = 2;
end
```

• Do not use a `while` loop when a `for` loop would be clearer.

• Avoid an unnecessary `else` clause by forcing the condition. For example, change

```
if iscell(chains)
    for i = 1:length(chains)
        f(chains{i});
    end
else
    f(chains);
end
```

to

```
if ~iscell(chains)
    chains = {chains};
end

for i = 1:length(chains)
    f(chains{i});
end
```

Note that the first version involves writing two calls to `f`, whereas the second doesn't.

• As much as possible, programs are parametized, not hard-coded. For example, if an assignment involves writing a script to perform a computation on only one protein, which happens to have 153 residues, the number 153 occurs only once, to define a variable, e.g., nres. Then use nres as often as you like. For another example, don't write a Monte Carlo loop this way:

```
for i = 1:1000
    Metropolis…
end
```

Instead, parametrize the loop and add a comment:

```
n = 1000; % length of Markov chain
for i = 1:n
    Metropolis…
end
```

Functions

• A function's name reflects its purpose.

• The template for a MATLAB function is:

```
function [y1, y2] = f(x1, x2, x3, x4)

% [y1 y2] = f(x1, x2, x3). Function description…
% Function description continued…
%
% x1: [m1 n1]: Description of x1…
%       Description of x1 continued…
% x2: [m2 n2 q3]: Description of x2.
% x3: [1]: Description of x3.
% x4: z = feval(x4, r): Description of x4, r, and z…
%
% y1: [r1 s1]: Description of y1…
% y2: [r2 s2]: Descripttion of y2…
```

```
Code begins here
```

Feel free to use a variation on this idea. But do not submit a function that does not include a comment explaining its purpose and its input and output arguments (and their types and sizes). A similar rule holds for scripts.

• Functions are ASAP.

• Favor more small functions to fewer big ones. This makes programs more modular and also increases the comment-to-code ratio.

• To curb the proliferation of m-files, feel free to define several functions per file. Use the following template:

```
...
previous function's last line of code


% ------------------------------------------------------------
function y = f2(x)

% Function comments, etc…
```

At least two blank lines follow the previous function's last line of code. A full line of commented dashes precedes the next function, which follows immediately.

Figures

• Figures have titles and legends, and graphs have labelled axes.