MATLAB Engineering Applications Programming Specifications

SA: Yazhuo LIU

Good programming habits can improve programming efficiency, not only make the code easy to modify, but also easy for others to understand and communicate. We have to write not only "code that can be read by the computer", but also "code that can be understood by someone else".

Here are some programming specifications that I hope you will read carefully and follow in your programming. Because some programming specifications will be used as scoring rubrics for homework, please treat them with caution.

(a) Principle

a) Correct: can accurately achieve the purpose of simulation.

b) Efficient: loop vectorization, fewer loops. Try to use less than three layers of loops.

c) Clear: develop good programming habits, the program has good readability;

d) Universal: the program is highly portable and easy to develop to avoid repetitive work.

(b) Scoring rubrics (portion)

a) Variables in English.

"*windSpeed*" is better than "*fengsu*". Cause "*fengsu*" may be "风速"、"风俗" etc.

b) Annotations in English

Chinese will be garbled on some computers without Chinese language package. Also, it is prohibited to use Russian, French, German, Japanese and other languages, because I can't understand them.

c) Variables in a mixed uppercase and lowercase form and begin with lowercase letters.

E.g. *userName*.

Note: some students like to use "underscore: _ " to separate words, for example *user_name*. It is also clear at a glance, but in MATLAB, this is not recommended, because the underscore will convert the subscript in the Tex interpreter. For example, "$maturity\_day$" will be displayed as "$maturity_day$" in the figure.

d) Structures in a mixed uppercase and lowercase form and begin with an uppercase letter,

which distinguishes variables.

E.g. *ParameterSet*

The naming of the structure should be implicit and does not need to include field names. For example, using *Segment.length* instead of *Segment.segmentLength*

e) Loop variables should be prefixed with *i* or *j* or *k*

For example, *iFiles*, *jPositions*. The reason why I don't use *i* and *j* is that *i* and *j* are imaginary numbers in MATLAB.

f) Avoid negative Boolean variable naming

For example, when naming *isNotFound*, it takes a long time to understanding ~isNotFound, right? Therefore, negative Boolean variable naming is not suggested.

g) Function name should have a specific meaning and use as few abbreviations as possible.

For example: using: *ComputeTotalWidth()* to avoid: *cptttw()*.

And *aaa()* etc. is not acceptable.

h) Appropriate tabs

Appropriate tabs can make the code easier to read.

```
for iFile = 1：nFiles
    for jPosition = 1：nPositions
        …
    end
end
```

is better than

```
for iFile = 1：nFiles
for jPosition = 1：nPositions
…
end
end
```

right?

i) All declarations of variables, constants and data structures with physical meaning must be annotated when they are declared, indicating their physical meaning and units.

Units is important, unless you want to exchange 10000USD for me for 10000KRW.

Notes:

a) In MATLAB, looping by column is faster than cycling by row.

b) Unless otherwise specified, other interfaces of MATLAB are not allowed in this course.

c) The error code can be commented out first. Do not delete it directly.

d) The use of large non-sparse matrices should be avoided.

e) Learn to use MATLAB help documents.