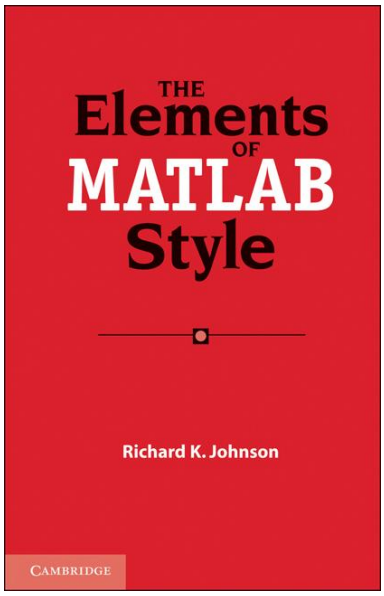
UPDATES TO THE ELEMENTS OF MATLAB STYLE



By Richard Johnson

The little red book was published in January 2011.

For a professional review of the book, see Loren on the Art of MATLAB. You can also read some user reviews on the Amazon page.

Since publication, I have received good feedback including questions, observations, and disagreements. This document consists of modifications and additions for the book that address that feedback. It also makes some additional recommendations on MATLAB style-related considerations and changes in the recent MATLAB versions through R2019a.

The numbered sections in this document contain additional or modified contents for sections in the book with the same number. The unnumbered sections contain new material.

General Principles

Some ways of writing code are better than others. It's as simple as that. The best code is correct, simple and clear. The best way to develop code is to act with care at every step. Of course the details are not always obvious.

Developers don't all agree on the best coding style and development practices. There is probably something in the book to offend everyone. If you don't like a guideline, feel free to cross it out. Just be sure that the rest of your workgroup agrees.

Naming

Variables and Parameters

56. Avoid Variable Names That Shadow Functions.

When you write code in functions, you can be more relaxed about name collisions because the variable names go away after the function completes. This is especially true when you write short functions with few variable names.

Use the Numerical Naming Style For Tall Numerical Arrays.

Provide consistency and avoid future renaming issues by using the same naming convention for both. In particular, avoid including "Tall" in these names.

Cell Arrays

Avoid Confusion With String Arrays.

Reserve names that include "string" or "str" for string arrays. Be aware that existing code may have identifiers that include "str" but do not refer to string arrays. Also existing code comments may use the word "string" but not refer to string arrays.

Tables

Use UpperCamelCase for Table Names.

The table container is a hybrid of a matrix, a structure, and an object. This naming convention is consistent with structures and objects.

Use Simple VariableNames in Tables.

Use the property VariableUnits rather than including units in VariableNames. You may also be able to use the property VariableDescriptions to simplify VariableNames.

Use the Table Naming Style For Tall Tables.

Provide consistency and avoid future renaming issues by using the same naming convention for both. In particular, avoid including "Tall" in these names.

String Arrays

Use lowerCamelCase for String Array Names.

This is consistent with the guideline for character arrays and cell arrays of character data.

Functions

68. Give Functions Meaningful Names.

Another problem with shorthand names is that they suggest a connection that may not exist. For example `gcd`, `gcf`, and `gco` are not related.

Documentation

You can use the publish feature to generate reference pages that can be integrated with the Help Browser. To help automate the task, you can write code that will publish HTML pages for all the m-files in a folder.

The techniques for, quality of, and ease of integrating custom reference pages differ among releases. For successful reference page publishing, you want to turn off the Evaluate and Include Code parameters in the Publish options. For manual publishing, you may want to redefine User Default for these settings rather than create a new recipe. That way it will be available for use with any m-file. Note that the Publish options do not travel with the m-file. You will need to re-establish them for a different installation.

Comments

97. Be Sure That Comments Agree With the Code.

If you include the class of a variable in the comments, be sure that it is the same class used in the code. This is particularly a concern when code is changes. For example to take advantage of changes in the language, structures may be converted to tables or cell arrays to string arrays, and it is important to keep the comments in sync.

115. Format Header Comments for the Help Browser

Many versions ago, MATLAB only displayed help information in the Command Window. It displayed only the first comment block of the function and was of course monospaced. Subsequent versions added links to See Also entries and pages in the Help Browser. This approach to help still exists but has been largely superseded by direct use of the Help Browser, so changes in the header format are appropriate.

119. Use the Actual Function Name Case in Comments.

In recent releases, function names in comments are displayed in the Command Window using the correct case, with bolding to make them stand out. This bolding feature also works for functions that you write. The Help Browser displays function names in correct case, with bolding in the search results.

Use the Best Descriptor for Text Arrays

Use the phrase “character vector” for a 1 x n array of characters. Avoid referring to it as a “string”, which would be a 1 x 1 string array in this case. Use the phrase “cell array of character vectors” for a cell array whose elements are character vectors. Avoid referring to it as a “cell array of strings.”

Programming

Variables and Constants

149. Use Appropriate Numerical Class Conversions.

Maintain the numerical class of variables to provide clarity and support optimizations. Avoid reusing a variable name when changing variable class.

Replace

```
alpha = double(alpha)
```

with

```
b = double(alpha)
```

Cell Arrays

Prior to release R2016b, cell arrays were the preferred way to work with character data. This is no longer true. In new code, use string arrays instead. In some cases, it may be appropriate to use cell arrays of characters for compatibility with older releases or consistency with older code.

Do Not Use Cell Arrays of Strings.

Most standard functions that accept cell arrays of character vectors do not accept cell arrays of strings. In general, cell arrays of strings do not provide any advantages over string arrays.

String Arrays

Use String Arrays

For new code, string arrays are the preferred way to work with character data. Because they are a homogenous data type, string arrays usually provide better performance and require less memory than cell arrays of character vectors. String arrays are also likely to benefit from new additions to the language.

Use string arrays rather than cell arrays for character data except for compatibility with older releases or consistency with older code.

Use String Arrays for Function Input and Output.

In general, write functions that accept input values that are character vectors, string arrays, and possibly cell arrays of character vectors. Consider writing functions that provide output in the same character type as the input.

Use strlength.

The `strlength` function provides a more intuitive result for string arrays than `length`. The `strlength` function is a better way to test for a string that has no characters than `isempty`.

Character Data

MATLAB uses all three representations for character data. For releases that support string arrays, they are preferred to cell arrays of character vectors. Some applications still require or are clearer using character vectors. There are probably no applications that still require cell arrays of character vectors.

Be aware that MATLAB sometimes changes the representation of character data that you used to define a variable. This is particularly an issue for properties of built-in classes.

The usage guidelines that applied to cell arrays of character vectors in earlier versions also apply to string arrays in newer versions.

Structures

163. Organize A Structure Based On How It Will Be Accessed.

The organization can also affect the amount of memory required. In general, fields containing arrays (the second example) are more memory efficient and provide faster access than arrays of fields.

165. Be Careful with Fieldnames.

You can also use the try-catch construct in this context.

Consider Initializing Structures.

Initialize the fieldnames but not necessarily the contents of a structure. This practice can improve performance and code clarity. Also for better performance and memory usage, don't grow the contents of structure fields incrementally.

Tables

Tables are designed for tabular representation of column-oriented variables where all variables have the same number of rows. Tables are not as flexible as structures but can provide faster performance and additional functionality. Note that some table properties have been added or renamed since the table class was introduced in R2013b.

Use a Table to Force Data Type Consistency.

Each variable (column) in a table is required to have a single data type. A table can be simpler for enforcing this level of consistency than an object would be.

Make UserData a Structure.

The UserData property of a table can be any type of variable. Making it a structure supports flexibility in the number and type of variables that it contains.

Use CustomProperties for Variable Metadata.

The CustomProperties property can hold either table or variable metadata. Your code will be easier to understand if you use the Description property or the UserData property for table metadata and reserve CustomProperties for variable metadata.

Use removevars

The `removevars` function provides a clearer way to remove variables from a table than using the empty array notation.

Statements

176. Avoid Use of `eval` When Possible.

You can use the .() notation for structures rather than `eval` with dynamic fieldnames.
Replace

```
fieldname = 'threshold';
eval(['Signal.' fieldname '= 2'])
```

with

```
Flight.(fieldname) = 2;
```

Loops

177. Initialize Loop Variables Immediately Before the Loop.

Pre-allocation remains a good idea when it is easy, for example with for loops. But starting with R2011a, it is not as important as it used to be for speed. You may be able to neglect to do it for inconvenient cases such as while loops.

Logical Functions

Use Symbolic Relational Operators.

Code that uses relational symbols like > is easier to read than code using letter abbreviations delineated by periods like `.GT.`

Input and Output

Use The Functional Form Of load.

The functional form allows the filename to be a variable, which is more flexible than the command form.

Also consider using an output variable with load.

```
x = load(filename)
```

This usage makes it obvious where the variables come from, and it may support code optimization.

Use the Latest Input Functions.

For R2019a or later read spreadsheet or text delimited data files using `readmatrix`, `readcell`, or `readtable` rather than `xlsread`. These newer functions are faster, more flexible, and better at parsing input.

Be Careful With Indexing in dlmread and csvread.

The functions `dlmread`, `csvread`, `dlmwrite`, and `csvwrite` use offset based indexing such that the first array element is 0,0. This unusual feature could easily lead to an off by one error. Starting with R2019a you can use the more intuitive functions `readmatrix` and `writematrix`.

Classes and Objects

A class is a template that can be used to create objects. The class definition includes specifications for its properties (usually data) and its methods (often functions). These properties and methods determine what you can do with the object. An object is variable that is defined by its class.

244. Follow Constructor Conventions

Useful constructor behaviors have been established through experience in other languages.

- If the constructor is called with no input variables, then return a default object. If you cannot construct a valid object, then return an error. For example, an object that calculates depreciation could be valid with a default rate, but probably not a default initial value.
- If the input is a list of property values or a parameter-value list, then return an object with those values and any required defaults. If you cannot construct a valid object from the inputs and defaults, return an error.
- If the only input variable is an object of the same class, then return it. Constructing classes with this behavior makes it easier to write methods that support appropriate flexibility in input variables without leading to problems in use.

Avoid execution errors by providing a default value for any property that is required to produce a valid object.

Exceptions, Errors and Warnings

Keep try – catch Constructs Simple.
The catch branch can include any statements, but it is best to keep them simple. Be careful that you do not create spaghetti code that obscures program flow.

Tests

Consider using xUnit.
The MATLAB distribution includes a powerful unit test framework, xUnit. You won't need it for every function, but consider using it for serious project code, especially object-oriented code.

275. Write Tests You Can Automate.
Avoid user intervention in running test code as much as possible. The `assert` function and the MException class can help with automated test execution.

Data Files

280. Make Use of Mat Files.
Consider using the version 7.3 file format. It supports larger variable sizes. Also you can use the `matfile` function to load or save portions of an array in a version 7.3 mat file. This can help deal with memory constraints.

Obsolete and Deprecated Usage

283. Use the Best Option.
Every language designer makes mistakes. MATLAB continues to develop, and some of the changes are improvements over limited or unfortunate legacy usage. Use the best subset of the language and avoid the rest.

Use histogram Rather Than hist.
This newer function provides more clarity as well as additional capability and flexibility.

Minimize Dot Relative Addressing.
It can be tempting to use the `cd` and dot technique to navigate among folders. This approach is difficult to maintain if the folder organization changes. It can also leave the focus in an undesired location when an error occurs. Use path control and `fullfile` instead.

Avoid Unnecessary Use of cd.
Changing the execution directory can be slow since it requires a re-evaluation of context.

Graphing

General

MATLAB provides some very impressive tools for data visualization, especially for 3-D data. However, its default design options are sometimes lacking, especially for 2-D data. You can often produce a better presentation by recognizing when some aspect of a plot can be improved and knowing how to change it.

The plottools feature is very convenient for interactive manipulation of plot appearance. When changing multiple plots, you can achieve a more consistent look by capturing the desired changes in a program.

MATLAB graphs are often drawn in a dry, just the facts style. Contemporary audiences are more used to quickly scanning for content than taking the time to extract meaning from graphs themselves. It is good practice to use presentation style techniques to make the meaning of graphs clear, distinct, and easy to see.

Use An Appropriate Graph Type.
Bar charts are common, especially in a business context. But they are not the most communicative graph type for some data.

- Use a line graph (plot) rather than a bar chart to show a trend.
- Use a line graph (plot) rather than a bar chart when the X axis is continuous.
- Use a line graph (plot) rather than a bar chart to show three or more variables.

Use sorting by magnitude of the principal variable if the data order is arbitrary.

Avoid pie3
3-D pie charts have no socially redeeming values. Just don't. And while you are at it, avoid `bar3`.

Choose the Baseline.
By convention, bar charts almost always have a baseline at 0. Most audiences for line and scatter plots will be fine with baselines that are not 0.

Be Careful With Axis Direction.
Our general expectation is that the X axis values increase to the right and the Y axis values increase upward. If another direction is appropriate, as with depth or geologic time, strive to make it clear.

Consider Using More Than One Plot.
Plots with a lot of data can be hard to take in. You may be able to use `subplot` and good alignment to tell the story of the data in a presentation that is easier to understand.

Use Good Tick Marks.
MATLAB sometimes presents tick marks with awkward values. You can use the axes properties XTick, YTick, and ZTick or the functions `xticks`, `yticks`, and `zticks` to specify better values.

Use Simple Tick Labels.
You can use the functions `xticklabels`, `yticklabels`, and `zticklabels` to specify better labels. You can use the functions `xtickformat`, `ytickformat`, and `ztickformat` to control the number of decimal places or include special characters.

Consider a Right Side Y Axis.
Almost all MATLAB plots have the Y axis on the default left side. If the data points near the right side of the graph are more important, you can use the YAxisLocation property of the axes to present the Y axis on the right.

Consider Unboxing.
Almost all MATLAB plots have a box outline, which is very common in technical use. You can use the axes Box property to turn off the box outline for a less formal, more contemporary look.

Make the Title a Headline.
Help the viewer easily get the message of the plot by choice of wording in the title.

Emphasize the Most Important Data.
In the default presentation using the plot function, different points and lines have the same visual impact. Feel free to use marker size or line width or color to emphasize the most important data.

Use a Large Enough Marker Size.
When combining data points and lines, make the marker diameter at least twice the line width. This makes it easier to see the points, especially in locations where the line is straight.

A marker on top of an axis can be difficult to see. This is especially a concern with the dot marker type. Make the marker size larger than the axis linewidth.

Many of the marker types are difficult to distinguish in small sizes. Make them bigger or choose different combinations.

Consider a Larger Dot.
The default dot marker (period) is smaller than the default size of the other markers. If your plot includes additional marker types, it can be useful to increase the dot size for consistency. A MarkerSize property of 20 produces a dot that is approximately the same size as the default circle.

Use at Least Two Differences For Marker and Line Styles.
When you are plotting more than one variable, make their appearance clearly different. You can use color, style, and weight to make distinctions easier to see.

Use line, xline and yline For Reference Lines.
These functions add reference lines to a plot without cluttering the plot object vector.

Development

Development practices should generally follow Agile Programming. In the 10 years since the Agile Manifesto, these practices have become widespread, leading to more projects that are developed smoothly, with less grief.

Practices

Automate the Release Process.
Aim for a one button release process. You can use a script to run all tests and many demos, publish function reference pages, and integrate the user documentation with the Help Browser system.

IDE Tools

310. Use Find and Replace.
When you select a variable name in the editor, the Code Analyzer bar will indicate other locations of the name. This feature makes it easy to see where the variable is used and where its name will be replaced.

Use Rename.
In recent MATLAB versions, you can use the Rename feature of the Editor. Rename does a better job of attending to syntax than Find, but it does not make changes in comments. If you find the Rename highlighting distracting, you can turn it off in Preferences.

311. Pay Attention to the Code Analyzer.
The former M-Lint feature is now named Code Analyzer. You can selectively turn off any Code Analyzer warnings that are distracting in the Preferences menu.

315. Use the Profiler.
In addition to studying performance, you can also use the Profiler feature to assess test coverage of code lines (whether they are executed). In recent releases, there is a convenient Coverage Report based on the Profiler results.

Generate a Code Compatibility Report.
The Code Compatibility Report flags functions and syntax that may cause problems with the installed MATLAB version. It lists functionality that has been removed, will be removed, has changed, will change, or is not recommended.

Use the Latest Development Tools.
The MathWorks programmers are constantly improving and adding to the quiver of development tools. Some recent examples are the Comparison Tool and automated renaming in the Editor.

is* function list

Delete from this list in the book:

```
isocaps
isocolors
isonormals
```

Acknowledgements

Thanks to: The MathWorks team for producing such useful software. I have made my living teaching and programming MATLAB for more than 10 years. The generous and thought provoking authors at MATLAB Central File Exchange. We all learn from each other. The book reviewers who saw the things I missed and helped make this a better book. You know who you

are. My friends and family who stayed supportive as I disappeared for hours at a time to write day after day for all too many months.

Special thanks to: Cheryl and Java, who waited patiently. John Lazenby, who introduced me to MATLAB. Loren Shure, tireless advocate for better MATLAB usage. My brother Tom, who is a better writer and programmer than I will ever be—though not in MATLAB. Lauren Cowles and Heather Bergman, my excellent and helpful editors at Cambridge University Press. Tom Keffer, my former grad student who wrote the paper that prompted this **ELEMENTS OF** book series. And of course my great clients: Department of Energy, NASA, NOAA, EPA, Bridgewater, Volvo and others.