# MATLAB CodeCount™

# Counting Standard

*University of Southern California*

**Center for Systems and Software Engineering**

December , 2016

## Revision Sheet

| Date | Version | Revision Description | Author |
|---|---|---|---|
| 5/26/2016 | 1.0 | Original Release | Matthew Swartz |
| 7/12/2016 | 1.1 | Updated complexity table | Matthew Swartz |

# Table of Contents

# 1. Definitions

1.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

1.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

1.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

1.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program. Matlab uses implicitly defined types so that there are no data declaration statements.

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile.

The following table lists the Matlab keywords that denote compiler directive lines:

| |
|---|
| import |

**Table 1 Compiler Directives**

1.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

Matlab single line comment delimiter is "%". Anything included between "%{" and "%}" is considered part of a block comment. A whole comment line may span one line and does not contain any compilable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of comments.

1.8. **Executable Line of code –** A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, ? operator, switch)
  - Iteration statements (for, while, do-while)
  - Empty statements (one or more ";")

  - Jump statements (return, goto, break, continue, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements
- An executable line of code may not contain the following statements:
  - Compiler directives
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

# 2. Checklist for source statement counts

**Physical SLOC Counting Rules**

| MEASUREMENT UNIT | ORDER OF PRECEDENCE | PHYSICAL SLOC | COMMENTS |
|---|---|---|---|
| **Executable Lines** | 1 | One per line | Defined in 1.8 |
| **Non-executable Lines** | | | |
| Declaration (Data) Lines | 2 | One per line | Defined in 1.4 |
| Compiler Directives | 3 | One per line | Defined in 1.5 |
| Comments | | | Defined in 1.7 |
| On their own lines | 4 | Not Included | |
| Embedded | 5 | Not Included | |
| Banners | 6 | Not included | |
| Empty comments | 7 | Not included | |
| Blank lines | 8 | Not Included | Defined in 1.6 |

**Logical SLOC Counting Rules**

| NO. | STRUCTURE | ORDER OF PRECEDENCE | LOGICAL SLOC RULES | COMMENTS |
|---|---|---|---|---|
| R01 | "for", "while", "parfor", or "if" statement | 1 | Count once | Loops and conditionals are independent statements |
| R02 | Statements ending by a semicolon or comma | 2 | Count once | Semicolons and commas within matrix assignments are not counted |
| R03 | Line terminated by a new line character and last symbol is not ellipsis "…" | 3 | Count once | End of command |
| R04 | Compiler Directive | 4 | Count once per directive | |

# 3. Examples

## Executable Lines

### SELECTION Statements

**ESS1 – if-else if-else and nested if statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| if <boolean expression> | if rem(4, 2) == 0 | 1 |
|    <statements> |    disp('4 is even') | 1 |
| end | end | 0 |
| | | |
| if <boolean expression> | if x > 0 | 1 |
|    <statements> |    disp ('x is positive') | 1 |
| else | else | 0 |
|    <statements> |    disp ('x is zero') | 1 |
| end | end | 0 |
| | | |
| if <boolean expression> | if x > 0 | 1 |
|    <statements> |    disp ('x is positive') | 1 |
| elseif <boolean expression> | elseif x < 0 | 1 |
|    <statements> |    disp ('x is negative') | 1 |
| . | else | 0 |
| . |    disp ('x is zero') | 1 |
| . | end | 0 |
| else | | |
|    <statements> | if x != 0 && x > 0 | 1 |
| end |    disp ('x') | 1 |
| | end | 0 |
| | | |
| NOTE: complexity is not considered, i.e. multiple “&&” or “\|\|” as part of the expression. | | |

**ESS2 – switch and nested switch statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| switch <expression> | switch input_num | 1 |
|    case <constant 1> |    case -1 | 0 |
|       <statements> |       disp ('negative one'); | 1 |
|    case <constant 2> |    case 0 | 0 |
|       <statements> |       disp (‘zero’); | 1 |
|    case <constant 3> |    case 1: | 0 |
|       <statements> |       disp (‘positive one’); | 1 |
|    otherwise |    otherwise | 0 |
|       <statements> |       disp (‘other value’); | 1 |
| end | end | 0 |

**ESS3 – try catch blocks**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| try | try | 1 |
|    <statements> |    fid = fopen(‘abc’, ‘r’); | 1 |
| catch <exception-declaration> |    d_in = fread(fid); | 1 |

| | | |
|---|---|---|
| <statements> | catch exception | 1 |
| end | rethrow(exceptioin) | 1 |
| | end | 0 |

## ITERATION Statements

### EIS1 – For Loop

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| for <index> = <start>:<increment>:<end> | for k = 1:2:24 | 1 |
| <statements> | C{k} = k * 2; | 1 |
| end | end | 0 |
| | | |
| | for x = 1:10 | 1 |
| | x | 1 |
| | end | 0 |

### EIS2 – While Loop

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| while <boolean expression> | n = 1; | 1 |
| <statements> | while prod(1:n) < 1e100 | 1 |
| end | n = n + 1; | 1 |
| | end | 0 |

### EIS3 – Parfor Loop

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| parfor <index> = <start>:<end> | parfor i = 1:length(A) | 1 |
| <statements> | B(i) = f(A(i)); | 1 |
| end | end | 0 |

## JUMP Statements

(are counted as they invoke action-pass to the next statement)

### EJS1 - return

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| return | if i == 0 | 1 |
| | return; | 1 |
| | end | 0 |

### EJS2 – break statements

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| break | if i > 10 | 1 |
| | break; | 1 |
| | end | 0 |

### EJS3 – continue statements

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| continue | if i < 5 | 1 |
| | continue; | 1 |
| | end | 0 |

## EXPRESSION Statements

### EES1 – function and procedure call

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| <function_name> ( <parameters> ) | surf(peaks) | 1 |

**EES2 – assignment statement**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| <name> = <value> | X = [1 2 3 4];<br>Y = X; | 1<br>1 |

## Compiler Directives

**CDL1 – directive types**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| import <package> | import packagename.ClassName | 1 |

# 4. Complexity

Complexity measures the occurrences of different keywords in code baseline. Below table identifies the categories and their respective keywords that are counted as part of the complexity metrics.

| Math Functions | Trig | Log | Calculations | Conditionals | Logic | Pre-processor | Assignment |
|---|---|---|---|---|---|---|---|
| polyfit | arcsin | reallog | .* | elseif | == | IMPORT | = |
| polyval | arccos | log10 | .^ | switch | ~= | | |
| interp1 | arctan | log1p | ./ | while | <= | | |
| interp2 | arcsec | logm | .\ | case | >= | | |
| polyder | arccsc | log2 | + | else | ~ | | |
| median | arctan | log | - | for | < | | |
| deconv | cos | | * | if | > | | |
| spline | sin | | ^ | | & | | |
| unmkpp | tan | | / | | \| | | |
| fzeros | sec | | \ | | | | |
| ode113 | csc | | | | | | |
| ode23s | cot | | | | | | |
| ode23t | | | | | | | |
| ode23b | | | | | | | |
| ode15s | | | | | | | |
| odeset | | | | | | | |
| floor | | | | | | | |
| round | | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| randn | | | | | | | |
| roots | | | | | | | |
| fmins | | | | | | | |
| quad1 | | | | | | | |
| trapz | | | | | | | |
| ode23 | | | | | | | |
| ode45 | | | | | | | |
| ceil | | | | | | | |
| sign | | | | | | | |
| rand | | | | | | | |
| mean | | | | | | | |
| conv | | | | | | | |
| poly | | | | | | | |
| fmin | | | | | | | |
| quad | | | | | | | |
| diff | | | | | | | |
| exp | | | | | | | |
| sum | | | | | | | |