# User Guide for FLOPS

Hang Qian

April 2015

## Introduction

Decades ago, MATLAB had a function counting the floating point operations (FLOPS) at runtime. With the introduction of a new computing engine since MATLAB 6, the function retired because it was not feasible to keep a running count of FLOPS.

Overshadowed by memory references and cache usage, FLOPS is no longer the dominant factor in execution speed on modern computers. However, given the same hardware and software platforms, it is still of interest to compare the computational complexity of two algorithms. For example, by reading the math formulae, we note that the square-root version of the Kalman filter is more computationally insensitive than the original covariance-based recursion. We would like to estimate how many additional FLOPS are needed if we switch to the square-root filter.

Instead of runtime counting, we plan to renew the function FLOPS by scanning and parsing each line of the MATLAB codes and infer the floating point operations. Similar to human visual code examination, we estimate the required mathematical operations by analyzing matrix sizes. For example, the code $C = A * B$, where $A$ is $n \times p$ and $B$ is $p \times k$, requires about $2npk$ FLOPS for matrix multiplication. Our accounting conversion is summarized in Table 1 and 2 below. Users are welcome to add or modify counting rules in the EXCEL file `ExtendedRules.xlsx`.

Our program counts FLOPS of the entire MATLAB file, either as scripts or functions. This is different from the Lightspeed MATLAB Toolbox by Tom Minka, who offers a suite of functions that count FLOPS for individual operators (say matrix multiplication) and math functions (say cholesky decomposition).

## Usage

To count FLOPS, we need three things: MATLAB codes, variable sizes and the number of execution times for each line of the codes. Follow these steps to use the program.

Step 1: Prepare your MATLAB codes in a script or function, say `fileName.m`. Try your best to make the structure of the codes as simple as possible. Only include arithmetic operations, matrix manipulations, and common math functions with one input argument. Variable types and matrix sizes cannot change anywhere in the codes. If the codes must call other functions, internalize them by subfunctions so that they can be parsed as well.

Step2: At the end of the MATLAB script or function, save all the variables in a MAT file. For example:

```
save MATfileName.mat
```

If the codes contain subfunctions, the MAT file must include variables in all functions. Use

```
save('MATfileName','-append')
```

Step 3: Profile the MATLAB codes.

```
profile on
```

(run your MATLAB codes here, say `filename(A,B,C,D)`)

```
profileStruct = profile('info');
```

Step 4: Count FLOPS by calling the function

```
FLOPS(fileName,MATfileName,profileStruct);
```

The parsing and counting results are displayed on the screen.

The standard and recommended syntax of `FLOPS` has three input arguments:

```
[flopTotal,Details] = FLOPS(fileName,MATfileName,profileStruct)
```

As for the output arguments,

`flopTotal` is a scalar that shows the total floating-point operations.

`Details` is a n-by-1 struct with three fields:

- `FLOPS`: floating-point operations of each line of codes.
- `Multiplier`: number of times each line is executed.
- `Detail`: Accounting details in a r-by-2 cell array: the first column is the operator/function name, and the second column is FLOPS for that operator/function.

Step 2 is required because we need a variable list with variable sizes to infer the FLOPS. Step 3 provides information on how many times each line of the codes are executed. It is legal, but not recommended, to skip Step 3. If you call this function by `FLOPS(fileName,MATfileName)`, the program will analyze `FOR` loops and guess how many times the codes are executes. However, the program does not know whether codes in the `IF...END` blocks are executed or not, and thus could over-estimate FLOPS.

Last but not the least, there is a convenient but risky way to use this program by providing only one input argument.

If it is a MATLAB scripts, provide the file name. For example, `FLOPS('MyScriptName')`.

If it is a MATLAB function, provide its signature, say `FLOPS('MyFunName(A,B,C)')`, where the input matrices A,B,C must exist in the base workspace.

In that case, the program tries to do the following: 1) make a copy of the codes and add a line of codes `save(...)` at the end of the newly generated file; 2) profile the new codes; and call `FLOPS` with three input arguments; and 4) delete the temporary files (if you uncomment the last two lines of the subfunction `OneStep`). Such automatic algorithm may or may not work on your computer, as it requires reading, writing and deleting files on your computer disk. **We do not know and cannot be responsible for its potential hazards to your computer. Backup your files and use the program at your own risk.**

## Accounting

Things to be counted:

- ✓ Arithmetic operations, including plus, minus, multiplication, left and right division are counted. Both scalar and matrix operations are supported. Element-by-element binary operation `bsxfun` is supported.
- ✓ Common matrix decompositions, such as `chol` and `lu`, are roughly counted by the leading terms, similarly to the Big-O accounting rules.
- ✓ Common statistics functions, such as `sum`, `mean` and `std` are counted.
- ✓ Elementary functions, such as `log`, `exp` and `sqrt`, are counted as one floating point operation.

Things not to be counted

- × Matrix generation, such as `zeros(n)`, `rand(n)`, is not counted.
- × Matrix transpose (`A'`), negation (`-A`), combination (`[A,B]`), indexing (`A(:,2)`) and masking (`A(A>1)`) are not counted.
- × Control statements, such as `for`, `if`, `switch`, etc., are not counted.
- × Relational operations (say >, >=) and logical operations (`&`,`|`,~) are not counted.
- × String operations are not counted.
- × Outsider functions called by the codes are not parsed. If the codes must call other functions, internalize them as subfunctions so that everything can be parsed. All variables must be saved in a single `MAT` file. Consider `save(fileName,'-append')`.

Table 1 illustrates our counting method for major arithmetic operations and Table 2 lists the supported statistics and math functions for which FLOPS are counted. The program `FLOPS` has a subfunction

`Rules` that specify these accounting rules. It is possible to add or modify rules in that subfunction directly, but it is not recommended. A better approach is to use the EXCEL file `ExtendedRules.xlsx` for defining new rules and overriding existing rules. See the next section for usage details.

Table 1: FLOPS for Arithmetic Operations

| Description | Expression | FLOPS |
|---|---|---|
| Matrix-matrix summation | $A_{np} + B_{np}$ | $np$ |
| Matrix-scalar summation | $A_{np} + c$ | $np$ |
| Matrix-matrix subtraction | $A_{np} - B_{np}$ | $np$ |
| Matrix-scalar subtraction | $A_{np} - c$ | $np$ |
| Matrix-matrix multiplication | $A_{np} * B_{pk}$ | $2npk$ |
| Matrix-scalar multiplication | $A_{np} * c$ | $np$ |
| Matrix-matrix dot multiplication | $A_{np} .* B_{np}$ | $np$ |
| Matrix-scalar division | $A_{np}/c$ | $np$ |
| Matrix-matrix right division | $A_{pn}/B_{nn} = \left(B'_{nn} \backslash A'_{pn}\right)'$ | $\frac{2}{3}n^3 + 2n^2p$ |
| Matrix-matrix dot right division | $A_{np} ./ B_{np}$ | $np$ |
| Matrix-matrix left division, backslash | $A_{nn} \backslash B_{np} = A_{nn}^{-1} * B_{np}$ | $\frac{2}{3}n^3 + 2n^2p$ |
| Matrix-scalar dot power | $A_{np} .\hat{} \, c$ | $2np$ |
| Matrix-scalar power | $A_{nn} \hat{} c$ | $2n^3 \cdot log2(n)$ |

Notes:

1. For the backslash operation $A_{nn} \backslash B_{np}$, the LU decomposition of $A_{nn}$ requires about $\frac{2}{3}n^3$ FLOPS, while solving a lower (or upper) triangular linear system by forward (or backward) substitution takes $n^2$ FLOPS. Therefore, the total FLOPS are roughly $\frac{2}{3}n^3 + 2n^2p$. As for the matrix-matrix right division $A_{pn}/B_{nn}$, we consider its equivalent form $\left(B'_{nn} \backslash A'_{pn}\right)'$, hence $\frac{2}{3}n^3 + 2n^2p$ FLOPS.

2. For the matrix-scalar power $A_{nn} \hat{} c$, it takes about $log2(n)$ matrix multiplications, each of which requires $2n^3$ FLOPS. Following the instructions in the help header of MATLAB 5, we compute temp1 = dec2bin(n); temp2 = length(temp1) + sum(temp1=='1') - 1; count = 2*n^3*temp2.

Table 2: FLOPS for Common Functions

| Description | Expression | FLOPS |
|---|---|---|
| Column-wise summation | sum($A_{np}$) | $np$ |
| Column-wise cum-summation | cumsum($A_{np}$) | $np$ |
| Column-wise product | prod($A_{np}$) | $np$ |
| Column-wise average | mean($A_{np}$) | $(n+1)p$ |

| | | |
|---|---|---|
| Column-wise variance | var($A_{np}$) | $4np$ |
| Column-wise standard deviation | std($A_{np}$) | $4np$ |
| Covariance matrix | cov($A_{np}$) | $4n \cdot \frac{p(p+1)}{2}$ |
| First-order difference | diff($A_{np}$) | $(n-1)p$ |
| Elementary functions | exp($A_{np}$), log($A_{np}$), sqrt($A_{np}$), sin($A_{np}$), cos($A_{np}$), tan($A_{np}$) | $np$ |
| Cholesky decomposition | chol($A_{nn}$) | $\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$ |
| LU decomposition | lu($A_{nn}$) | $\frac{2}{3}n^3$ |
| QR decomposition | qr($A_{np}$) | $2np^2$ |
| SVD decomposition | svd($A_{np}$) | $2np^2 + 2n^3$ |
| Inversion | inv($A_{nn}$) | $2n^3$ |
| Determinant | det($A_{nn}$) | $\frac{2}{3}n^3$ |

Notes:

1. The FLOPS for the above statistics and linear algebra functions are not precise. Most matrix decompositions cost $O(n^3)$ FLOPS, but the coefficient for $n^3$ and the magnitude of the lower-order terms are contingent on numeric implemenations. For example, we may take inv($A_{nn}$) as $\frac{2}{3}n^3$, but $inv(A_{nn})b_{n1}$ is not as fast as $A_{nn}\backslash b_{n1}$, which costs $\frac{2}{3}n^3 + 2n^2$. Also, some lower-order terms are omitted in the accounting. For example, to compute the determinant of $A_{nn}$, we may perform LU decomposition that costs about $\frac{2}{3}n^3$ FLOPS. After decomposition, we take the product of the diagonals, which actually costs another $2n$ FLOPS but we omit them. The listed numbers are reasonably close to the actual FLOPS produced by MATLAB 5, but it is by no means an accurate measure of the computational complexity.

## Extensions

It is understood that the internally supported functions in Table 2 is a fraction of the commonly used functions. Therefore, there is a need to define accounting rules for new functions. Also, users may override the internal rules of Table 2 by more accurate rules.

In the current folder, there is an EXCEL file `ExtendedRules.xlsx`, in which users can define how to compute FLOPS for new or existing functions.

The first column specifies the function names. The second column defines how to count FLOPS. Suppose that the first input argument of a new function is a `nrow1`-by-`ncol1` matrix and the second input argument is a `nrow2`-by-`ncol2` matrix, then we can define the number of FLOPS in terms of the variables `nrow1, ncol1, nrow2, ncol2`. If the function only has one input arguments, either `nrow1, ncol1` or `nrow, ncol` are acceptable. At most three input arguments are supported.

The internal rules can be overridden by the EXCEL file. For example, if you think sum($A_{np}$) costs $(n-1)p$ instead of our convention of $np$, then you may put `sum` as the new function name, and `(nrow-1)*ncol` as the accounting rule.

You may put some comments in the third column of the EXCEL file, but they have no impact on the codes.

If would like to adopt the counting conversion in Lightspeed MATLAB Toolbox by Tom Minka, Rename `ExtendedRulesLightSpeed.xlsx` as `ExtendedRules.xlsx`, then rules of counting FLOPS will be updated.

Figure 1 Screen Shot of the EXCEL file

| | A | B | C |
|---|---|---|---|
| 1 | Function name | FLOPS count | Notes |
| 2 | gamma | 50*nrow*ncol | Gamma function, approximated FLOPS estimation |
| 3 | mod | 3*max(nrow1*ncol1,nrow2*ncol2) | mode(A,B) equals to A - A.*floor(A./B) |
| 4 | inv | 2*nrow^3 | Override internal counting method for matrix inversion |
| 5 | | | |

## Limitations

- Variable classes and matrix sizes cannot change anywhere in the codes. A variable, once created, cannot be deleted (but its value may change as long as it does not undermine matrix decomposition, say `chol`). A `MAT` file that stores all variables is required.
- Outsider functions called by the codes are not parsed.
- Complex numbers are not supported.
- Sparse matrices are not supported.
- Real and dense matrices with no special structures (such as symmetric, banded, triangular etc.) are assumed in linear algebra operations.
- Regular usage of math and statistics functions is assumed. Usually only one input argument is supported. Special syntax, such as LU decomposition on a non-square matrix, is not supported. However, you may redefine the accounting rules in `ExtendedRules.xlsx` if you call those functions with more input augments.

## Examples

In the folder, there are two examples. One is a MATLAB script `exampleScript` and the other is a MATLAB function `exampleFun`. Run `example.m` to see the results.

The script `exampleScript` tests FLOPS counting on arithmetic operations, matrix decompositions and user-supplied counters.

The function `exampleFun` is an econometric application that estimates a Bayesian PROBIT model by Markov Chain Monte Carlo. The function contains a subfunction `TN_RND`, which samples truncated normal variates. Variables of both functions are saved in a `MAT` file. Note that statistics functions `normcdf` and `norminv` are used. However, the program did not define counting rules for `normcdf` and `norminv`. Therefore, it is left for the users to decide whether and how to count them in `ExtendedRules.xlsx`.

About the author:

Hang Qian is a Ph.D. of Economics, graduated from Iowa State University. He is a senior software developer of computational finance. He is also an independent researcher working on Bayesian statistics and financial time series analysis.

Questions and comments on this software can be sent to

matlabist@gmail.com