Computational Performance of MATLAB and Python for Electromagnetic Applications

Alec Weiss
Department of Electrical Engineering
Colorado School of Mines
Golden, Colorado
aweiss@mines.edu

Atef Elsherbeni*
Department of Electrical Engineering
Colorado School of Mines
Golden, Colorado
aelsherb@mines.edu

***Abstract*—MATLAB and Python are two commonly used scripting languages for prototyping electromagnetic problems today. Each of these languages provides access to computationally efficient functions allowing a user to easily run many math heavy problems with minimal programming. In this paper we will discuss the usage of MATLAB and a variety of libraries in Python capable of running these efficient computations. Tests will be run in both languages to compare both CPU and GPU computations. The runtimes of a variety of problems using each of these platforms will also be compared for a variety of mathematical operations typically used in electromagnetic problems. Finally, a simple angle of arrival calculation using conventional beamforming will be performed to show these speeds on a realistic problem.**

Keywords—Computational electromagnetics, GPU programming, MATLAB, python.

I. INTRODUCTION

Modern high-level programming languages such as MATLAB and Python provide an easy way to quickly create software for electromagnetic simulations and data processing. These languages excel by abstracting the difficulties and verbosity of lower level languages such as FORTRAN or C. This abstraction comes at the expense of computational efficiency. MATLAB and python libraries such as Numpy and Numba are designed specifically to increase the efficiency by calling more efficient C and FORTRAN subroutines.

Acceleration techniques in MATLAB for computational electromagnetic (CEM) problems has been previously studied in various papers [1,2]. Previous works have also compared MATLAB and Python in a variety of areas such as the usage for linear algebra [3] and general usability [4] in comparison to MATLAB. Previous work has not compared MATLAB and Python for usage with complex numbers with basic operators ranging from addition to complex exponentials which is important as these datatypes and operators are prevalent in a variety of CEM applications.

This work covers the comparison of MATLAB and Python for complex number calculations that may be seen in typical electromagnetic problems. Each language will be tested with variety of libraries and programming techniques. Initial tests run and compare speeds when performing a single operation such as (e.g., $a + b$) and a small collection of operations (e.g., $a * (b + c)$) on a set of data providing a basis for a large variety of electromagnetic problems. Results comparing a realistic angle of arrival (AoA) simulation using conventional beamforming will also be tested to provide a more realistic CEM application.

*Adjunct Professor, Department of Elec. & Computer Eng., King Abdulaziz University, Jeddah, Saudi Arabia.

II. INTRODUCTION TO PYTHON AND MATLAB FOR CEM

A. MATLAB

An inherent benefit of MATLAB is that all required commands are built in and do not require the importing of libraries. This can make the language easier to learn and implement for an inexperienced programmer but comes at the cost of flexibility. While many ways exist to perform computations in MATLAB, this work will focus on standard usage of built-in functions without any special optimizations. Code can also easily take advantage of graphics processing units (GPUs) using the gpuArray command. MATLAB Results were obtained using MATLAB R2018a.

B. Python

Unlike MATLAB, python libraries must be explicitly imported. While this adds an extra layer of difficulty to the language, it also provides the flexibility of using a variety of libraries. This work will test a few of the most popular libraries for vectorized computation on the CPU. These libraries are Numpy/Scipy and Numba. Numpy and Scipy are a set of commonly used libraries for performing many vectorized and linear algebra operations in a similar way to MATLAB where once the variables are declared, the multiplication operator will perform elementwise multiplication with optimized subroutines. Numba extends upon the capability of Numpy by allowing for user defined vector functions and precompiled sections of code. Like MATLAB, code can also easily be extended to a GPU using the CuPy library. Python Results were obtained with Python 3.7.3, Numpy 1.16.5, Scipy 1.3.1, and Numba 0.44.1.

III. OPERATOR TESTING

Initial speed comparisons were done with testing of single and multi-operator arithmetic. By testing these more generic situations, it is possible to estimate the runtimes of a variety of CEM problems. Each of these tests was run with both single and double precision complex data types.

Tests with a single operator were run for both single (using the single() command) and double precision complex values using MATLAB and Python. The tests were run for typical arithmetic operators (+,-,*,/) along with a complex exponential, matrix multiply, and summation. Additional

Submitted On: September 10, 2020
Accepted On: September 10, 2020
https://doi.org/10.47037/2020.ACES.J.351166

testing was also performed with LU decomposition, FFT, sparse matrix multiplication, and the equation $c = a + b * exp(a)$ where a and b are complex matrices. All tests were run on a Intel Xeon E5-2620 processor with 128GB of RAM. The runtimes in seconds from these tests for double and single precision are shown in Table and II, respectively. It should be noticed that for most of the tested cases, using either Numba or Numpy provides very close runtimes to MATLAB. In some cases, such as FFT, combined computations, and sparse matrix multiplication Python outperforms the similar routines in MATLAB.

Table I. Double Precision Complex Operation Runtimes (seconds)

Operation	MATLAB	Python (Numpy)	Python (Numba)
Add	0.2249	0.2814	0.2774
Subtract	0.2280	0.2745	0.2847
Multiply	0.2281	0.2706	0.2927
Divide	0.2322	0.4667	0.2845
Matrix Multiply	8.0504	7.5575	N/A
Exponential	0.2974	3.1985	0.3177
Sum	0.0169	0.0515	N/A
LU Decomposition	2.9076	4.4278	N/A
FFT	0.2113	0.1345	N/A
Combined	0.5750	3.8899	0.3292
Sparse	4.2567	3.0242	N/A

Table II. Single Precision Complex Operation Runtimes (seconds)

Operation	MATLAB	Python (Numpy)	Python (Numba)
Add	0.1135	0.1459	0.1512
Subtract	0.1136	0.1481	0.1538
Multiply	0.1180	0.1504	0.1567
Divide	0.1275	0.4082	0.1538
Matrix Multiply	3.7282	3.5604	N/A
Exponential	0.2127	2.9446	0.1575
Sum	0.0086	0.0287	N/A
LU Decomposition	1.5741	2.6686	N/A
FFT	0.1218	0.0783	N/A
Combined	0.3639	3.2805	0.1609
Sparse	N/A	1.8862	N/A

IV. REALISITC AOA EXAMPLE

Angle of arrival calculations are performed in a variety of EM applications to estimate the direction from which waves are impinging upon an antenna array. While many algorithms exist for this calculation (e.g., [5]), a basic example of conventional beamforming can be used for speed comparison because it contains complex elementwise multiplication, exponentials, matrix multiplication, and summations. The conventional beamforming equation with frequency domain data for AoA estimation can be written as:

$$\sum_{e=1}^{E} W(e)S(e,f)e^{k(f)\mathrm{r(e)}},$$

where E is our total number of elements, $W(e)$ is the weighting on each of the elements, and $e^{k(f)r(e)}$ is the steering vector. $S(e,f)$ is the received complex data at each element. For this problem incident plane waves were synthetically impinged upon a planar array allowing simulation of what an antenna array of that shape and size may measure. The runtimes in seconds for double and single precision beamforming on a 10x10 element planar array with a synthetic incident plane wave at 45 degrees show what MATLAB is having a slight performance advantage over Python using Numba as shown in Table III.

Table III. Beamforming Runtimes (Seconds)

Mean Runtime (Seconds)	MATLAB	Python (Numpy)	Python (Numba)
Double	0.6448	1.8860	0.7776
Single	0.2779	1.4079	0.3270

V. CONCLUSIONS

We have shown the runtime comparisons for a variety of operations on complex numbers in MATLAB and two Python libraries that are typically used in CEM applications using CPUs. Similar analysis conducted on GPUs will be presented at the conference. While in many of the cases MATLAB outperforms Python, Python comes very close in most operations and in some cases even outperforms MATLAB. Python comes with the added benefits of being completely free along with having many other libraries and acceleration techniques beyond Numpy and Numba to compete with the runtimes that MATLAB can achieve.

REFERENCES

[1] A. J. Weiss, A. Z. Elsherbeni, V. Demir, and M. F. Hadi, "Using MATLAB's Parallel Processing Toolbox for Multi-CPU and Multi-GPU Accelerated FDTD Simulations," vol. 34, no. 5, p. 7, 2019.

[2] M. Capek, P. Hazdra, J. Eichler, P. Hamouz, and M. Mazanek, "Acceleration Techniques in Matlab for EM Community," in 2013 7th European Conference on Antennas and Propagation (EuCAP), 2013, pp. 2639-2642.

[3] J. Unpingco, "Some Comparative Benchmarks for Linear Algebra Computations in Matlab and Scientific Python," in 2008 DoD HPCMP Users Group Conference, 2008, pp. 503-505.

[4] J. Ranjani, A. Sheela, and K. P. Meena, "Combination of NumPy, SciPy and Matplotlib/Pylab - A Good Alternative Methodology to MATLAB - A Comparative Analysis," in 2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT), 2019, pp. 1-5.

[5] P. Vouras, et al., "Gradient-Based Solution of Maximum Likelihood Angle Estimation For Virtual Array Measurements," in 2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2018, pp. 1257-1261.