Emile Goulard
uID: u1244855
CS3200 - Assignment 6 Report
10/26/2022

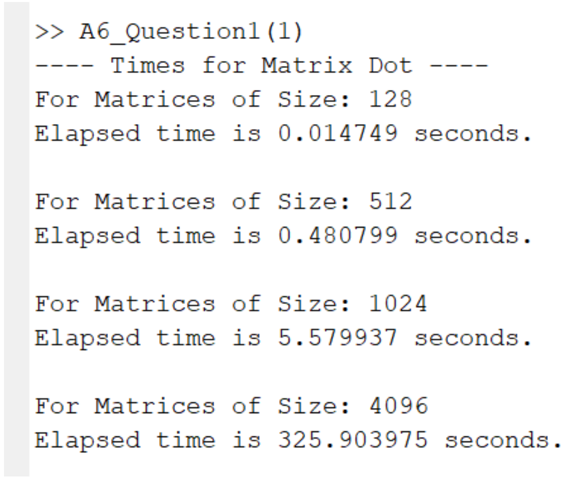
Question 1

For measuring the times of each matrix class provided, open the “A6_Question1.m” file in matlab and make sure you’ve also download “MatMatClass.m” inside the same directory. Refer to the README file for more information.

a) Essentially, the methods I mainly used for calculating the execution times was the built in ‘tic’ and ‘toc’ variables in MATLAB. First, I created variables A and B which store an empty array full of zeros for the inserted matrix size. From there, I surrounded the iterative code provided with beginning with tic and then ending it with toc for each Matrix class. Once that was done, I copied each class code and put it into the “MatMatClass.m” helper such that running “A6_Question1.m”, given the correct ‘figureNum’ parameter, will run any of those with the proper matrices of equal dimensions from 128, 512, 1024, and 4096.

After running each class, these were the execution times recorded upon first running the code:

Figure 1



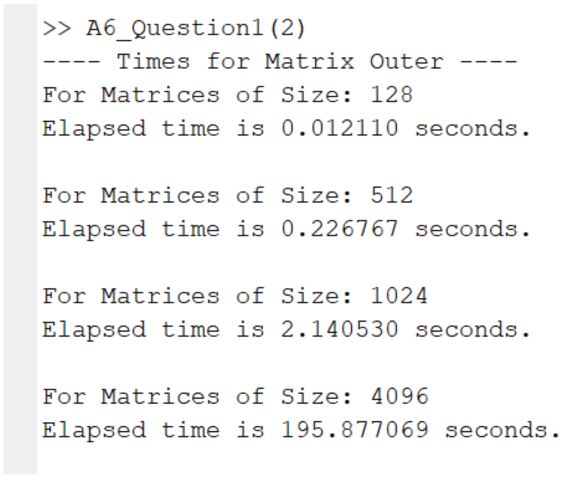
```
>> A6_Question1(1)
---- Times for Matrix Dot ----
For Matrices of Size: 128
Elapsed time is 0.014749 seconds.

For Matrices of Size: 512
Elapsed time is 0.480799 seconds.

For Matrices of Size: 1024
Elapsed time is 5.579937 seconds.

For Matrices of Size: 4096
Elapsed time is 325.903975 seconds.
```

Figure 2



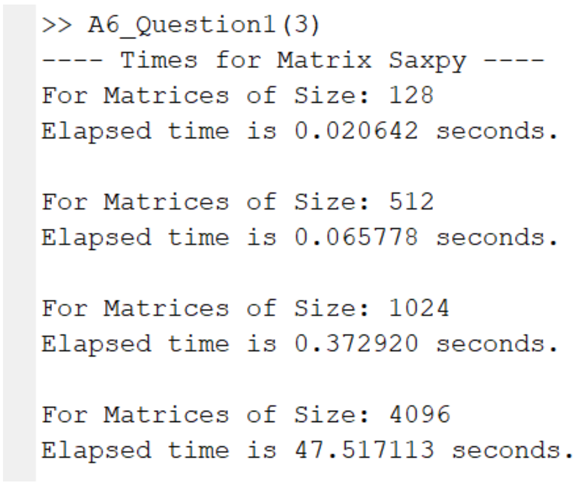
```
>> A6_Question1(2)
---- Times for Matrix Outer ----
For Matrices of Size: 128
Elapsed time is 0.012110 seconds.

For Matrices of Size: 512
Elapsed time is 0.226767 seconds.

For Matrices of Size: 1024
Elapsed time is 2.140530 seconds.

For Matrices of Size: 4096
Elapsed time is 195.877069 seconds.
```

Figure 3



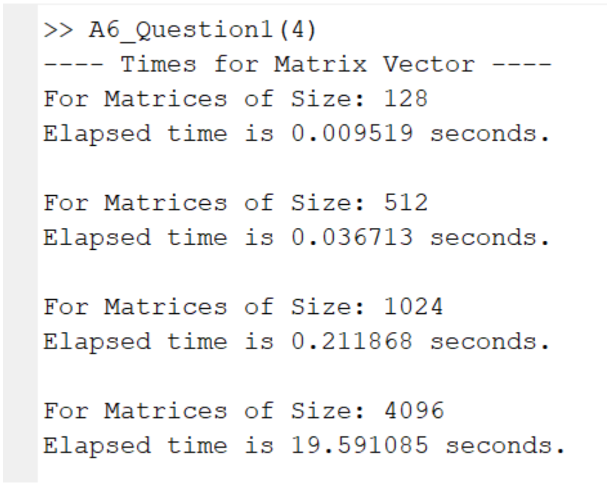
```
>> A6_Question1(3)
---- Times for Matrix Saxpy ----
For Matrices of Size: 128
Elapsed time is 0.020642 seconds.

For Matrices of Size: 512
Elapsed time is 0.065778 seconds.

For Matrices of Size: 1024
Elapsed time is 0.372920 seconds.

For Matrices of Size: 4096
Elapsed time is 47.517113 seconds.
```

Figure 4



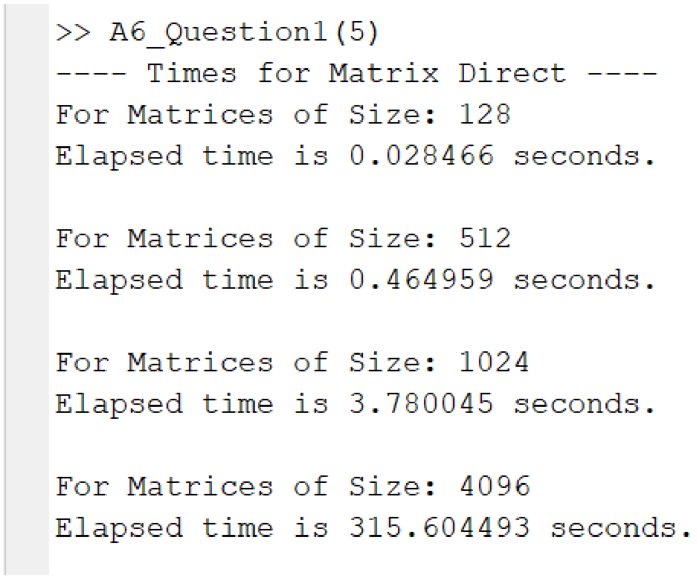
```
>> A6_Question1(4)
---- Times for Matrix Vector ----
For Matrices of Size: 128
Elapsed time is 0.009519 seconds.

For Matrices of Size: 512
Elapsed time is 0.036713 seconds.

For Matrices of Size: 1024
Elapsed time is 0.211868 seconds.

For Matrices of Size: 4096
Elapsed time is 19.591085 seconds.
```

Figure 5



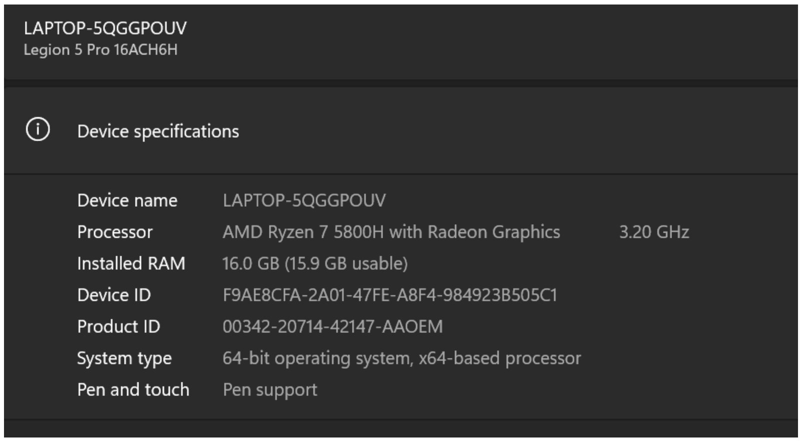
```
>> A6_Question1(5)
---- Times for Matrix Direct ----
For Matrices of Size: 128
Elapsed time is 0.028466 seconds.

For Matrices of Size: 512
Elapsed time is 0.464959 seconds.

For Matrices of Size: 1024
Elapsed time is 3.780045 seconds.

For Matrices of Size: 4096
Elapsed time is 315.604493 seconds.
```

Additionally, for context, these are my machine specifications:



LAPTOP-5QGGPOUV
Legion 5 Pro 16ACH6H

Device specifications

Device name	LAPTOP-5QGGPOUV	
Processor	AMD Ryzen 7 5800H with Radeon Graphics	3.20 GHz
Installed RAM	16.0 GB (15.9 GB usable)	
Device ID	F9AE8CFA-2A01-47FE-A8F4-984923B505C1	
Product ID	00342-20714-42147-AAOEM	
System type	64-bit operating system, x64-based processor	
Pen and touch	Pen support	

b) From my findings, the times show patterns of proper increasing as the matrices grow larger. The fastest times were from matrix multiplying through a Vector Matrix with a final time of 19.59 seconds and the slowest one was matrix multiplying through a Dot Matrix having a final time be 325.90 seconds. When comparing to the Direct Matrix times, the Dot Matrix was most similar. I'm a little surprised however that Dot Matrix was still slightly slower even though when analyzing the code at face value for both classes, Direct Matrix is O(n^3) and Dot Matrix is O(n^2). Yet, I think this is because of how matlab iterates arrays given a ':' is signaled to search all values in a column or row. The rest of the values are much faster than the Direct Matrix Multiplication which is expected since they have far more intuitive iterations and are less computationally expensive.

c) I think there are many discrepancies in the execution times here to account for. Firstly, my hardware is high end compared to the computers in the CADE Lab, so that might have an affect on the execution times by a few hundreths decimal places. This also carries into how many background programs I have open, for instance, when running the times, I had Google Chrome open which takes up CPU space. Additionally, running the code multiple times, despite cleaning the cache of tic toc and other variables within those classes, will result in more and more inaccurate times. So running these programs in some iterative loop might have a slight affect on the outcome since my computer has a different clock rate compared to other computers. Because of all these types of discrepancies, I suggest running programs based on some commonality among different peers. I think we should've all ran code through the CADE Lab computers since they have the same hardware specifications. This could lead to more accurate, albeit, averaged results among different students.

<u>Question 2</u>

To run code for iterative solvers that solve A1x=b1 and A2x=b2, simply open "A6_Question2.m" in MATLAB. Refer to the README file for more information.

The methods used in parts a and b started with creating the loaders and then implementing the iterative solvers for the Jacobi, Gauss-Seidel, and Successive Over-Relaxation. To create the loaders, I loaded the A1 and A2 text file into a sparse matrix using the 'spconvert' function. Then for b1 and b2, I scanned their text files and then called a 'cell2mat' function on it to convert to a double array. I then referred to all source code provided by the sources under the 'Sources' section in this report, using those matrices that I read from in order to solve A1x=b1 and A2x=b2.

Part c) **IMPORTANT: All reported values are located in the "A6_PartC_Sheet.pdf" attached to this zip file. The number of values/iterations printed justifies attaching it separately, so please follow that file to look at recorded values when reading the rest of my report.**

i) The Jacobi and Gauss-Seidel Methods converged to the same solution for the same number of iterations. The Successive Over-Relaxation Method didn't converge to the same solution and actually had more iterations to my surprise. For A1x=b1, Jacobi and Gauss-Seidel Methods converged to 774.42813 after 2 iterations and the Successive Over-Relaxation Method converged to 2523.429582 after 6 iterations. For A1x=b1, the shift between values to the Successive Over-Relaxation is nearly 4 times the amount from Jacobi and Gauss-Seidel Methods. I think the reason why this is the case are two factors: the scalar relaxation factor and the convergence tolerance because my convergence tolerance was the same for Jacobi and Gauss-Seidel Methods. Upon testing some of these variables, I noticed that the number of iterations decreased and the values decreased as well, however, this might mean that this method is more accurate to the true solution. For A2x=b2, Jacobi and Gauss-Seidel Methods converged to 141.143311 after 122 iterations and the Successive Over-Relaxation Method converged to 150.117667 after 134 iterations.

ii) As mentioned earlier, the fewest number of iterations were both the Jacobi and Gauss-Seidel Iterators for A1x=b1 with only 2 iterations. This might be because of the tolerance set to compute the convergence error.

iii) The Iterative solver that converged the fastest in terms of CPU time was the Gauss-Seidel Method with 1.938834 seconds. It was very close to Jacobi Method with 1.966398 seconds which makes sense because the Gauss-Seidel Method is supposed to be faster than the Jacobi with fewer iterations as well. However, in my case, I still found that the number of iterations between both are the same which is odd. It might be because of the tolerance being 0.00001.

iv) To be honest, I cannot figure out a way to solve the direct solution for both A1x=b1 or A2x=b2 even after thoroughly searching for ways to solve linear systems in MATLAB. This is because, for my inputs, I cannot compare sparse matrices of different sizes which is unfortunate. I'd be willing to talk about it during office hours eventually.

Sources

1. https://en.wikipedia.org/wiki/Jacobi_method
2. https://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method#:~:text=In%20numerical%20linear%20algebra%2C%20the,a%20system%20of%20linear%20equations.
3. https://en.wikipedia.org/wiki/Successive_over-relaxation
4. https://my.eng.utah.edu/~cs3200/Iterative_Solvers.pdf