# CENG 371 - Scientific Computing

Fall 2022

Homework 2

Anıl Utku Ilgın
2260073

December 5, 2022

## Question 2

Note: I couldn't implement the "Crout's Algorithm" recursively, it's only implemented in iterative version I wrote based on the algorithm at the wikipedia page in references, hence I didn't include it in my plots.

**a) Compare the algorithms in terms of their total run times and in terms of the plots of their relative errors.**

**a1) Runtime Comparison**

Run time of both Pickett's Charge and Shermann's March algorithms are given in Figure

(1) and Figure (2) respectively.

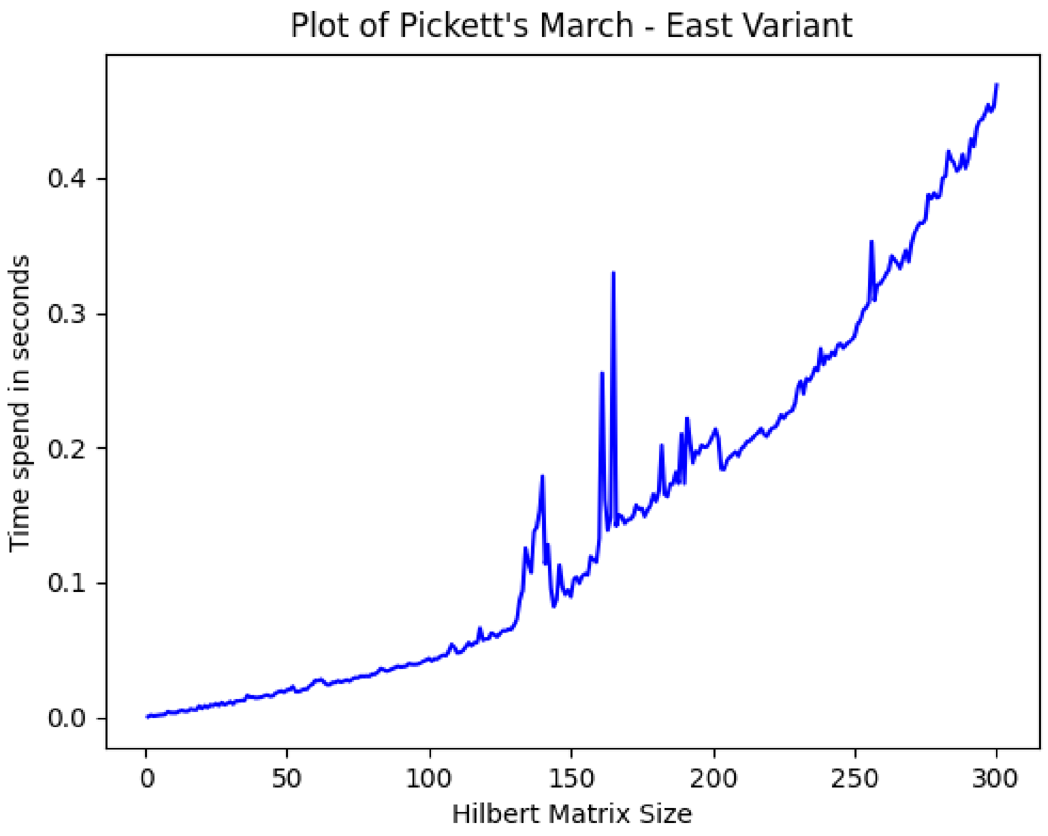

Figure (1): Plot of Pickett's Charge - East variant (Matrix Size vs Time Spend)

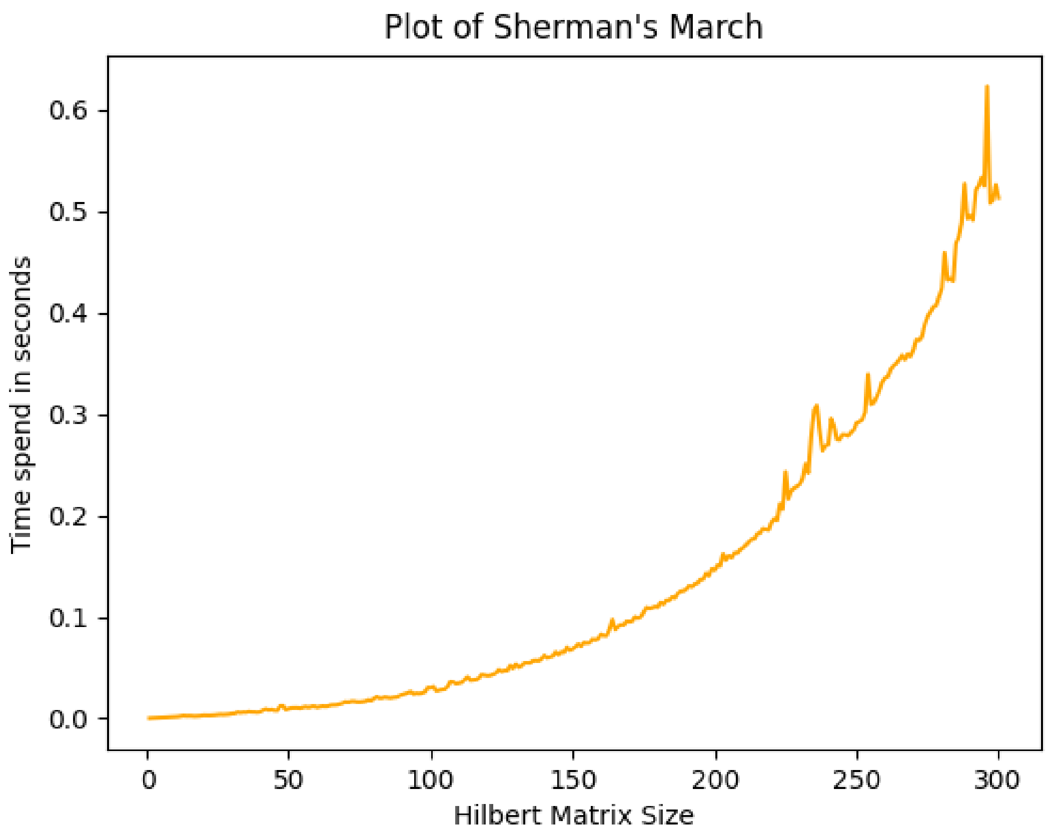

Figure (2) : Plot of Shermann's March (Matrix Size vs Time Spend)

As seen in figure (1) and (2) respectively, Pickett's Charge algorithm is acting slightly faster than Shermann's March algorithm, as n grows large. Reason behind this is there is more matrix multiplication and inversion operation in Shermann's algorithm than Pickett's algorithm. This slight difference is putting a heavy toll on calculations as n grows, and especially after size of 250 both inverse and matrix multiplication operations are influencing the cost of operations quite heavily.

**a2) Relative Error Comparison**

Relative errors of both Pickett's Charge and Shermann's March algorithms are given in

Figure (3) and Figure (4) respectively.

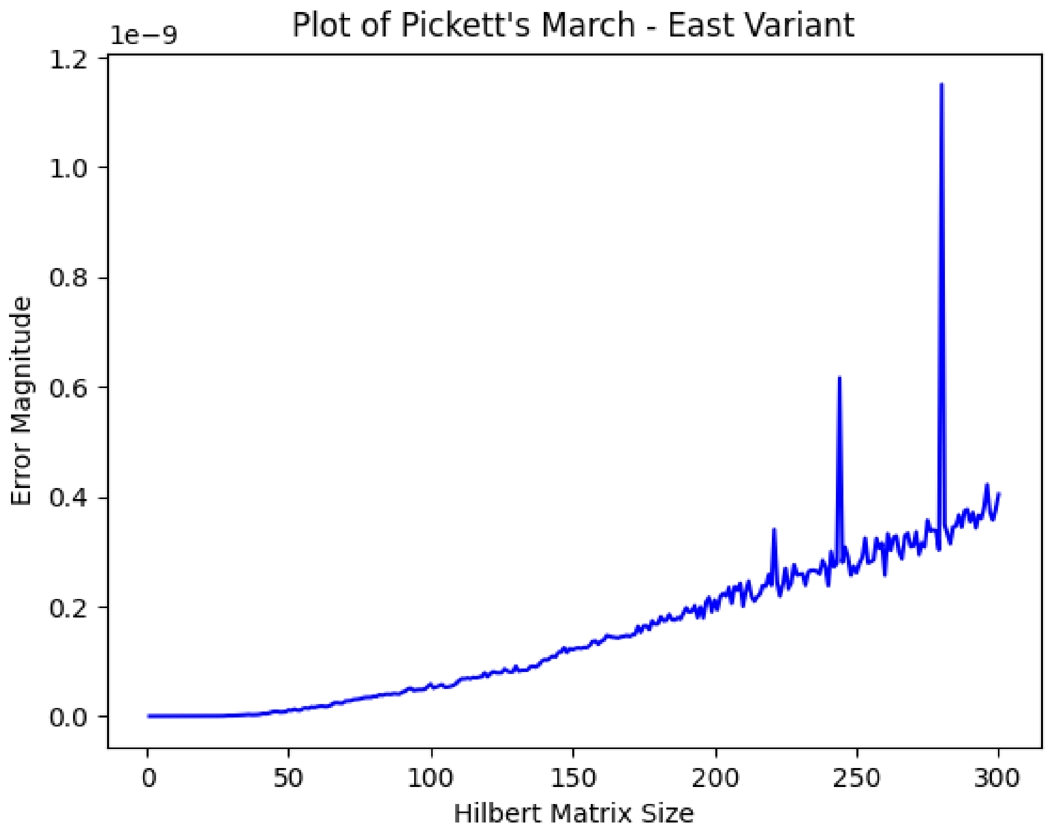

Figure (3): Plot of Pickett's Charge - East Variant (Error Magnitude vs Hilbert Matrix Size)

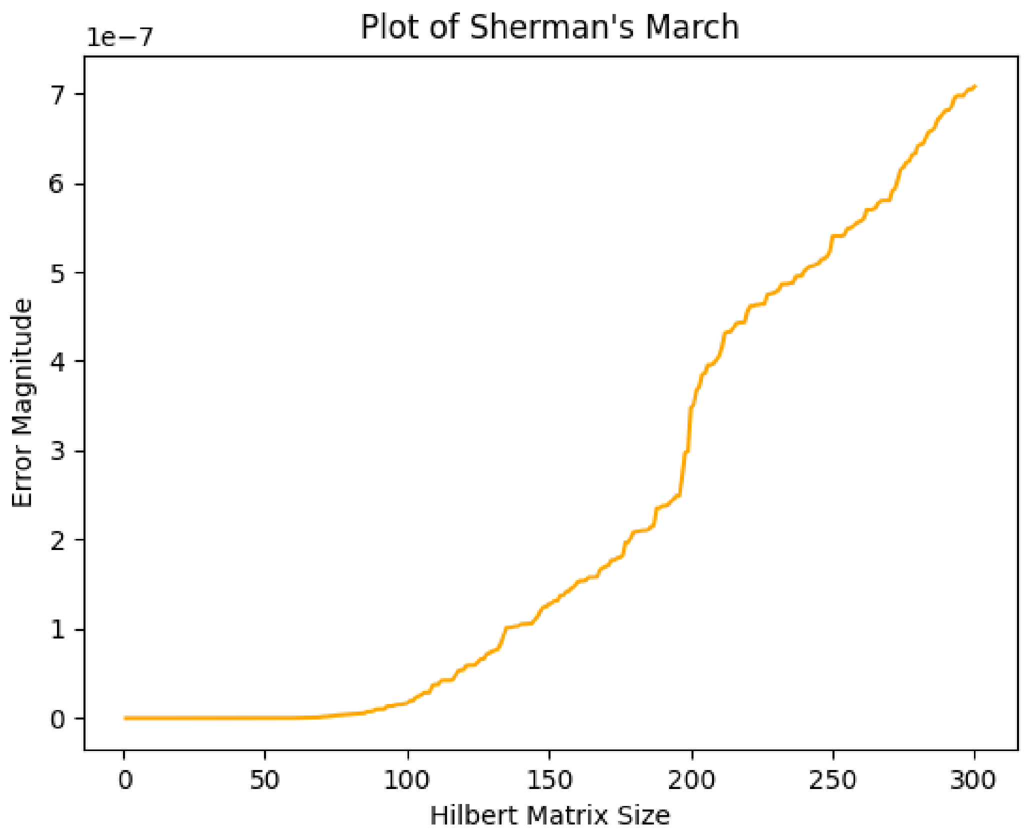

Figure (4): Plot of Shermann's March (Error Magnitude vs Hilbert Matrix Size)

As seen in Figure(3) and (4), Pickett's Charge algorithm is more accurate than the Shermann's March algorithm by a factor of 500-600 times (measured between peaks). The reason behind this error difference is that Pickett's Charge is doing less multiplication, difference and inversion operation than the Shermann's algorithm, hence there are less rounding errors accumulating in the Pickett's version.

I couldn't implement Crout's algorithm, however since it's accumulating the error in a separate sum and eliminating it like the compensated sum in the iterative algorithm, I think it should have the least error among the algorithms or at least equal to Pickett's.

**a) Do the algorithms you have implemented successfully (up to numerical errors) factorize any square matrix? Explain.**

Yes I think they do factorize square matrices successfully. I checked the L and U parts of the factorized results separately, and they fulfill the requirements to be lower and upper triangular matrices respectfully. In addition, Since their relative errors are below 1e-7, multiplication of them represents the original Hilbert matrices accurately. However, they cannot factorize **any** square matrices since they include inversion operations.

Shermann's march algorithm fails if the first row's first column of a matrix is (A[0][0]) ~0 (zero) . Reason is that after algorithm pops back from the deepest recursive step, this line:

```
u_1k = np.matrix(np.matmul(np.linalg.inv(L_11), a_1k.transpose()))
```

will yield a matrix multiplication as INF * a_1k^T, hence the algorithm fails.

In the Pickett's Charge algorithm, the same reason applies. Due to this line:

```
u_1k = np.matrix(np.matmul(np.linalg.inv(L_11), a_1k))
```

if L_11 has values that inverses are close to INF, the algorithm fails.

My implementation of Recursive Crout fails to factorize matrices so I cannot comment on them. However, as I looked in the iterative version, I see that there is a division operation to L[j][j], which could yield INF values, therefore it might also fail at factorizing matrices which could cause that.

## References

https://en.wikipedia.org/wiki/LU_decomposition

https://en.wikipedia.org/wiki/Crout_matrix_decomposition

G.W.Stewart, *Algorithms Volume I:Basic Decompositions*

John A. Gunnels, Robert A. van de Geijn, F*ormal Methods For High-Performance Linear Algebra Libraries*