

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE



EEN-300 INDUSTRY ORIENTED PROBLEM

REPORT

C++ Implementation of various Matrix Operations and Applications related to Electrical Engineering

Guided by:

Prof. Dheeraj K Khatod, Department of Electrical Engineering

Prepared by:

Aayush A Chaturvedi 16115002

Adarsh Kumar 16115009

Aviroop Pal 16115033

Samarth Joshi 161151099

Abstract

Matrix operations hold one of the prime location in the world of scientific computing. Along with that, engineering fields especially like electrical engineering demand highly efficient matrix operations tools. Thus it is critical that these are covered in all computationally powerful languages like C++, Python, MATLAB etc. It might not always be convenient to use system heavy software packages like MATLAB for small usages, given that the licensed version has a whooping amount per month/year. Also, all C++ implementations of the various matrix operations, although contain the efficient implementation, but are very large and cumbersome to use. Thus, we took this project, and attempted to make a user friendly library in C++ containing many useful matrix operations. In an attempt to justify the usage, we have solved two real life electrical engineering based problems using our matrix operations library.

Matrices operations implemented

- *Multiplication in the natural way*

The normal matrix multiplication algorithm goes by the definition of matrix multiplication. To compute the value of each of the new element we take linear time. Since there are a quadratic number of new elements we take overall $\tilde{O}(n^3)$ time complexity.

- *Multiplication using Strassen's Multiplication Algorithm*

The Strassen's Multiplication Algorithm breaks the problem of matrix multiplication recursively into seven parts. Natural recursion would require eight parts. Thus, in this we get the time complexity of multiplication to be $\tilde{O}(n^{\log_2 7}) \sim \tilde{O}(n^{2.81})$.

- *Non negative integral power of a matrix*

In general, we can compute the non-negative power 'p' of any number, say 'a' in $\tilde{O}(p)$ time per multiplication. However, using divide and conquer paradigm, we can effectively reduce this to $\tilde{O}(\log_2 p)$ per multiplication. So when coupled with Strassen's Multiplication, we can get an overall $\tilde{O}(\log_2 p * n^{2.81})$ instead of $\tilde{O}(p * n^3)$ time complexity.

- *Fractional power of a matrix*

The fractional power has been found by diagonalisation.

- *Finding eigenvalues and eigenvectors, using QR Method*

The QR Method is an iterative method to find out the eigenvalues of a given matrix. It forms two matrices Q and R, one which is triangular. Then it forms $A' = R * Q$ which iteratively converts to a triangular matrix with eigenvalues at the diagonals.

- *Matrix inversion using the standard adjoint determinant method*

By definition, the inverse of a matrix is the adjoint of the matrix divided by its determinant. This is the most basic approach and has a much higher time complexity $\tilde{O}(n^5)$.

- *Matrix inversion using Gauss-Jordan Inversion*

Gauss Jordan elimination aims at reducing a $n \times n$ matrix into identity matrix of the same order by elementary row or elementary column operations. The parallel running matrix gives the inverse. The overall time complexity is still $\tilde{O}(n^3)$.

- *Matrix inversion using LU Decomposition*

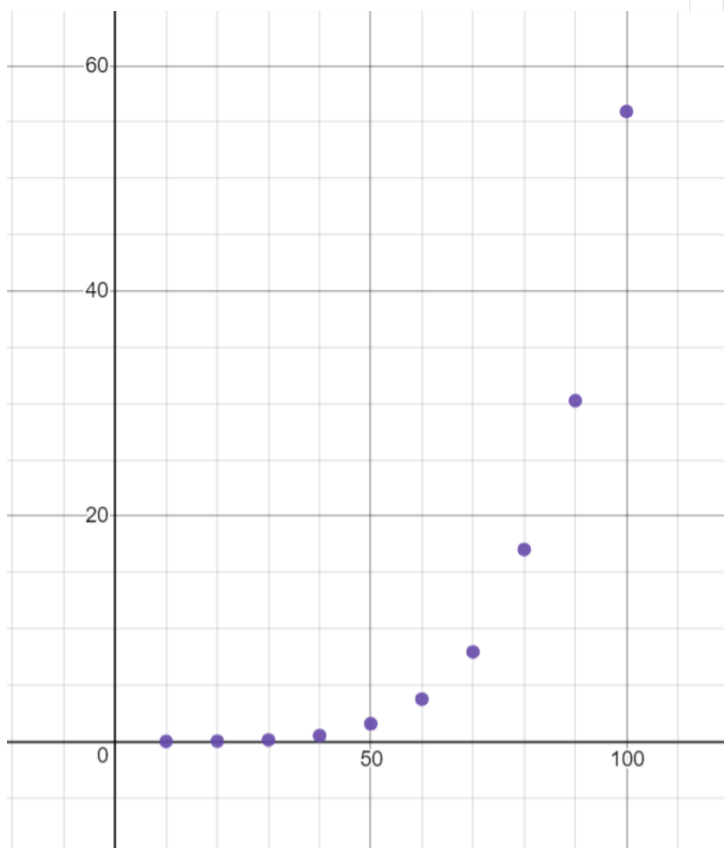
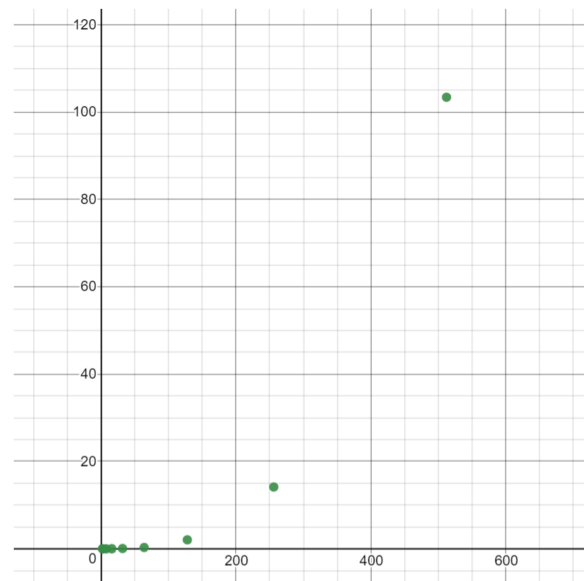
LU Decomposition is a method in which the input matrix is decomposed into two matrices L and U, both of which are triangular. There is a separate proof that every invertible matrix can be converted in $L \cdot U$ form. Since solving with a triangular matrix is far more easier by Gauss elimination, it can be used to compute the inverse in efficient time.

Runtime Analysis for various algorithms

The runtime analysis of each code has been done using **input size n on the horizontal axis** and **time taken in seconds in the vertical axis**. This is same for all the graphs below.

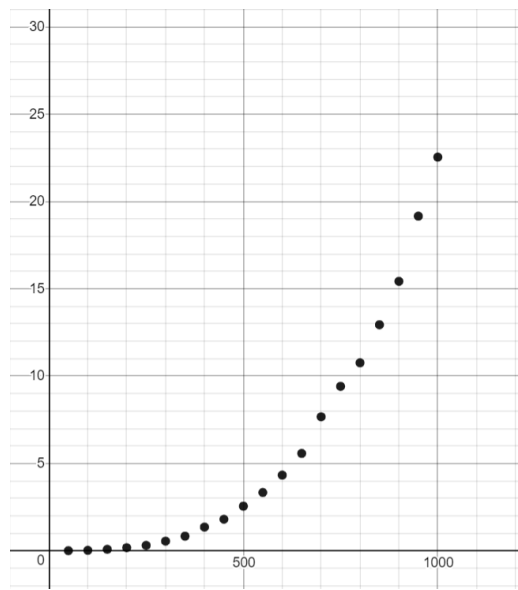
All these run times have been recorded using Intel^(R) Core™ i7-8550U CPU at 1.8GHz.

Strassen's Multiplication:

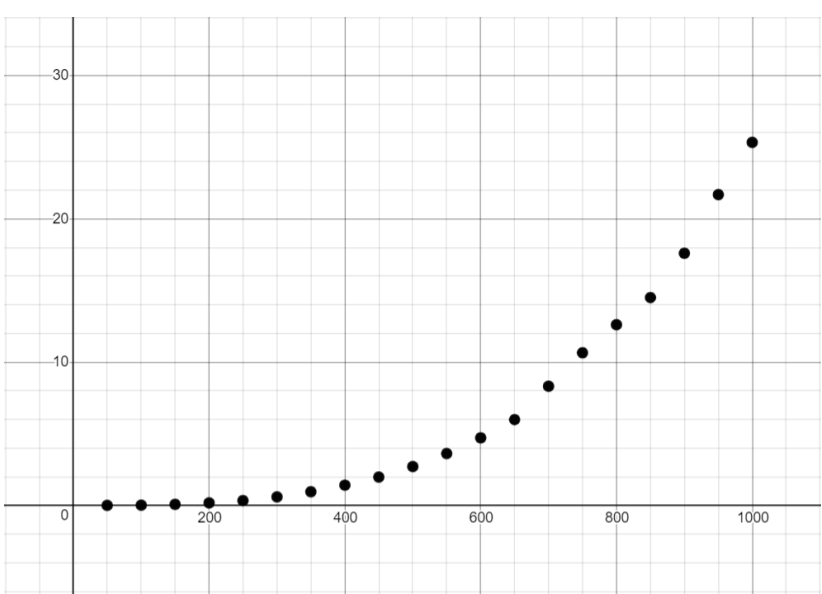


Inverse using Adjoint Determinant Method:

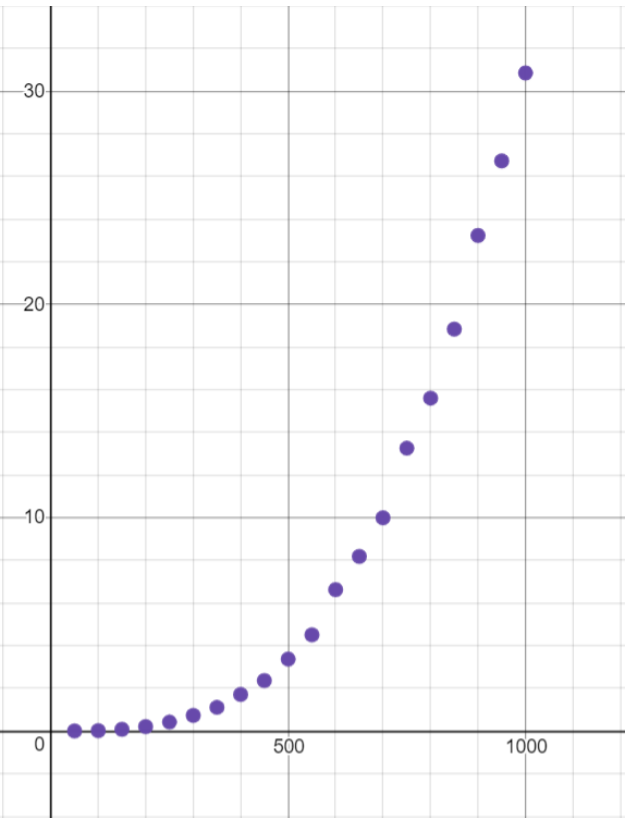
Inverse using Gauss Elimination
Method



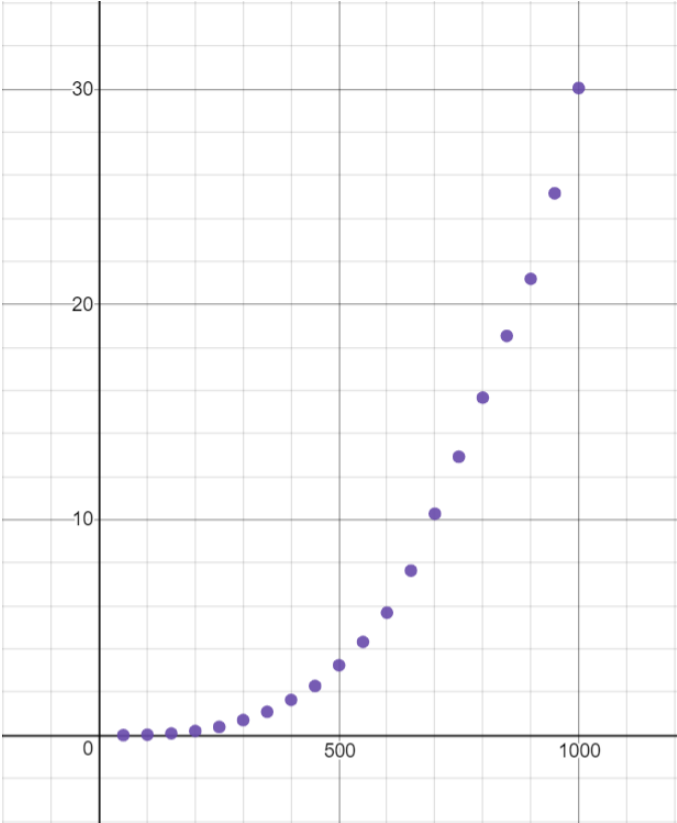
Cholesky LU Decomposition



Dolittle LU Decomposition



Crout LU Decomposition

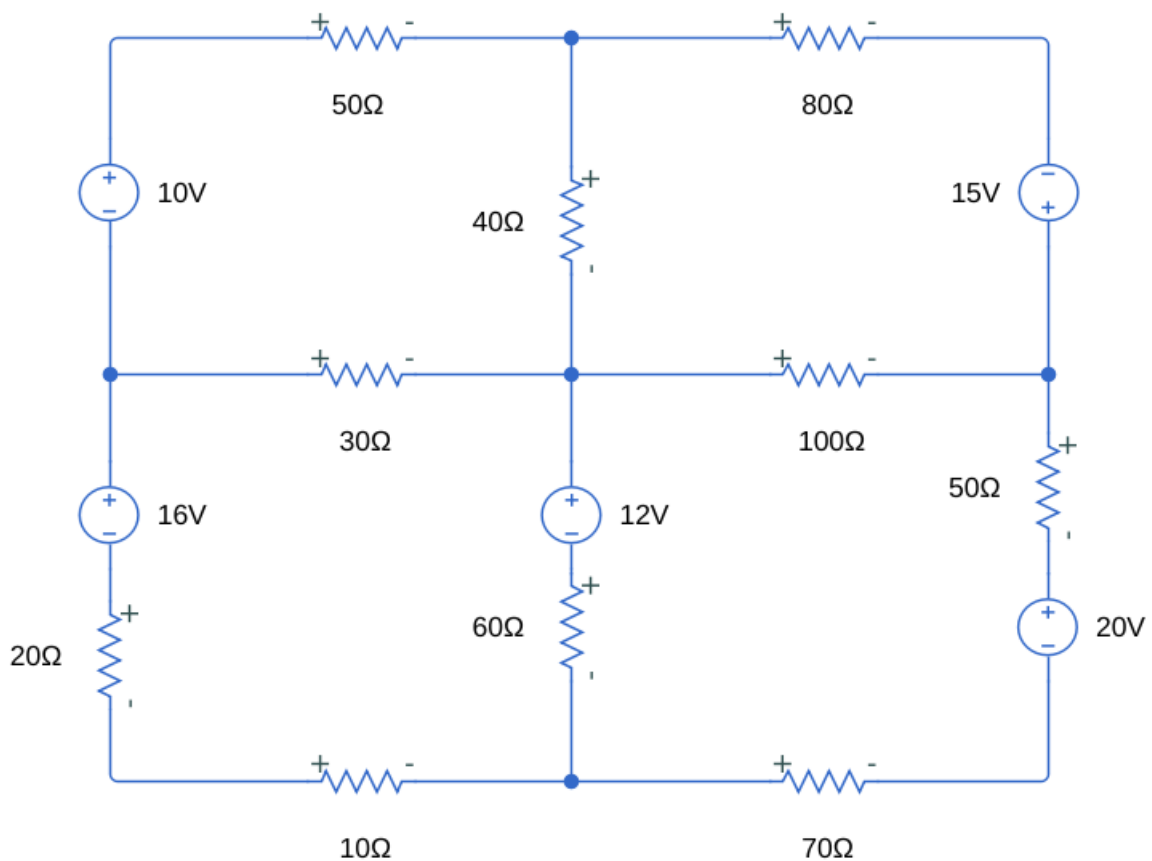


Electrical Engineering Applications

Application 1:

Finding currents in all branches of a linear, passive network with given number of nodes and values of voltages and resistances in every branch.

A model network has been shown below.



Model electrical network used for comparison.

Input given to the C++ solver:

9 (nodes)

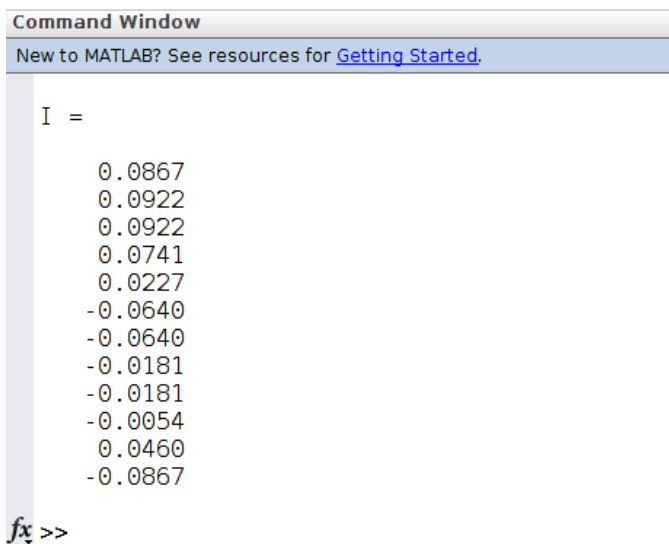
12 (links/branches)

Format for branches:

<node1><node2> <V2 -V1 sources><Resistance>

1	2	0	50
2	3	0	80
3	4	15	0
4	5	0	100
5	6	0	30
6	7	-16	20
7	8	0	10
8	9	0	70
9	4	20	50
2	5	0	40
5	8	-12	60
1	6	-10	0

We ran *NetworkSolver.cpp* on this network with the mentioned inputs, and then we made a simulink model and simulated it. The results have been shown below.



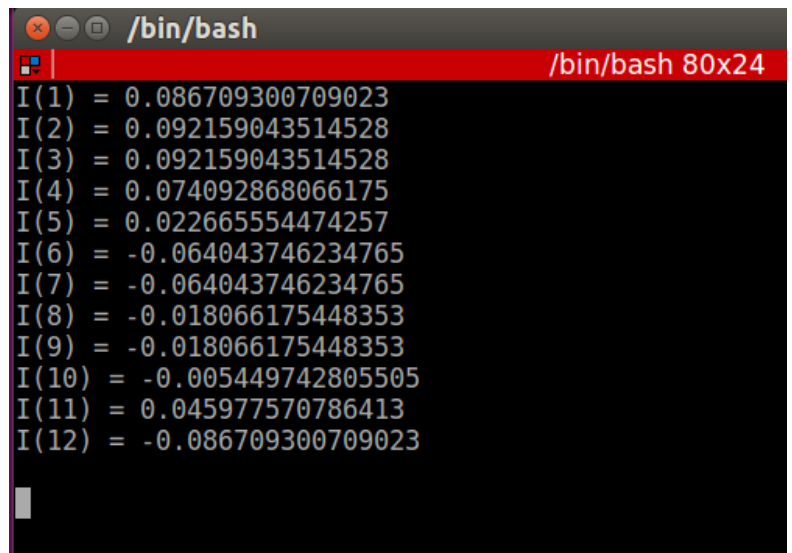
```
Command Window
New to MATLAB? See resources for Getting Started.

I =

    0.0867
    0.0922
    0.0922
    0.0741
    0.0227
   -0.0640
   -0.0640
   -0.0181
   -0.0181
   -0.0054
    0.0460
   -0.0867

fx >>
```

Matlab / simulink output



```
/bin/bash
/bin/bash 80x24

I(1) = 0.086709300709023
I(2) = 0.092159043514528
I(3) = 0.092159043514528
I(4) = 0.074092868066175
I(5) = 0.022665554474257
I(6) = -0.064043746234765
I(7) = -0.064043746234765
I(8) = -0.018066175448353
I(9) = -0.018066175448353
I(10) = -0.005449742805505
I(11) = 0.045977570786413
I(12) = -0.086709300709023
```

NetworkSolver.cpp ouptut

Application2:

Finding out whether a given linear control system is controllable and observable from its State-Space Representation

This uses the definition of Contralabilty and Observability in advanced Control Systems, and tells whether the given system is Controllable/ Observable.

Input to [CO_check.cpp](#):

States = 4

Inputs = 1

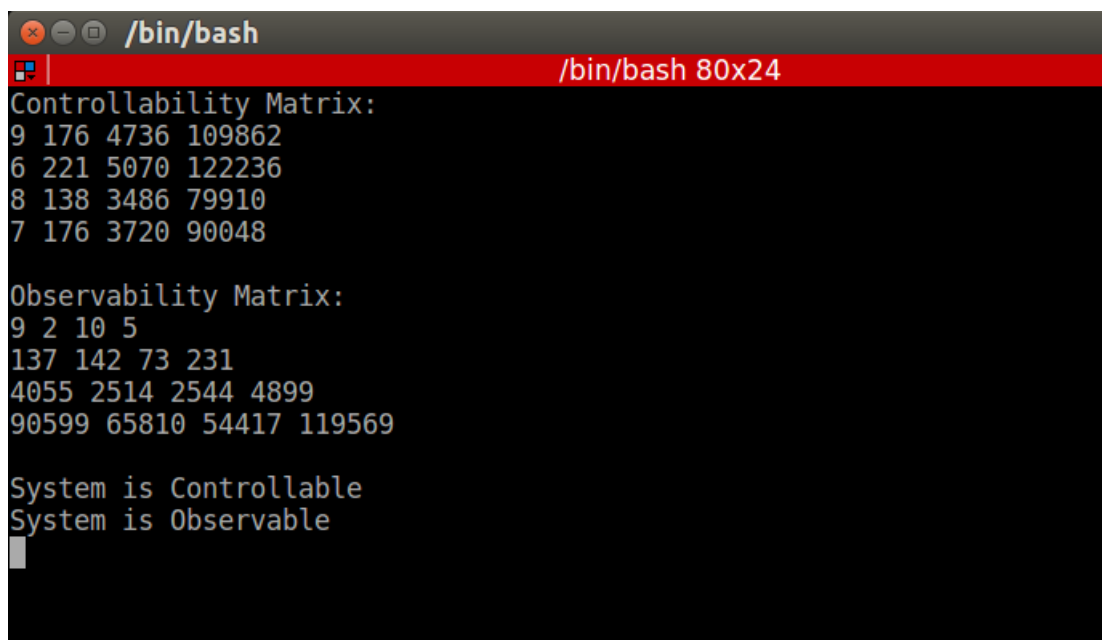
Outputs = 1

A = 4 10 3 8
 8 6 8 7
 4 4 1 10
 9 0 4 9

B = 9
 6
 8
 7

C = 9 2 10 5

D = 1



```
/bin/bash
/bin/bash 80x24
Controllability Matrix:
9 176 4736 109862
6 221 5070 122236
8 138 3486 79910
7 176 3720 90048

Observability Matrix:
9 2 10 5
137 142 73 231
4055 2514 2544 4899
90599 65810 54417 119569

System is Controllable
System is Observable
```

Output produced by [CO_check.cpp](#)

Results and Conclusion

Matrices are of prime importances to scientific and mathematical computing. We have successfully made an attempt to give an open-source code for all these computations for Engineering fields. All the above said matrix operations have been implemented and have been tested extensively by us. Corresponding libraries have also been made and written in accordance to industry standards. Two simple applications relating **Electrical Engineering** have been demonstrated. Many applications can be made using the QR Method and fractional power but, those are beyond the scope of this report.

A case study along with the complete C++ code can be found at out IOP's **github Repository** at the link:

<https://github.com/AayushChaturvedi/IOP2019>