BSc Computer ScienceFinal Report

Decomposing Unitary Matrices

Candidate Number :221337

Project Supervisor: Prof. Niel De

BeaudrapSchool of Engineering and

Informatics University of Sussex

August 2021

Statement of Originality

This report is submitted as part of requirement for the degree of computer science at the university of Sussex. It's the product of my own research and labor where indicated in the text.

Acknowledgments

I would like to thank Dr. Niel De Beaudrap for supervising my work and providing guidance where needed.

Abstract

Through algorithm research and attempting to reimplement the algorithm, this paper will explore the python process of decomposing a unitary matrix. My project aims to successfully implement a python algorithm that can successfully decompose a unitary matrix.

Contents

Introduction

Motivation and problem area

Unitary matrices are used to build quantum computers. A quantum computer harnesses some of the almost mystical phenomena of quantum mechanics to deliver huger leaps forward in processing power (Giles, 2021). Quantum computers contain a commodity called a quantum gate. In a quantum gate the input qubit numbers and output numbers must be equal (Grumbling and Horowitz, n.d.). The action is being performed by the quantum gate is defined by multiplication of the unitary matrix.

Through the study of program analysis and fundamental of computer science mathematics, it was realized the importance of this project when it comes to re-inventing and rebuilding and understanding the function of some advanced computer science fields. This project offers a steppingstone into what would be a more complicated and user oriented fundamental of computer science.

Unitary decomposition is beneficial in a variety of situations. The first type of algorithm is the board class, which generates arbitrary unitary gates that must be translated into quantum circuits. However, it can also be used to facilitate more modular quantum algorithm creation or as an aggressive optimization strategy.

As an example of a possible application for unitary decomposition, we employ quantum techniques that we developed in the context of genome sequence. A genome sequence is first read as a series of small bits, which must then be joined to form a whole DNA sequence. Currently, a variety of algorithms are used to do this., which are executed using (classical) high performance computing systems.

For genome sequencing using quantum accelerators, the algorithms that will be discussed both use unitary matrix in the process of finding the position of a short rad (sequence of a small piece of DNA) on a reference genome. That matrix needs to be decomposed before the algorithm can be run on a quantum accelerator or simulator.

The first quantum genome sequencing algorithm we'll use is quantum indexed, bidirectional associative memory, which uses a unitary oracle assembled from a binomial distribution, where d is the number of qubits required to store the memory states and hamming distance is the hamming distance between the query pattern and all memory states. D is also the dimension of the generated vector matrix.

Quantum states can also be written as complex vectors, in addition to the notation. This is especially helpful when combining the states of several qubits, as the first row of the vector corresponds to the binary number “0.” There are as many qubits as there are bits. The second row corresponds to the number "1," and so on. For a three-qubit state, for example, the first row corresponds to |000 and the second to. |001. This continues to the final row, which is the state vector has 2^n rows for the state of n qubits.

Qubits are manipulated using gates, which are matrices that operate on the qubit state vector. To calculate the effect of gates on the combined qubit state, the state vector is multiplied by the matrix representation of the gates in reverse order (Krol et al., 2021).

Project aim and object

The overall aim of this project is to decompose a unitary matrix of shape 2^k by 2^k, once the matrix has been decomposed a software/program will need to be realized so that the matrix can be decomposed on code. The problem area of the project is that the given matrices requires decomposing so that the I can realize a program/software that I can write the code in, the challenging aspect will be for me to first learn how to decompose a unitary matrix as well learn just exactly how a unitary matrix works, then I must decompose the unitary matrix and write up my algorithm (method) in code. Various research about coding languages and how to experiment and implement algorithms that solve matrices will need to be researched as well. On top a further research on the different methods that are used to decompose a unitary matrix. Searching through all these sources should allow me to successfully design/implement and algorithm that has the capability of decomposing a unitary matrix.

Project relevance

For this project I will need to know how to apply the skills I learned in programming languages and hardcode them so that I can get the best desired output, the two languages that I will be experimenting with the most so that I can develop the software for this project are java and python, along with java and python I will also take it upon myself to practice and experiment with MATLAB. I first began to learn my skills in java during my international year one module “introduction to programming”, and I began to learn my skills in python during my 2nd year module “natural language engineering”. during my first year I as well learned the technique and format for writing algorithms in my module “Algorithmic thinking. With matrices I began learning the skill during my 2nd year module program analysis, this however is my first time experimenting and learning unitary matrices

"1. Public Interest

You shall:

a. have due regard for public health, privacy, security and wellbeing of others and the environment.

b. have due regard for the legitimate rights of Third Parties.*

c. conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, color, race, ethnic origin, religion, age or disability, or of any other condition or requirement

d. promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.

"

"2. Professional Competence and Integrity

You shall:

a. only undertake to do work or provide a service that is within your professional competence.

b. NOT claim any level of competence that you do not possess.

c. develop your professional knowledge, skills and competence on a continuing basis,

maintaining awareness of technological developments, procedures, and standards that are relevant to your field.

d. ensure that you have the knowledge and understanding of Legislation and that you comply with such Legislation, in carrying out your professional responsibilities.*

e. respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.

f. avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.

g. reject and will not make any offer of bribery or unethical inducement.

"

With regards to points 2 (a)(b)(c) is have discussed the project with my technical supervisor and we believe that the project is within my professional competence.

In accordance with point 2(d), I shall not break any legislations during the projects lifetime.

With regards to point 2(e), I shall regularly share the progress of the project with my technical supervisor.

With regards to point 2(f), I shall ensure the project is completed with a functioning tool that meets its aims and objectives.

"3. Duty to Relevant Authority

You shall: 6

a. carries out your professional responsibilities with due care and diligence in accordance with the Relevant Authority's requirements whilst exercising your professional judgement at all times.

b. seeks to avoid any situation that may give rise to a conflict of interest between you and your Relevant Authority.

c. accepts professional responsibility for your work and for the work of colleagues who are defined in each context as working under your supervision.

d. NOT disclose or authorize to be disclosed or use for personal gain or to benefit a third party, confidential information except with the permission of your Relevant Authority, or as required by Legislation.

e. NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information) or take advantage of the lack of relevant knowledge or inexperience of others."

With regards to point 3(a), I shall ensure that during the projects lifetime, I will meet the requirements of the university. I shall do this by producing the deliverables for each of the project's deadlines.
With regards to point 3(b) I shall avoid any conflicts of interest that may arise through the development of the project between myself and the university.
With regards to point 3(c), I shall accept full responsibility for the project and the project's success.
With regards to points 3(d) I will not disclose confidential information about the project by refusing to comment when asked.
With regards to point 3(e), I shall not misrepresent of withhold information about the project by remaining honest about the progress of the project.

Ethical considerations

Due to most of this project being conducted on a theoretical aspect and it being a very research heavy project, there is no need to take any major ethical considerations into account. The only ethical consideration that should be measured is that the correct citation is issued in the correct place, and research papers used are correctly citied in the bibliography.

Related Work

Decomposing Unitary Matrices Background research

How is your approach different and better than others?

With my approach to the research topic. It is decided that the LU decomposition method will be used as the basis to decompose the unitary matrices. The program will be coded with programming language python, while that does not seem extraordinary, the choice to use python is to experiment and see if such a program can work on python, as oppose to MATLAB which is the language that is typically used to complete project like this.

What is your contribution to the field?

Due to the large number of studies conducted on this subject, there have been a variety of publications for this project, and there are a variety of sources available for a variety of research fields. My contribution to the field with this project would be to study and observe it in order to learn new insights and expand my mathematical knowledge. My contribution to this field is slightly different approach to complete the project. With previous research papers it is shown that the language that was used for the project was python.

How to decompose a unitary matrix

Suppose you have a simple 2*2 matrix with complex values. The matrix is represented as

$$U = \begin{bmatrix} \frac{i}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{bmatrix}$$

It is then important to find the conjugate of the matrix which in this case can be represented as.

$$Conjugate = \begin{bmatrix} -\frac{i}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{bmatrix}$$

For this example it required that we find the Hermitian matrix, these are square matrices whose conjugate transpose is the same as the original matrix, which can be represented with the value of U^H

$$u^H = \begin{bmatrix} \overline{-\frac{i}{\sqrt{2}}} & \overline{\frac{1}{\sqrt{2}}} \\ -\frac{1}{\sqrt{2}} & \frac{i}{\sqrt{2}} \end{bmatrix}$$

Once the Hermitian matrix is found we can then multiple the matrices of U^H*U. This matrix decomposition .

$$U^H U = \begin{bmatrix} \frac{i}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{i}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} -i/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & i/\sqrt{2} \end{bmatrix}$$

$$U^H U = \begin{bmatrix} \frac{-1}{2} + 1/2 & \frac{i}{2} - i/2 \\ -\frac{i}{2} + i/2 & \frac{1}{2} - (-1)/2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

This shows that the matrix used here is unitary because the value of U^H*U = I which is the identity matrix (Lakshminarayan, 2019). This gives a quick summary and a general understanding as to how unitary matrices work, however for the purpose of this project, a slightly different decomposition method will be used, LU decomposition.

Decomposing a unitary matrix

The basic idea is to multiply U on the left with 2 x 2 unitarires until the identity is obtained. This method provides a sequence of gates Uk such that U1…UnU=I which then gives you the decomposition of U in terms of 2 x 2 unitaries: U=U1…Un

For example, suppose you start with

$$U = \begin{bmatrix} 1/2 & 1/2\sqrt{3} & \sqrt{2/3} \\ -1/2 & \sqrt{3}/2 & 0 \\ 1/\sqrt{2} & 1/\sqrt{6} & -1/\sqrt{3} \end{bmatrix}$$

We start by finding a matrix which only mixes first and third row of U, with the goal of annihilating the (3,1) element. In general, a matrix U1 mixing only first and third column of U has the form.

$$\begin{bmatrix} * & 0 & * \\ 0 & 1 & 0 \\ * & 0 & * \end{bmatrix}$$

To sed U31 to zero essentially amounts to finding parameters a,b such that au11+bU31=0. Which gives a = - bU31/U11 (the case U11 = 0 can be handled separately). You then put the resulting a,b in the third row of

U1 and then adjust the rest of the paraments in U1 in order to make into a unitary. This resulting following U1:

$$U1 = \begin{bmatrix} \overline{1/\sqrt{3}} & 0 & \overline{\sqrt{2/3}} \\ 0 & 1 & 0 \\ -\sqrt{2/3} & 0 & 1/\sqrt{3} \end{bmatrix}$$

Then

$$U1U = \begin{bmatrix} \sqrt{3}/2 & 1/2 & 0 \\ -1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

In this case we were luck in that we obtained two zeros in one step, but this only due to this particular choice of U. You now need to find some U2 which only mixes the first two rows of U, that is a matrix with the structure.

$$\begin{matrix} * & * & 0 \\ * & * & 0 \\ 0 & 0 & 1 \end{matrix}$$

To send to zero the (2,1) element of U1U we can use (again focus on coefficients that sent to xero the elements in the first column of U1U and then put them in the second row of U2, then filling out the rest to make U2 unitary.):

$$U2 = \begin{bmatrix} -\sqrt{3/2} & 1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So that finally

$$U2U1U = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Notice how some of the sings in the resulting matrix are wrong. This is because I didn't put much care in the signs of the matrices Ui. Following the more accurate prescription given in the paper (note how they also include the determinants in the expressions and take care of the cases with compiles numbers). You will always obtain the identity at the end.

Finally, it might be interest to note that this essential LU decomposition applied to the special case of aunitary matrix.

The reason as to why the unitary matrix decomposition is in effect in two different methods here is to collect a general understanding of how unitary matrices work. As well as to outline how the program will be coded on the python IDE.

Algorithm Research

	Aim of Study	How did it achieve this aim	Conclusion	Problem area of the research studies

Unitary decomposition implemented in the OpenQL programming language for quantum computation **By: A.M. Krol**	With this project the researcher is attempting to decompose and program a unitary matrix in the openQL language. The reason for attempting this research was to provide support for the quantum programming language, openQL.	The process of decomposition was don theoretically and by hand, meaning that at this point no software was used. The researcher then began to put to put the decomposed matrices into a feasible programming language and begin with the experimentation. The researcher method of circuit and demultiplexing decomposition is all a concept that is new to me. However, the researcher approach to this exact topic is unique.	With the conclusion of this study the unitary decomposition that was implemented in the OpenQL give the correct results in a reasonable amount of time. With this decomposition algorithms can be done completely inside OpenQL, and do not require any more manual copying and positing of code. This makes the whole process faster and provides fewer opportunities for human error than before.	The problem with this project is that the user is investigating to complete the decomposition of the unitary matrix as to further increase the capabilities of a programming language. While the project is very useful it follows a very physics and quantum computing background, this is too advanced for the current need of the project that I am attempting to complete.
Decomposition of unitary matrices for finding quantum circuits: Application to molecular Hamiltonians **By: Anmer Daskin and Sabre Kais**	The aim of this project is to use and optimization algorithm to decompose a given unitary matrix into, proper-minimum cist quantum gate sequences. They are using this to find the approach for chemical bonding of gates	They needed to achieve the aim with minimum costs and errors. Therefore, the two factors which need to be optimized with the optimisation are the error and the cost circuit, the cartesian genetic programming is the program that was used to represent the quantum gates. Each of the inputs that were placed are represented as integers and genotypes. This is because quantum gates influence the whole system.	The overall project seemed to be partially successful. They managed to use the quantum algorithms to solve the various problems produced on the quantum gates for different molecules, but they are still continuing research to develop farther methods to for molecules that have still not reached a solution for.	The problem with this research project even though it seems very interesting and intergerning is that it is also far too complex for my academic knowledge on the topic currently. As well the project is very chemistry based, whereas chemistry is field of study that I have not delved into for a long time.

Decomposition of unitary matrix into quantum gates **By: Dmytro Fedoriaka**	In this paper the researcher is attempting to solve and implementation of unitary matrix with as sequence of quantum that can be expressed using standard library of Q# language. The language is a domain-specific programming language that is for expressing quantum algorithms developed tech company Microsoft.	The researcher investigated an algorithm that can help them solve two-level decomposition. Once they had completed their investigation of suitable algorithm that can allow them to solve the decomposition, they began the software implementation phase, they implemented a python program which uses the described algorithm to transform an arbitrary uniform matrix into a Q# operation. The program performs all the decomposition steps that the researcher intended.	They reached the conclusion that a two-level matrix can be decomposed into sequence of simpler gates. Their process is shown in each of the previous steps that taken throughout the research paper.	The problem with this research project even though it seems very interesting and intergerning is that it is also far too complex for my academic knowledge on the topic currently. As well the project is very chemistry based, whereas chemistry is field of study that I have not delved into for a long time.

Compare and contrast the algorithms

When the function of matrices is clear, it then becomes easy to identify and define an eigen vector. The input vector that goes in and the output vector that comes out are both parallel to each other, and that vector is called an eigenvector. That is, they have either sustain or are unaffected by the dissertation, this is called eigen decomposition (Sayyed, 2021). eigen decomposition divides a matrix into eigenvectors and eigenvalues. Eigen vectors and values are non-zero vectors that alter by a scalar factorwhen a linear transformation is applied to them.

In linear algebra, a matrix decomposition or matrix factorization is a factorization of a matrix into a product of matrices. There are many different matrix decomposition. One of them is Cholesky decomposition. The Cholesky decomposition or Cholesky factorization is the decomposition of matrices. It is done by breaking the matrix into a lower triangular matrix and its conjugate transpose.

While the research that was done was good enough to complete the programming, the general research of this algorithm mostly comes down to two different decomposition methods, LU decomposition and Cholesky decomposition. In both LU decomposition and Cholesky decomposition a square nonsingular

matrix A is factored into a product of triangular is factored into a product of triangular matrices. In a Cholesky factorization the lower and upper triangular factors are transposes of each other (Yang, 2021).

Once a matrix is factored as either an LU or Cholesky factorization then the linear system of equations can be solved. In either backward or forward substitution. The key inside is that if written out the equation will be easily computer (Higham and Pranesh, 2021).

When computing LU factorization, one may possibly encounter a zero pivot which necessitation permuting the rows of the matrix to obtain a nonzero pivot point. When pivoting is used one actually computes a decomposition of the form PA = LU. When performing and LU factorization on a computer using inexact arithmetic. It would be important to permute when a pivot entry is simply small, not necessarily exactly zero. In fact it is common to permute the matrix such that we always pick the largest picot in the column. When perform Cholesky decomposition the pivot will never be encountered, because Cholesky decomposition does not need a pivot to ensure the accuracy of the computation ((Reid, 2014)).

For the purpose of this project, research supports that when it comes to programming LU decomposition may be more effective for this specific task. That is because LU decomposition works for unitary matrices, however Cholesky factorization only works for a positive-definite matrix. To attempt to decompose a unitary matrix would prove to be impractical.

Method and approach

The algorithm is coded using python. The MATLAB algorithm is details graphs and movements of the matrix. While this does provide a very useful visual aid. The algorithm itself is very complicated and needs to be decomposed in a through fashion. The algorithm is very well laid out.

The algorithm presented here is very simple, it is not simple because of the method that it is written in, it is simple because the work that is put in the algorithm does not work with unitary matrices, various changes need to be made so that the matrix can be able to handle unitary matrices.

This algorithm is good. It shows what the python program is doing under the function of numpy. This is very useful for the work that I am doing because it helps me with factorizing and fixing the matrix. This is useful, the algorithm was implemented in python, this makes it a good algorithm and a useful one for me.

The algorithms displayed each contain their own set of positives and negatives. The algorithms that were researched were all somehow used to complete the final algorithm that is submitted. The algorithms helped me build a thorough understanding of the current situation and how it must be assessed. Through implementations on python, java, MATLAB. The papers that were assed each reflect exactly how to implement the algorithm on the specific code language that is being used.

```
import pprint
import scipy
import scipy.linalg   # SciPy Linear Algebra Library

A = scipy.array([ [7, 3, -1, 2], [3, 8, 1, -4], [-1, 1, 4, -1], [2, -4, -1, 6] ])
P, L, U = scipy.linalg.lu(A)

print("A:")
print(A)

print("P:")
pprint.pprint(P)

print("L:")
pprint.pprint(L)

print ("U:")
pprint.pprint(U)
```

```
A:
[[ 7  3 -1  2]
 [ 3  8  1 -4]
 [-1  1  4 -1]
 [ 2 -4 -1  6]]
P:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
L:
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.42857143,  1.        ,  0.        ,  0.        ],
       [-0.14285714,  0.21276596,  1.        ,  0.        ],
       [ 0.28571429, -0.72340426,  0.08982036,  1.        ]])
U:
array([[ 7.        ,  3.        , -1.        ,  2.        ],
       [ 0.        ,  6.71428571,  1.42857143, -4.85714286],
       [ 0.        ,  0.        ,  3.55319149,  0.31914894],
       [ 0.        ,  0.        ,  0.        ,  1.88622754]])
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWarning: scipy.array is deprecated and will be removed in SciPy 2.0.0,
  """
```

Project requirements

This project has many requirements that need to be fulfilled. To begin the program on this project was completed using python. Various attempts have also been done on MATLAB and java. This along with functional features as well as errors that where encountered on the way, they allow for the coded language to fully function in the intended way. The following is a outline of the requirements that were set to be met for this project, ranging from desirable outcomes, to fully functioning requirements.

Desirable

The goal of this project is to deliver a software program that will be able decompose a unitary matrix. Whilst the method that I am intending on using is not fully set in stone the approach and research has already begun. Currently as it stands, I am intending to use past research papers to learn and understand how exactly to decompose a unitary matrix. Upon that I will begin the coding phase. The intended software should be able to run on all operating systems, as well the intended software should compile without trouble. Thus, is that the compiled program can run quickly and take up little memory space. As well as compiled program is always ready to run (Kersten, Leis and Neumann, 2020).

1. The program runs on windows operating system
2. The program compiles at fast enough time, preferably a few seconds to no more than 7 minutes depending on the size of the matrix currently being compiled.
3. If the program experiences a crash or a bug it will self-terminate and stop running
4. The program should be able to decompose unitary matrices of any specification i.e. it should also be able to compile and run rectangular matrices.

1. The program can be written and/or duplicated in all the programming languages mentioned i.e. java, python, MATLAB
2. The program can run on all operating systems mainly windows, iOS and Linux.

Functional requirements

- **High time complexity:** Time complexity is important when it comes to functioning and running the algorithm. The time complexity is important because the less time the better the algorithm functions, the importance of this algorithm is to make sure that it functions at the highest and most important efficiency.
- **Algorithm compiling:** It is important that the algorithm can compile the code that is put into it. The algorithm complication needs to be smooth and shouldn't not be affected by anything. This is because if the algorithm has a steady decomposition then it should work with any trouble. This shows the versatile of the algorithm.

Nonfunctional requirements

Coding language

Description	All the coding of the project will be done in python 3 and using the google colab IDE. This is done because of the SciPy in built tool. It contains many out of the box tools to perform pre-processing techniques, model fitting and evaluation. Furthermore, python is a language that was personally used before. The built-in features in python that allow mathematical calculation are very useful in such cases (McKinney, 2012).
Notes	The use of python was due to the inbuilt mathematical functions in python that can help with completing a research of this caliber.

Clear workflow

Description	Throughout the code, there are comments. This is to make it easier to grasp the code line by line. The commented code elevates the project's professionalism by allowing the user to fully comprehend all the code's processes. Collab notebooks make it simple to show the program's final output. This allows users to see what the code is doing while still assisting them in arriving at a result. Colab notebooks also allow you to mark down cells in which you can write tests. This text will include a detailed description of what the code does.
Notes	

Commented code

Description	Throughout the code, there should be explicit comments. The user would be able to interpret the code more clearly line by line as a result of this. Well-commented code enhances the project's professionalism by allowing the user to fully comprehend all of the code's procedures.
Notes	

Efficient code

Description	The code should be as easy as possible to execute. It is important to write the code in such a way that no lines repeat themselves. The names of variables should make sense and be easy to understand. This is influenced by the fact that clean and usable code is easier to read and understand.
Notes	

Scalability

Description	The codebase needs to be scalable. Scalability refers to the code's ability to accommodate potential experimental results. Extra data can be easily pre-processed, a model fitted, and tested without requiring major code changes.

	This is a fascinating subject. The code must be flexible since it must be evaluated with a variety of matrices of various sizes and values.
Notes	

Compatibility

The intended software should run on all operating systems and compile with ease. Therefore, the compiled program is easy to execute and uses little memory. The importance of code compatibility stems from the fact that when the code is compatible, the algorithm runs smoothly on alloperating systems.

Results and discussion

Results

```
[7, 3, -1, 1.4142135623730951j]
[3, 8, 1, 1.7320508075688772j]
[-1, 1, 4, -1]
[2, -4, -1, 6]
A:
[[7, 3, -1, 1.4142135623730951j],
 [3, 8, 1, 1.7320508075688772j],
 [-1, 1, 4, -1],
 [2, -4, -1, 6]]
P:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
L:
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.42857143,  1.        ,  0.        ,  0.        ],
       [-0.14285714,  0.21276596,  1.        ,  0.        ],
       [ 0.28571429, -0.72340426,  0.08982036,  1.        ]])
U:
array([[ 7.        ,  3.        , -1.        ,  2.        ],
       [ 0.        ,  6.71428571,  1.42857143, -4.85714286],
       [ 0.        ,  0.        ,  3.55319149,  0.31914894],
       [ 0.        ,  0.        ,  0.        ,  1.88622754]])
```

Fig. 1 (Matrix experiment 1)

```
A:
[[ 7.+0.j          3.+0.j         -1.+0.j          0.+1.41421356j]
 [ 3.+0.j          8.+0.j          1.+0.j          0.+1.73205081j]
 [-1.+0.j          1.+0.j          4.+0.j         -1.+0.j         ]
 [ 2.+0.j         -4.+0.j         -1.+0.j          6.+0.j         ]]
P:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
L:
array([[ 1.        ,  0.        ,  0.        ,  0.        ],
       [ 0.42857143,  1.        ,  0.        ,  0.        ],
       [-0.14285714,  0.21276596,  1.        ,  0.        ],
       [ 0.28571429, -0.72340426,  0.08982036,  1.        ]])
U:
array([[ 7.        ,  3.        , -1.        ,  2.        ],
       [ 0.        ,  6.71428571,  1.42857143, -4.85714286],
       [ 0.        ,  0.        ,  3.55319149,  0.31914894],
       [ 0.        ,  0.        ,  0.        ,  1.88622754]])
```

Fig 2 (Matrix experiment 1)

Result	Algorithm successfully compiled and decomposed the matrix
Descripiton	The matrix was successfully decomposed by the algorithm. The matrix had two sets of negative numbers, square root -2 which square root -3, and was 2 by 2. The algorithm was also able to identify imaginary numbers. As a result, you can see in some of the figures that the algorithm has +0j for some of the figures, when this represented it means that the number adjacent to the symbol is not an imaginary number.
Notes	Matrix works well and compiles well, this is shown with the initial testing. Matrix showed issues with complication. The program crashed 2/5 times when running this algorithm. A lot of the crashes where due to changes in syntax and requiring to import any and all mathematical functions.

```
A:
[[ 7.+0.j          3.+0.j         -1.+0.j          0.+1.41421356j]
 [ 3.+0.j         -8.+0.j          1.+0.j          0.+1.73205081j]
 [-1.+0.j          1.+0.j          4.+0.j          0.+2.23606798j]
 [ 2.+0.j         -4.+0.j         -1.+0.j          6.+0.j        ]]
P:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
L:
array([[ 1.         +0.j,  0.         +0.j,  0.         +0.j,
         0.         +0.j],
       [ 0.42857143+0.j,  1.         +0.j,  0.         +0.j,
         0.         +0.j],
       [-0.14285714+0.j, -0.15384615-0.j,  1.         +0.j,
         0.         +0.j],
       [ 0.28571429+0.j,  0.52307692-0.j, -0.35849057+0.j,
         1.         +0.j]])
U:
array([[ 7.         +0.j        ,  3.         +0.j        ,
        -1.         +0.j        ,  0.         +1.41421356j],
       [ 0.         +0.j        , -9.28571429+0.j        ,
         1.42857143+0.j        ,  0.         +1.12595928j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         4.07692308+0.j        ,  0.         +2.61132299j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        ,  6.         -0.05688968j]])
```

Fig 3 (Matrix trial 2)

```
[7, 3, -1, 1.4142135623730951j]
[3, -8, 1, 1.7320508075688772j]
[-1, 1, 4, 2.23606797749979j]
[2, -4, -1, 6]
A:
[[7, 3, -1, 1.4142135623730951j],
 [3, -8, 1, 1.7320508075688772j],
 [-1, 1, 4, 2.23606797749979j],
 [2, -4, -1, 6]]
P:
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
L:
array([[ 1.         +0.j,  0.         +0.j,  0.         +0.j,
         0.         +0.j],
       [ 0.42857143+0.j,  1.         +0.j,  0.         +0.j,
         0.         +0.j],
       [-0.14285714+0.j, -0.15384615-0.j,  1.         +0.j,
         0.         +0.j],
       [ 0.28571429+0.j,  0.52307692-0.j, -0.35849057+0.j,
         1.         +0.j]])
U:
array([[ 7.         +0.j        ,  3.         +0.j        ,
        -1.         +0.j        ,  0.         +1.41421356j],
       [ 0.         +0.j        , -9.28571429+0.j        ,
         1.42857143+0.j        ,  0.         +1.12595928j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         4.07692308+0.j        ,  0.         +2.61132299j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        ,  6.         -0.05688968j]])
```

Fig 4 (Matrix trial 2)

Result	Algorithm successfully compiled and decomposed the matrix
Description	This time, the algorithm was able to successfully decompose a matrix containing three complex numbers. Square root -2, square root -3, and square root -5 are complex numbers. All of these were sorted into separate matrices. They were able to successfully decompose the matrix using the algorithm's LU decomposition. The effectiveness of this method shows that the algorithm can decompose the matrix.
Notes	The reason for testing the algorithm with these specific values is because the algorithm is meant to be cable to compile and successfully decompose a matrix with these specific values, the decomposition of this matrix is because it is meant to represent the minimum benchmark requirement that is meant to be meant in this algorithm. The algorithm this time managed to run smoothly. There were some changes made to the algorithm at first before the function was changed so that I can attempt repeat success.

```
A:
[[ 7.+0.j          3.+0.j         -1.+0.j          0.+1.41421356j]
 [ 0.+2.64575131j  0.+3.31662479j  0.+1.41421356j  0.+1.73205081j]
 [-1.+0.j          1.+0.j          4.+0.j          0.+2.23606798j]
 [ 2.+0.j         -4.+0.j         -1.+0.j          6.+0.j        ]]
P:
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.]])
L:
array([[ 1.        +0.j        ,  0.        +0.j        ,
         0.        +0.j        ,  0.        +0.j        ],
       [ 0.28571429+0.j        ,  1.        +0.j        ,
         0.        +0.j        ,  0.        +0.j        ],
       [-0.14285714+0.j        , -0.29411765-0.j        ,
         1.        +0.j        ,  0.        +0.j        ],
       [ 0.        +0.37796447j, -0.        -0.44938587j,
         0.        +0.40339029j,  1.        +0.j        ]])
U:
array([[ 7.        +0.j        ,  3.        +0.j        ,
        -1.        +0.j        ,  0.        +1.41421356j],
       [ 0.        +0.j        , -4.85714286+0.j        ,
        -0.71428571+0.j        ,  6.        -0.40406102j],
       [ 0.        +0.j        ,  0.        +0.j        ,
         3.64705882+0.j        ,  1.76470588+2.31925701j],
       [ 0.        +0.j        ,  0.        +0.j        ,
         0.        +0.j        ,  1.65166756+3.71650081j]])
```

Fig 5 (Matrix trial 3)

```
[7, 3, -1, 1.4142135623730951j]
[2.6457513110645907j, 3.3166247903554j, 1.4142135623730951j, 1.7320508075688772j]
[-1, 1, 4, 2.23606797749979j]
[2, -4, -1, 6]
A:
[[7, 3, -1, 1.4142135623730951j],
 [2.6457513110645907j,
  3.3166247903554j,
  1.4142135623730951j,
  1.7320508075688772j],
 [-1, 1, 4, 2.23606797749979j],
 [2, -4, -1, 6]]
P:
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.]])
L:
array([[ 1.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        ,  0.         +0.j        ],
       [ 0.28571429+0.j        ,  1.         +0.j        ,
         0.         +0.j        ,  0.         +0.j        ],
       [-0.14285714+0.j        , -0.29411765-0.j        ,
         1.         +0.j        ,  0.         +0.j        ],
       [ 0.         +0.37796447j, -0.         -0.44938587j,
         0.         +0.40339029j,  1.         +0.j        ]])
U:
array([[ 7.         +0.j        ,  3.         +0.j        ,
        -1.         +0.j        ,  0.         +1.41421356j],
       [ 0.         +0.j        , -4.85714286+0.j        ,
        -0.71428571+0.j        ,  6.         -0.40406102j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         3.64705882+0.j        ,  1.76470588+2.31925701j],
       [ 0.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        ,  1.65166756+3.71650081j]])
```

Fig 6 (Matrix trial 3)

Result	Algorithm successfully compiled and decomposed the matrix.
Description	For this experimentation with the matrix algorithm I attempted to change one of the matrices for it to almost exclusively be compromised of complex numbers. One of the matrices is completely covered in negative numbers. I did this because I wanted to see exactly how far the algorithm can be pushed before it crashes. So far, the algorithm seems to be working well. The algorithm has managed to decompose matrices, when the matrices are all of the same size, and the contents of the matrix is not needed.
Notes	The reason as to why it was decided to test this algorithm with matrix of all complex numbers was to see if the matrix decomposition algorithm

	that is implemented here would work. If this works then it is good because it helps with matrix decomposition and recalibration.

```
A:
[[ 7.+0.j          3.+0.j          2.+0.j         -1.+0.j
   0.+1.41421356j -3.+0.j        ]
 [ 0.+2.64575131j  0.+3.31662479j  0.+1.41421356j  0.+1.73205081j
   1.+0.j         -1.+0.j        ]
 [-1.+0.j          1.+0.j          4.+0.j          0.+2.23606798j
   2.+0.j         -2.+0.j        ]
 [ 2.+0.j         -4.+0.j         -1.+0.j          6.+0.j
   5.+0.j          9.+0.j        ]]
P:
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.]])
L:
array([[ 1.        +0.j        ,  0.        +0.j        ,
         0.        +0.j        ,  0.        +0.j        ],
       [ 0.28571429+0.j        ,  1.        +0.j        ,
         0.        +0.j        ,  0.        +0.j        ],
       [-0.14285714+0.j        , -0.29411765-0.j        ,
         1.        +0.j        ,  0.        +0.j        ],
       [ 0.        +0.37796447j, -0.        -0.44938587j,
         0.        -0.01252591j,  1.        +0.j        ]])
U:
array([[ 7.        +0.j        ,  3.        +0.j        ,
         2.        +0.j        , -1.        +0.j        ,
         0.        +1.41421356j, -3.        +0.j        ],
       [ 0.        +0.j        , -4.85714286+0.j        ,
        -1.57142857+0.j        ,  6.28571429+0.j        ,
         5.        -0.40406102j,  9.85714286+0.j        ],
       [ 0.        +0.j        ,  0.        +0.j        ,
         3.82352941+0.j        ,  1.70588235+2.23606798j,
         3.47058824+0.08318903j,  0.47058824+0.j        ],
       [ 0.        +0.j        ,  0.        +0.j        ,
         0.        +0.j        , -0.02800878+4.95609419j,
         1.71505978+2.29040162j, -1.        +5.56944869j]])
```

Fig 7(Matrix trial 4)

```
[7, 3, 2, -1, 1.4142135623730951j, -3]
[2.6457513110645907j, 3.3166247903554j, 1.4142135623730951j, 1.7320508075688772j, 1, -1]
[-1, 1, 4, 2.23606797749979j, 2, -2]
[2, -4, -1, 6, 5, 9]
A:
[[7, 3, 2, -1, 1.4142135623730951j, -3],
 [2.6457513110645907j,
  3.3166247903554j,
  1.4142135623730951j,
  1.7320508075688772j,
  1,
  -1],
 [-1, 1, 4, 2.23606797749979j, 2, -2],
 [2, -4, -1, 6, 5, 9]]
P:
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.]])
L:
array([[ 1.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        ,  0.         +0.j        ],
       [ 0.28571429+0.j        ,  1.         +0.j        ,
         0.         +0.j        ,  0.         +0.j        ],
       [-0.14285714+0.j        , -0.29411765-0.j        ,
         1.         +0.j        ,  0.         +0.j        ],
       [ 0.         +0.37796447j, -0.         -0.44938587j,
         0.         -0.01252591j,  1.         +0.j        ]])
U:
array([[ 7.         +0.j        ,  3.         +0.j        ,
         2.         +0.j        , -1.         +0.j        ,
         0.         +1.41421356j, -3.         +0.j        ],
       [ 0.         +0.j        , -4.85714286+0.j        ,
        -1.57142857+0.j        ,  6.28571429+0.j        ,
         5.         -0.40406102j,  9.85714286+0.j        ],
       [ 0.         +0.j        ,  0.         +0.j        ,
         3.82352941+0.j        ,  1.70588235+2.23606798j,
         3.47058824+0.08318903j,  0.47058824+0.j        ],
       [ 0.         +0.j        ,  0.         +0.j        ,
         0.         +0.j        , -0.02800878+4.95609419j,
         1.71505978+2.29040162j, -1.         +5.56944869j]])
```

Fig 8 (Matrix trial 4)

Results	Algorithm successfully complied and decomposed the matrix.
Description	For this matrix the function that where presented where changed. The matrix presented instead of if being a and ordinary 2x2 matrix, I decided instead to test the matrix and test it limits as much as possible. The matrix is instead a 3x3 matrix. This matrix shows that the algorithm works well. While it still has many limitations the algorithm has currently for now satisfied the requirements that where set. Any further improvements that can be laid out will be placed in the further implementation section of this report.
Notes	This matrix algorithm was to test the upper limits of the algorithm. Because many errors where encountered along the process of building this

	algorithm. There was some concern that the algorithm would not be able to properly decompose. So far due to the success of this algorithm it is shown that the algorithm upper limits requires the matrix to comprise of an algorithm of any even size and complexity. The algorithm however experienced immense problems with the code compiling. The high time complexity for this algorithm was bad, it took a very long time to run. While it is good that the algorithm managed to run, the compliation has proved a problem for an algorithm of this size.

Discussion

This program's coding performed admirably. This indicates that the algorithm was efficient in writing the program as needed. The algorithm seems to be working well and capable of handling a strong unitary matrix decomposition. The algorithm's smooth design demonstrates that the software can perform well. The algorithm however did have a lot of problems, these problems did not help the algorithm with processing some of the data correctly, and when it comes to some of the numbers in the unitary matrix they are not all fully functioning. This is due to the way in which this program was coded, the method in which this program was coded did not allow it to have the smoothest of compilation withall of the programs.

While the algorithm was attempted to be as clean and as precise as possible, the algorithm does not reflect what the final expected outcome was to be for this dissertation. The expected final outcome for this dissertation was the program would be able to compile properly with harm, this unfortunately was not possible. The algorithm while it does run cleanly does not run smoothly unfortunately.

The algorithm's failure can be traced back to the code's implementation. This algorithm's implementation was not necessarily the cleanest or best coordinated. The implementation of the code should have been done better. As well the high time complexity for the algorithm is feeble, this means that the algorithm requires a lot of work that can help it to be propelled forward into a more efficient and higher functioning algorithm. A lot of the problem with the algorithm and results seems to be with the compile time of the algorithm.

The algorithm, on the other hand, is not without flaws, as it is unable to perform all the tasks set forth by the program. The program performs as expected when it was created, however it struggles to adequately decompose complex numbers. A unitary matrix's aim is also to enable quantum computers to use it. This is due to the fact that unitary matrices are a critical component of quantum computers; nevertheless, this matrix lacks the essential complexity to be employed in a quantum computer.

Further improvements

Implementing the algorithm in a different method

While the algorithm managed to successfully work, I feel like the algorithm could have been written and in turn implemented in a different method. There are various methods that are used to implement an algorithm to make them work. While the method that was used works, it is felt that it is a bit too

Adjusting the software so that it operates better and more fluently is also another change that may be made. For example, changing the programming language from java to OpenQL could be an option. The reason java was chosen in the first place was because it reminded me of something familiar. However, after finishing the program and writing the report, I realized that there were still several things I might have done to better my programming and the program's efficiency. Fixing the java code so that not too many programs are printed is another adjustment that may have been made. Some suggestions are provided below.

complicated and that with further research the algorithm could work and be implemented in a different method. The algorithms that could have been used instead could have outlined the various decomposition styles presented in various articles.

Different methods that where considered was to break apart the LU decomposition section of the program and make it fully work. The reason as to why this was not done was because the program would require a rebuild of the algorithm and due to time constraints, this unfortunately was a reality that could not be perceived. While this algorithm implementation would have made way for a new unique matrix decomposition, its IDE and language and time constraints unfortunately made this very difficult for it to come true.

Changing the multiplication structure

On top of changing the algorithm structure I believe that it would have been better if the algorithm was decomposed in a different multiplication style. This is because the pivoting style that is used in the algorithm is not exactly the most present algorithm decomposition method. As well it adds an addition unnecessary layer of matrix decomposition. The pivoting function makes the matrix decomposition unnecessarily complicated and it adds an addition layer of complexity to the algorithm which I believe is no need for the algorithm to be decomposed. The reason as to why I left it in the algorithm is because the algorithm itself will not work properly with adding this additional layer of matrix decomposition.

While LU decomposition is mathematically proven to be one of the most effective ways for matrix decomposition there are many other methods that could have been used when it came to the matrix decomposition. Different decomposition methods such as Jordan decomposition and Schur decomposition could have all been used. However due to the success of the algorithm, further improving and developing on the LU decomposition could have been proven to be a much efficient and effective way to complete this algorithm.

Re defining the LU decomposition

Lu decomposition is one of if not the most popular form of matrix decomposition. The matrix decomposition that is presented in the algorithm is a slightly more complicated form of matrix decomposition, the LU decomposition that was presented in the python implementation is a bit complicated. Perhaps if a different approach was taken to completing this algorithm then possibly the LU decomposition of this algorithm could have been smoother.

Lu decomposition is a proven effective method for matrix decomposition. The algorithm that was implemented throughout this project uses LU decomposition. However, the algorithm that was implemented throughout the project seems to be complicated. Perhaps by tackling the LU decomposition in a different method it would allow the algorithm to reduce it high time complexity, as well it would help the algorithm with reintroducing the LU decomposition in a simpler method. Perhaps by creating different classes with different variable names then it would allow the algorithm to re-structure itself, this would as well allow for better and easier to read results in the long run.

Further improving on the pivot matrix

The pivot matrix is where the biggest issue in the algorithm lies. The algorithm is already functioning well without the pivot matrix. However, it is important for pivot matrix to be embedded into the

algorithm because it makes the algorithm work. This is a positive aspect of the algorithm however the overall complexity of the pivoting matrix makes the algorithm much more difficult to decompose.

The pivoting of the matrix works by creating floating points in the algorithm that can allow the values in the algorithm to decompose and restructure themselves. This aspect of the algorithm process is one of the most important if not the most important section when it comes to the unitary decomposition. However, the implementation of that section of the algorithm is not the cleanest and best organized. By taking it and putting it in a singular class, with the LU decomposition, theoretically it could be possible for the algorithm to successfully decompose the matrix.

The algorithm cannot take in imaginary numbers into its matrix

The algorithm is unable to take imaginary numbers into the algorithm. This is because when the matrix decomposition of the algorithm was being coded the algorithm did not need any of the supporting in built mathematical functions on python to work. However, that was when the algorithm was only taking in positive numbers. The algorithm had to be re written and re coded so that it can take in negative numbers on the algorithm to make it work.

While this issue does seem like it can be solved relatively easily. The algorithm unfortunately does not take any imaginary numbers as a matrix input. This is because when it was experimented with putting direct imaginary numbers in the algorithm, it unfortunately did not create the required function. The algorithm would constantly crash due to errors. Due to simplicity sake and attempting to get the best possible working algorithm it was better for the matrix decomposition to be left without any imaginary numbers as a value in the matrix.

Conclusion

In conclusion the algorithm has managed to work. The algorithm completes the requirements that were set to be completed for this project. The algorithm much like many algorithms has many positives and negatives. the outline of the functionality of the algorithm has been mentioned on the report many times throughout. However overall there is a general satisfaction with the functionality of the algorithm. The algorithm was built python coding language. There were many considerations on different languages to use while completing this algorithm. The algorithm that was completed will be in correlation with the BSc code of conduct. The method for coding the algorithm was LU decomposition. The reason as to why LU decomposition was used when it came to complete this dissertation was because LU decomposition is very effective in trying to complete the task at hand.

Due to the effectiveness the algorithm presented shows that it is a very well-functioning algorithm. This means the that the algorithm could complete the required task. The task that the algorithm was meant to complete is successfully being completed. The algorithm works very well, and it shows that these types of functions can fully run the algorithm at the pace that it is meant to be run at.

There were many problems that were encountered when attempting to complete the algorithm. One of the biggest issues with attempting to complete the algorithm was finding an effective mathematical approach that can help me solve a matrix of high complexity.

In summary the research that was completed for this dissertation was needed in order to meet the criteria of the algorithm. The algorithm in the end felt like it worked very effectively for the project at hand. The effectiveness of the algorithm showed compliance and allowed the program to work at the full capability at which it needs to be operated. The reason as to choosing LU decomposition was due to the mathematical advantage that it provided when it came time to work on the algorithm.

Many different programming languages where trialed and errored in order to complete the algorithm. The coding languages that where experimented with are python, MATLAB etc. this shows that such a project can be completed on many different levels and many different programming routes.

To conclude the effectiveness of the code and resilience of the algorithm to crack under experimentation shows a good step up in the functioning and handling of this code. This project shows how different mathematical approaches can be used when it comes to writing and handing out errors in such a way. This is a good way to show the transparency and abstract of the code that is presented.

While the program did manage to work, and it did manage it unfortunately id not manage to fully complete the requirements that were set out to me for this dissertation. The program is however capable of decomposing a unitary matrix; however, the program was meant to resemble that of a unitary matrix decomposition that is used on quantum computers. This is unfortunate because the program that was written unfortunately is not capable of completing such a task. While the program however can decompose unitary matrices it unfortunately does not meet the level of requirement that is needed for this assessment. In the future more time and as well different programming methods will need to be used for the program to successfully decompose a unitary matrix.

Project Plan

#	Activity	Start Date	Predecessor	Time Estimates in days			Expected End Date
Optimistic		**Normal**		**Pessimistic**		**Expected**	
1	Backgroundund Research h	28/10/2020	4	5	4	13	
2	Project Proposal	1/11/2020	1	5	7	9	21	4/11/2020
3	Related Work Research	28/10/2020	1	5	6	5	16	11/16/2020

4	Algorithm m and Unitary matrices practice	30/11/2020	1,2	5	5	5	15	10/2/2021
5	Coding trial and error with different	30/12/2021	1,2,4	7	5	4	16	10/2/2021
	languages							
6	Testing	10/3/2021	5	5	5	5	15	10/4/2021
7	Software Improvements	17/4/2021	6	6	5	4	15	1/5/2021
8	Draft Report	10/3/2021	4,5,6,7	7	7	4	18	4/5/2021
9	Final Report	5/5/2021	8	5	6	7	18	11/5/2021

Supervisor

References

1. *Complex, Hermitian, and Unitary Matrices*. 2021. [video] Directed by D. Farina. Youtube.
2. Giles, M., 2021. *Explainer: What is a quantum computer?*. [online] MIT Technology Review.Available at: <https://www.technologyreview.com/2019/01/29/66141/what-is-quantum- computing/>.
3. Grumbling, E. and Horowitz, M., n.d. *Quantum computing*. Washington DC.
4. Higham, N. and Pranesh, S., 2021. Exploiting Lower Precision Arithmetic in Solving SymmetricPositive Definite Linear Systems and Least Squares Problems. *SIAM Journal on Scientific Computing*, 43(1), pp.A258-A277.
5. Lakshminarayan, A., 2019. Out-of-time-ordered correlator in the quantum bakers map and truncated unitary matrices. *Physical Review E*, 99(1).
6. Quantstart.com. n.d. *LU Decomposition in Python and NumPy | QuantStart*. [online] Available at: <https://www.quantstart.com/articles/LU-Decomposition-in-Python-and-NumPy/> [Accessed 18 May 2021].
7. McKinney, W., 2012. *Python for Data Analysis*. Santa Rosa: O'Reilly.
8. Reid, M., 2014. *Pivoting for LU Factorization*. [online] Buzzard.ups.edu. Available at: <http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-reid-LU-pivoting.pdf> [Accessed May 2021].
9. Sayyed, T., 2021. *Matrix Decomposition Decoded - KDnuggets*. [online] KDnuggets. Available at: <https://www.kdnuggets.com/2020/12/matrix-decomposition-decoded.html> [Accessed May2021].

10. Yang, M., 2021. *Matrix Decomposition*. [online] Users.eecs.northwestern.edu. Available at: <http://users.eecs.northwestern.edu/~mya671/files/Matrix_YM_.pdf> [Accessed May 2021].
11. Krol, A., Sarkar, A., Ashraf, I., Al-Ars, Z. and Bertels, K., 2021. *Efficient decomposition of unitary matrices in quantum circuit compilers*. University of Porto, Portugal.