How does Grover's Quantum Search Algorithm function and will it ever replace classical algorithms?

Aarambh Sinha 9648
Altrincham Grammar School for Boys 33611
June 2017

Abstract

This project aims to give a brief introduction into the theories behind quantum computing and provide a visual realisation as to how Grover's Quantum Search Algorithm functions. The paper is split into 3 parts where ideas are discussed about quantum physics and how we can use the standardised methods, such as Bloch's Sphere to be able to demonstrate the mathematical properties of a quantum bit. The second section delves into existing solutions and currently available simulation systems to provide a visual representation of the algorithm The purpose of my work is to be able to visualise the effects matrix transformations (quantum gates) have on individual qubits and how the state of a qubit changes throughout the algorithm. The final part analyses the logistics and limitations current solutions have and the problems that affect the algorithm and quantum computers themselves. By looking at the nature of algorithms, we soon realise that certain classical algorithms will not be replaced.

To see how the state of a quantum bit changes in every step, I pass in state vectors through predetermined phase differences to show the position of the state vectors in every process. I use QuTiP, which is a package written to simulate quantum environments. What I soon discovered is that the algorithm is quite probabilistic in its nature and does not always provide a solution to an NP-Hard problem.

Content

Part I - Foundations

1.1 Fundamental Understanding

1.1.1 History of Quantum Mechanics

During the early 1900s, multiple theories were suggested to explain different phenomena in subatomic particles. These theories introduced multiple exceptions due to the explanation not being self sufficient. Physicists therefore started disregarding these hypotheses and started working on the fundamental principles of quantum mechanics. Researches such as Max Planck and Louis de Broglie were able to mathematically derive explanations such as the wave/particle duality of photons as well as the structure of the atom and DNA.

Essentially, this set of principles form the foundations of quantum computing as researchers have managed to extract usefulness out of the erratic behaviour. By studying the behaviour of these particles, we are able to use probability to find the solution to the problem. By theory, this is completely different to how the computers, that we use, work. Binary based systems are dependent on deterministic solutions to problems. That means the core concept of Quantum Computing is fundamentally different.

Despite the research dating back to the 80s, quantum computing is still in a premature stage. The access to information has only been available to researchers that have spent their life in this field. Even though we are decades away from a fully functioning quantum computer that can be accessed by everyone, we have started to put the pieces together. You could say this is a reflection of the past, comparing today's work in quantum computing to the first use of transistors and combining them together to create a binary based system. Very recently, we have managed to implement two and three-qubit algorithms using photons, nuclei of atoms and energy state of electrons. This is different to classical bits, which tend to just be electric currents passed through gates in a processor. As we have a variety of systems to indicate qubits, we are still at the stage of analysing the effectiveness of every method.

But what makes Quantum Computing so intrinsically important is the idea of 'Quantum Entanglement'. As applications of Quantum Computing are focused on analysing a pair of objects, if they are entangled, they are deeply related to each other. Therefore if we know the state of one object, we can determine the state of another. The speed of determination can be considered to be 'faster than light'. You may be reading this paragraph with a raised eyebrow, you are not alone as this very idea seemed to go against Einstein's theory of Relativity. Einstein never accepted this theory and instead insisted particles had 'secret information' and communicated with each other. Since then, researchers have conducted multiple information and have found a random set of data for both particles. But what the data had shown was that both particles had opposite states defined to each other. So while disproving secret information theory, researchers have shown there is not any communication occurring at speed faster than light leaving the Theory of Relativity intact.

1.1.2 Using Dirac's Notation

When performing calculations using imaginary numbers in matrices, it is important to use Dirac's Notation. Traditionally known as the 'Bra-ket pair, we can represent matrices in a specific form that is essential for Schrodinger's wave mechanics. The pair is made up of two parts ; bra $\langle |$ and ket$| \rangle$. In order to understand what role these pairs have in Quantum Mechanics, we have to agree with the fact that fundamental particles exhibit a wave-particle duality. Therefore the wave function of the particle we are observing is $| \phi \rangle$. We have shown the wave function in the Ket form. Bra $\langle |$ is supposed to represent the value of the final state and ket $| \rangle$ is the state of an unknown constant we are looking for. To find the wave function, we need a constant value of the state $|0\rangle$ and $|1\rangle$ which is α and β respectively. α and β are complex numbers which means they are a combination of real and imaginary numbers. Despite the object or particle being in a 'real state', we are mathematically deriving abstract qualities.

$|\phi\rangle = \alpha|0\rangle + \beta\ |1\rangle$.

Every time we measure the qubit, we will only get the answer as either 1 or 0. This is not a deterministic quality, like with classical bits. With qubits, it is a probabilistic answer and thats what superposition is known as. To work out the probability using the same conditions, we will go back to the equation. $|\alpha|^2$ will give a probability of the qubit being measured as 0 and $|\beta|^2$ will give a probability of the qubit being measured as 1. As these are the only two possibilities;
$|\alpha|^2 + |\beta|^2 = 1$

1.1.3 Understanding Bloch's Sphere For a Single Qubit

Using the complex numbers alpha and beta, we can say the qubit's state is a unit vector in a two dimensional complex plane. This means that each qubit has 4 unique parameters with real magnitudes. However after we normalise ϕ, we are essentially left with just the magnitude of the vector. A physicist called Felix Bloch designed a simple representation to visualise how these 4 parameters affect the probability.

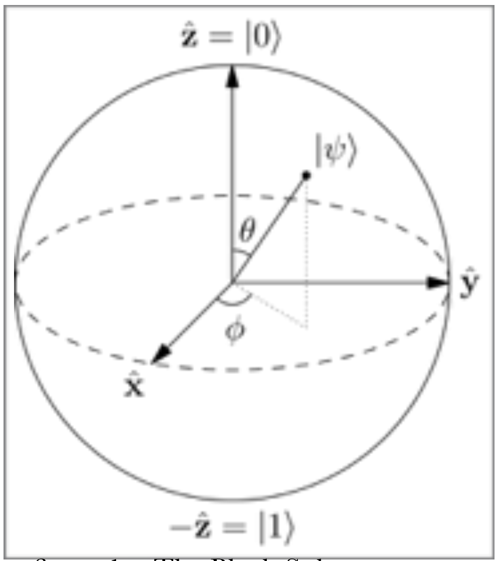


figure 1 - The Bloch Sphere

$| \phi \rangle = \alpha| 0 \rangle + \beta\ | 1 \rangle$ can be written as:
$| \phi \rangle = \cos(\theta/2)| 0 \rangle + e^{i\phi}\sin (\theta/2)| 1 \rangle$

θ and ϕ are both parameters that hold value of angles of the vector in respect to the z axis and x axis. As we have addressed two parameters, we must address their range.
$0 <= \theta <= \pi$
$0<= \phi <= 2\pi$
Using the explanation in 1.1.3 and if the qubit is a halfway state;
$(\cos(\theta/2))^2 + (e^{i\phi}\sin (\theta/2))^2 = 1$
therefore
$(\cos(\theta/2))^2 = 0.5$

We can now work out the value of θ and ϕ thus finding the value of 2 of the 4 parameters These two parameters are essential for working out the probability of the two measured states of a Qubit. Nielson and Chuang referred to this system and asked 'how much information could a Qubit represented?'. The argument they stated was 'as there are an infinite number of points on the unit sphere, so that in principle one could store an entire text of Shakespeare in the infinite binary expansion of θ. They then further explained this notion of the Bloch Sphere representation is incorrect as we measure the state of the Qubit, the superposition collapses down to the definite states |0⟩ and |1⟩. When we say collapse, what Computer Scientists mean is that if a Qubit in superposition is measured (using various techniques) and the output is 0, the state of the Qubit is now |0⟩. Nielson and Chuang colloquially called this behaviour part of the 'fundamental postulates of quantum mechanics'.

As mentioned Qubits are logically different to how classical bits function. If we were to measure a classical bit, we will receive the same output 100% of the time. We can use an analogy of a coin flip to explain how classical bit measurement works. If we have a coin, we do not know the state of it (whether its head's up or not). We flip it (ie put it through a function) and we look at what state its in. At this point, the 'fait' of the coin is already determined but we do not know about it yet. Comparing that to Qubit, we will never know the 'fait' of the Qubit's state for it does not exist. The state of that Qubit will be measured at a certain time and it will be subject to change.

1.2 Building a Quantum Computer

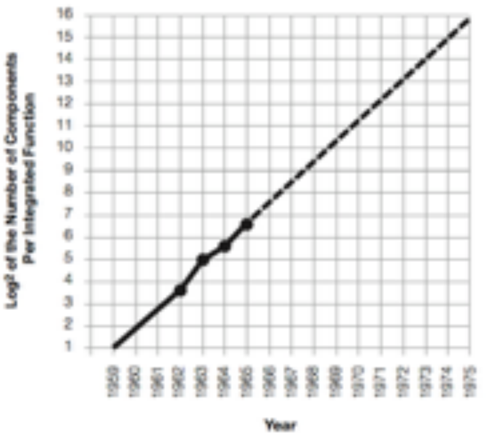


figure 2 - Linear relationship of Log^2 of number of components per area against time

1.2.1 Moore's Law and Quantum Tunnelling

In 1965, Gordon Moore published an article in Electronics showing his hypothesis of the future of integrated electronics. This is what was written at a time where there was a big boom in silicon semiconductor research:

'The complexity for minimum component costs has increased at a rate of roughly a factor of two per year'

shows that during early 1960s, the power of cheap microprocessors was doubling every year. However, if we compare this to the excerpt published in 2005:

'…Dave House, who used to work here at Intel, did that, he decided that the complexity was doubling every two years and the transistors were getting faster, that computer performance was going to double every 18 months…'

we know his hypothesis had little to do with ground breaking scientific research into the semiconductor industry and more to do with Intel's Economic plan, calculations show that by 2019, the Moore's Law will come to an end due to transistors smaller than 7nm potentially experiencing quantum tunnelling effects.

Despite Moore's Law proving to be largely correct over the decades, it is fast approaching a dead end due to Quantum phenomena interfering with minuscule transistors. Researchers have designed single atom transistors but the effects of quantum tunnelling are becoming prevalent. One of the issues researchers are facing is shown in figure 3. The white blob is a packet of electrons. As current is passing through, there is a flow of electrons occurring. If we place a 'potential barrier' through the space, it should stop electrons from going though to the right hand side. The strange effect of 'quantum tunnelling' shows that some electrons actually pass through the barrier. This can be further explained by using the wave-particle duality model but scientists can only observe WHAT happens rather than explain HOW these phenomena are possible. Quantum Computer Scientists do not need to understand how this phenomena occurs but they can use this behaviour to implement a Quantum Computer.

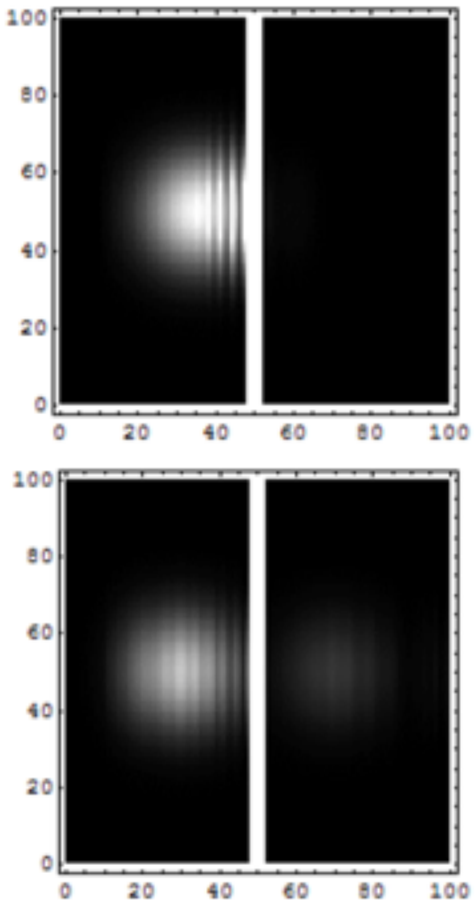


figure 3 - De Broglie Wave incident on a barrier

1.2.2 Multiple Qubit Formation

In 1.1, we have stated there are 4 parameters when a Qubit is in a complex state, we have also mentioned those 4 parameters can not be measured. Does this mean Einstein was correct about there being 'hidden information'? Possibly, but what Computer Scientists believe is more important is that this 'hidden information' is remembered so it can technically be described and so despite the rate of hidden 'information growing exponentially with the number of Qubits', we can use this form of memory as a 'powerful tool for information processing'.

Previously, we mentioned measuring a qubit causes it to collapse down to a binary state, one or zero, so in a way, we could say that information is being passed through a quantum gate. A classical gate takes in one or more sources of information and outputs certain information if certain requirements are met. Simple examples are NOT, AND and OR gates. Quantum computers have their own set of gates that physically change the orientation or direction of qubits.

Fundamental Understanding

When studying GCSE Physics, many students would have come across a set of logic gates. These logic gates are classical, in that they take in classical bits and output definite classical bits. The simplest example one can give is a NOT gate.

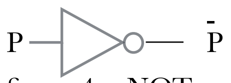


figure 4 - NOT gate

Unlike the classical NOT gate, the CNOT gate has two inputs, one acts as a 'control' and the other is the target qubit. We can write the inputs in the form of;

$| 0\ 1 \rangle$

The first number denotes the control. What the control qubit states is that the transformation will only take place with the target qubit if the first number is 1. How do we know what the values are for the target and control? We will not as the qubits are always in a state of superposition. Instead we have to wait for the program to complete, measure the qubit and then backtrack through the series of gates.

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This gate therefore needs two inputs and there are 4 possible states.:
$| 0\ 0 \rangle$, $| 0\ 1 \rangle$, $| 1\ 0 \rangle$, $| 1\ 1 \rangle$
We could write these possible states to make it possible to now do matrix multiplication
[0,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{X} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Another important quantum gate is the Walsh-Hadamard transformation. We can again represent this transformation in a matrix which can then be used in the future for a lot of calculations in the future.

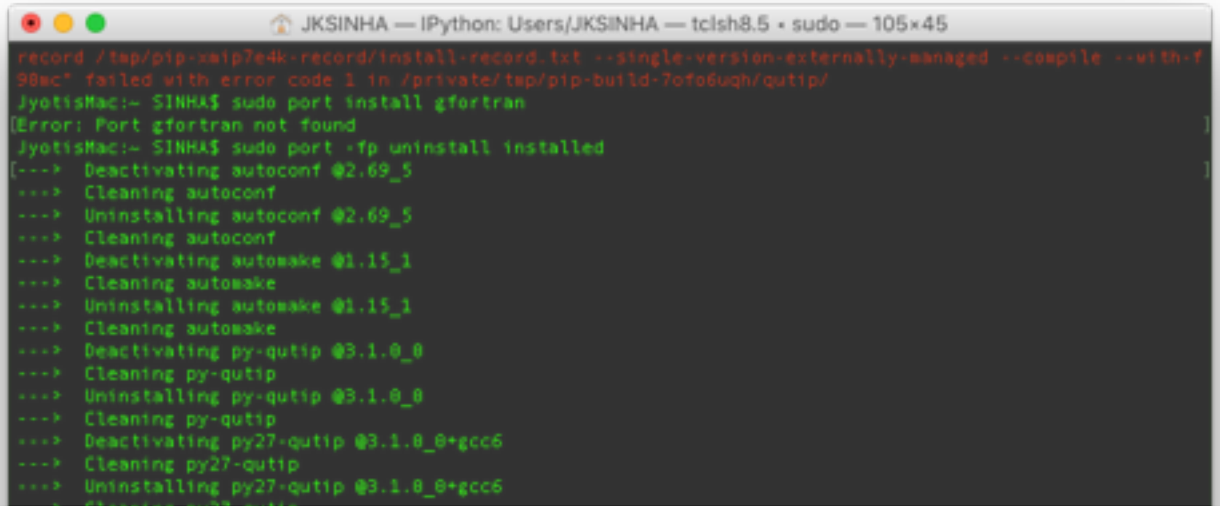
$$H = \quad (1/(\sqrt{2}) \quad \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This is a unitary matrix so if we multiply H by its inverse H', we get the identity
H H' = I

It is important to note that these gates are only given unitary values to make it easier for the user to understand how the program is being processed. Similar effect is therefore pushed onto classical based simulations of quantum computers. If we restrict the transformations to classical operators, we lose the ability to parallel process multiple bits of data. A qubit gate works as a Same Instruction Multiple Data if we compare it to a classical processor. If a qubit gate has the same classical operators as a generic logic gate, it is now effecitvely working as a classical computer. What we can therefore deduce from this is that a quantum computer can now act as a classical computer but its processing speed is no faster than it (perhaps slower). This gradually foreshadows the idea that classical computers cannot be replaced by quantum computers for binary based algorithms and infrastructure. But we can also deduce that quantum computers can be designed to work as co processors, such as Graphics Processing Units.

Part II - The Algorithm

2.0 Systems Configuration

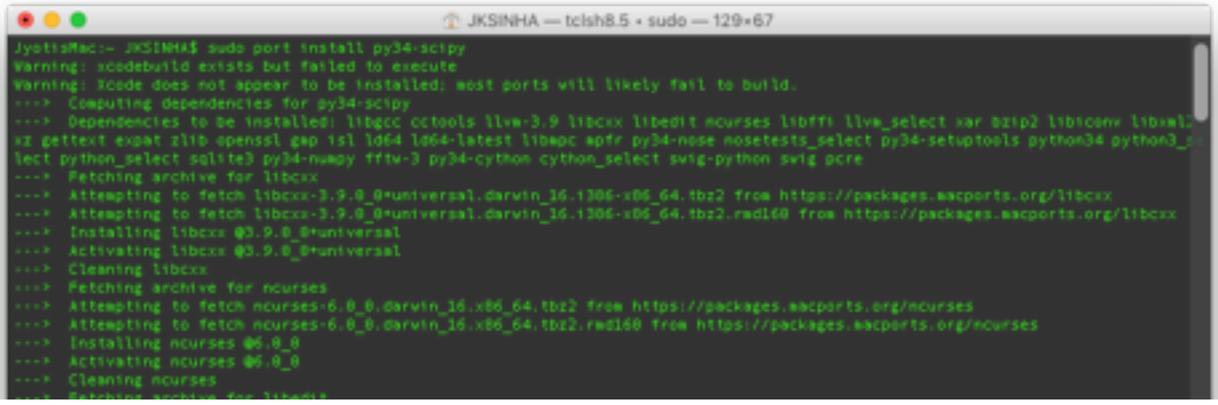


```
record /tmp/pip-xmip7e4k-record/install-record.txt --single-version-externally-managed --compile --with-f
90mc" failed with error code 1 in /private/tmp/pip-build-7ofo6uqh/qutip/
JyotisMac:~ SINHA$ sudo port install gfortran
Error: Port gfortran not found
JyotisMac:~ SINHA$ sudo port -fp uninstall installed
--->  Deactivating autoconf @2.69_5
--->  Cleaning autoconf
--->  Uninstalling autoconf @2.69_5
--->  Cleaning autoconf
--->  Deactivating automake @1.15_1
--->  Cleaning automake
--->  Uninstalling automake @1.15_1
--->  Cleaning automake
--->  Deactivating py-qutip @3.1.0_0
--->  Cleaning py-qutip
--->  Uninstalling py-qutip @3.1.0_0
--->  Cleaning py-qutip
--->  Deactivating py27-qutip @3.1.0_0+gcc6
--->  Cleaning py27-qutip
--->  Uninstalling py27-qutip @3.1.0_0+gcc6
```

figure 4 - Removing all files for fresh install

2.0.1 Setting up QuTiP and Tex

In order to move on to the second part of this project, exploring the further mechanics of this model, I began planning to start designing on Python using the QuTiP toolbox. This is an open source module that can be installed on Python and runs calculations at C-code level using the Numpy and Scipy dependencies. Despite using Python in the past, I had to remove all the package managers that were causing clashing problems when installing all the dependencies. This meant uninstalling all the managers and Python related programs.
The dependencies were then installed using MacPort's package manager.



```
JyotisMac:~ JKSINHA$ sudo port install py34-scipy
Warning: xcodebuild exists but failed to execute
Warning: Xcode does not appear to be installed; most ports will likely fail to build.
--->  Computing dependencies for py34-scipy
--->  Dependencies to be installed: libgcc cctools llvm-3.9 libcxx libedit ncurses libffi llvm_select xar bzip2 libiconv libxml2
xz gettext expat zlib openssl gmp isl ld64 ld64-latest libmpc mpfr py34-nose nosetests_select py34-setuptools python34 python3_se
lect python_select sqlite3 py34-numpy fftw-3 py34-cython cython_select swig-python swig pcre
--->  Fetching archive for libcxx
--->  Attempting to fetch libcxx-3.9.0_0+universal.darwin_16.i386-x86_64.tbz2 from https://packages.macports.org/libcxx
--->  Attempting to fetch libcxx-3.9.0_0+universal.darwin_16.i386-x86_64.tbz2.rmd160 from https://packages.macports.org/libcxx
--->  Installing libcxx @3.9.0_0+universal
--->  Activating libcxx @3.9.0_0+universal
--->  Cleaning libcxx
--->  Fetching archive for ncurses
--->  Attempting to fetch ncurses-6.0_0.darwin_16.x86_64.tbz2 from https://packages.macports.org/ncurses
--->  Attempting to fetch ncurses-6.0_0.darwin_16.x86_64.tbz2.rmd160 from https://packages.macports.org/ncurses
--->  Installing ncurses @6.0_0
--->  Activating ncurses @6.0_0
--->  Cleaning ncurses
```

figure 5 - Reinstalling all dependencies

- Numpy

Powerful array object initialisation tools, Fourier transform functions.

- IPython

Powerful interpreter to see effect on qubits during functions.

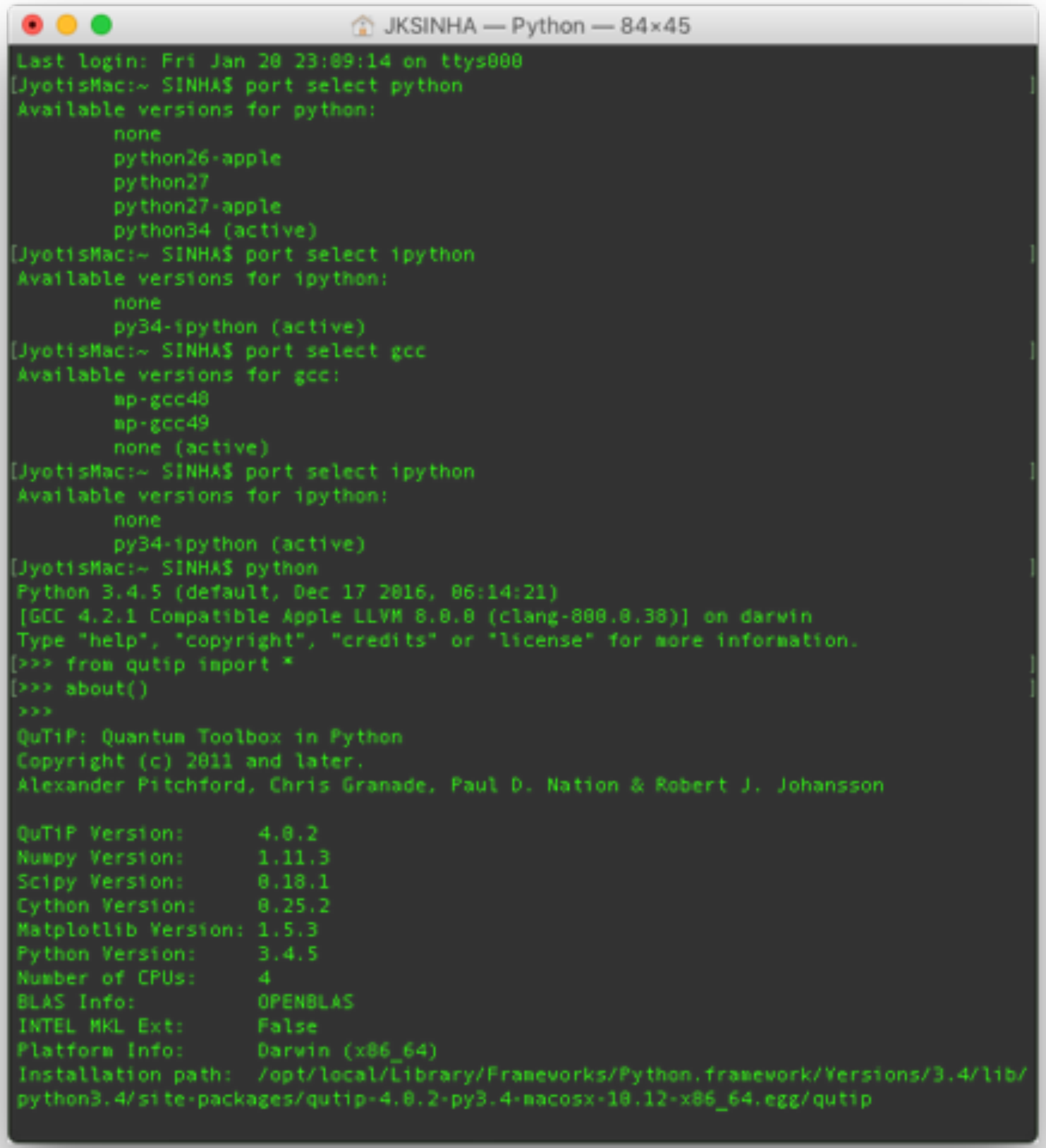
- Matplotlib

2D plotting library.

- Cython

Platform for C oriented programming.

After insuring the correct dependencies were selected, the Python interface was opened to check QuTiP was correctly installed. The lower part shows the dependencies' versions as well as the processing cores. This is import for future reference as the model gets optimised for multiple cores.



```
Last login: Fri Jan 20 23:09:14 on ttys000
[JyotisMac:~ SINHA$ port select python
Available versions for python:
        none
        python26-apple
        python27
        python27-apple
        python34 (active)
[JyotisMac:~ SINHA$ port select ipython
Available versions for ipython:
        none
        py34-ipython (active)
[JyotisMac:~ SINHA$ port select gcc
Available versions for gcc:
        mp-gcc48
        mp-gcc49
        none (active)
[JyotisMac:~ SINHA$ port select ipython
Available versions for ipython:
        none
        py34-ipython (active)
[JyotisMac:~ SINHA$ python
Python 3.4.5 (default, Dec 17 2016, 06:14:21)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> from qutip import *
[>>> about()
>>>
QuTiP: Quantum Toolbox in Python
Copyright (c) 2011 and later.
Alexander Pitchford, Chris Granade, Paul D. Nation & Robert J. Johansson

QuTiP Version:      4.0.2
Numpy Version:      1.11.3
Scipy Version:      0.18.1
Cython Version:     0.25.2
Matplotlib Version: 1.5.3
Python Version:     3.4.5
Number of CPUs:     4
BLAS Info:          OPENBLAS
INTEL MKL Ext:      False
Platform Info:      Darwin (x86_64)
Installation path:  /opt/local/Library/Frameworks/Python.framework/Versions/3.4/lib/
python3.4/site-packages/qutip-4.0.2-py3.4-macosx-10.12-x86_64.egg/qutip
```

figure 6 - Priority modes configured on macOS terminal

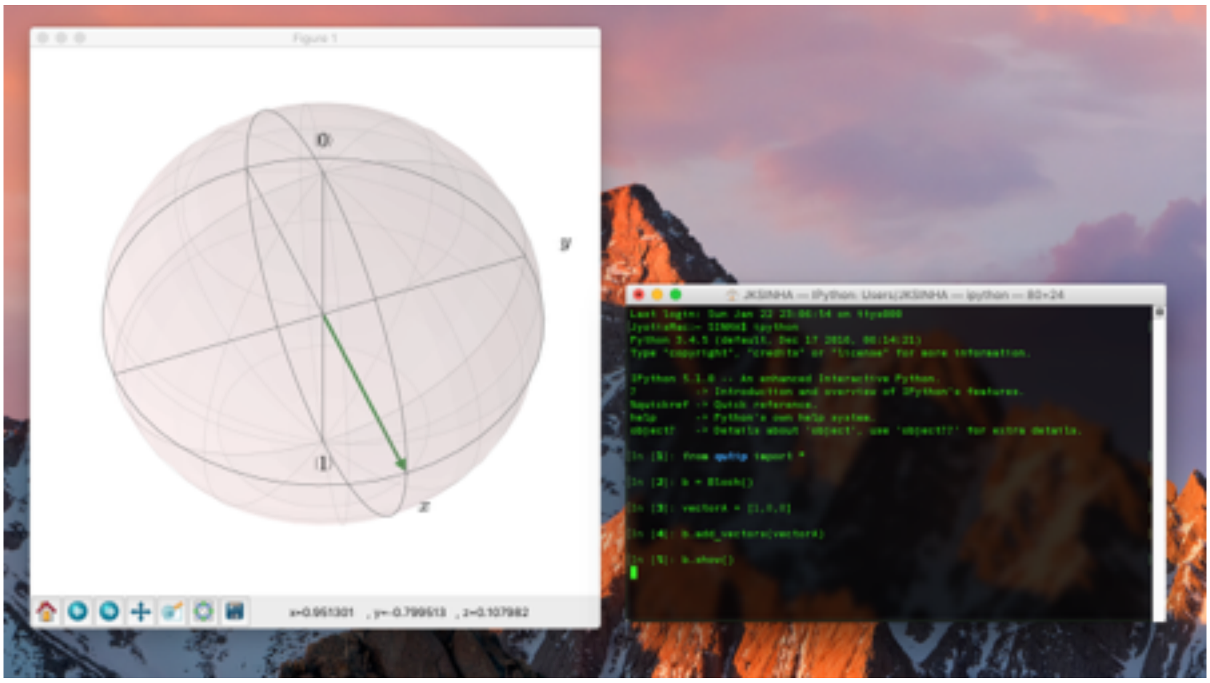


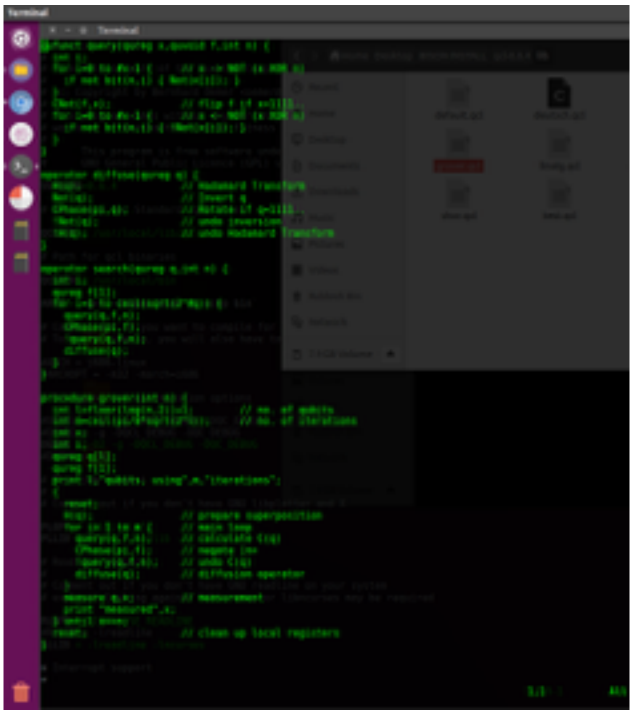
Figure 1
|0⟩
|1⟩
y
z
x=0.951301 , y=-0.799513 , z=0.107982
JKSINHA — IPython: Users/JKSINHA — ipython — 80×24

```
Last login: Sun Jan 22 23:06:14 on ttys000
JyotiMac:~ SINHA$ ipython
Python 3.4.5 (default, Dec 17 2016, 00:14:21)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from qutip import *

In [2]: b = Bloch()

In [3]: vector4 = [1,0,0]

In [4]: b.add_vectors(vector4)

In [5]: b.show()
```

figure 6 - Testing QuTiP

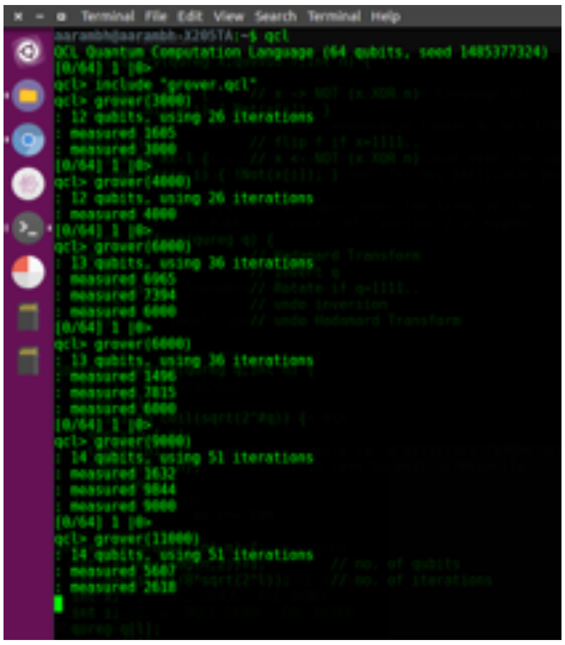
2.0.2 Data Analysis from Quantum Computing Language/ C++

In order to create a set of results to see the effectiveness of Grover's algorithm, I have installed QCL on Ubuntu using G++. One of the sample codes given in their library is Grover's search. The image on the right shows the code for the algorithm. The syntax used in QCL is very similar to C and Python. This allows me to create an automated set of data and be mapped onto a graph. On a 64bit system, the users are given 64 qubits. This creates a maximum of 1.844674407E19 items that can be searched through. The only problem is that using this system yields results very slowly. This is why it does not make sense to run these algorithms on classical computers as they cannot hold so much information and be put through functions.



```
qufunct query(qureg x,quvoid f,int n) {
  int i;
  for i=0 to #x-1 {          // x -> NOT (x XOR n)
    if not bit(n,i) { Not(x[i]); }
  }
  CNot(f,x);                 // flip f if x=1111..
  for i=0 to #x-1 {          // x <- NOT (x XOR n)
    if not bit(n,i) { !Not(x[i]); }
  }
}

operator diffuse(qureg q) {
  H(q);                      // Hadamard Transform
  Not(q);                    // Invert q
  CPhase(pi,q);              // Rotate if q=1111..
  !Not(q);                   // undo inversion
  !H(q);                     // undo Hadamard Transform
}

operator search(qureg q,int n) {
  int i;
  qureg f[1];
  for i=1 to ceil(sqrt(2^#q)) {
    query(q,f,n);
    CPhase(pi,f);
    !query(q,f,n);
    diffuse(q);
  }
}

procedure grover(int n) {
  int l=floor(log(n,2))+1;          // no. of qubits
  int m=ceil(pi/8*sqrt(2^l));       // no. of iterations
  int x;
  int i;
  qureg q[l];
  qureg f[1];
  print l,"qubits, using",m,"iterations";
  {
    reset;
    H(q);                    // prepare superposition
    for i= 1 to m {          // main loop
      query(q,f,n);          // calculate C(q)
      CPhase(pi,f);          // negate |n>
      !query(q,f,n);         // undo C(q)
      diffuse(q);            // diffusion operator
    }
    measure q,x;             // measurement
    print "measured",x;
  } until x==n;
  reset;                     // clean up local registers
}
```

figure 7 - Source code of Grover's on QCL



```
aarambh@aarambh-X205TA:~$ qcl
QCL Quantum Computation Language (64 qubits, seed 1485377324)
[0/64] 1 |0>
qcl> include "grover.qcl"
qcl> grover(3000)
: 12 qubits, using 26 iterations
: measured 1605
: measured 3000
[0/64] 1 |0>
qcl> grover(4000)
: 12 qubits, using 26 iterations
: measured 4000
[0/64] 1 |0>
qcl> grover(6000)
: 13 qubits, using 36 iterations
: measured 6965
: measured 7394
: measured 6000
[0/64] 1 |0>
qcl> grover(6000)
: 13 qubits, using 36 iterations
: measured 1496
: measured 7815
: measured 6000
[0/64] 1 |0>
qcl> grover(9000)
: 14 qubits, using 51 iterations
: measured 1632
: measured 9844
: measured 9000
[0/64] 1 |0>
qcl> grover(11000)
: 14 qubits, using 51 iterations
: measured 5607
: measured 2618
```

figure 8 - Testing different values of N and time outcome

Number of Items	Qubits Used	Iterations
50	6.0	4.0
100	7.0	5.0
200	8.0	7.0
400	9.0	9.0
500	9.0	9.0
750	10.0	13.0
1000	10.0	13.0
2000	11.0	18.0
3000	12.0	26.0
4000	12.0	26.0
5000	13.0	36.0
6000	13.0	36.0
8000	13.0	36.0
10000	14.0	51.0
12000	14.0	51.0
14000	14.0	51.0
16000	14.0	51.0
20000	15.0	72.0
25000	15.0	72.0
30000	15.0	72.0
35000	16.0	101.0
40000	16.0	101.0
45000	16.0	101.0
75000	17.0	143.0
100000	17.0	143.0
500000	19.0	285.0
750000	20.0	341.0
1000000	20.0	393.0
2000000	21.0	556.0
5000000	23.0	879.0
7500000	23.0	1076.0
10000000	23.0	1242.0
50000000	26.0	2777.0
100000000	27.0	3926.0

This is the moment we can truly see the relationship between the number of items, number of iterations required and the number of qubits required. This algorithm uses the simplified (pi/8) * (√N) method. This is the set of results we got from the algorithm.

Theoretically, the relationship between the number of iterations (y axis) should be an exponential of the number of qubits required (x axis). Previously we discussed that the number of items N is equal to 2^n where n is the number of qubits. We will also discuss why the number of iterations is equal to constant * root(N). Therefore number of iterations is equal to constant * root(2^n). This now asks the question, what is the relationship like? One may hypothesise that the relationship closely resembles an exponential function as we use powers. We are also square rooting the overall number of items. So theoretically, the graph should look like this (red line).

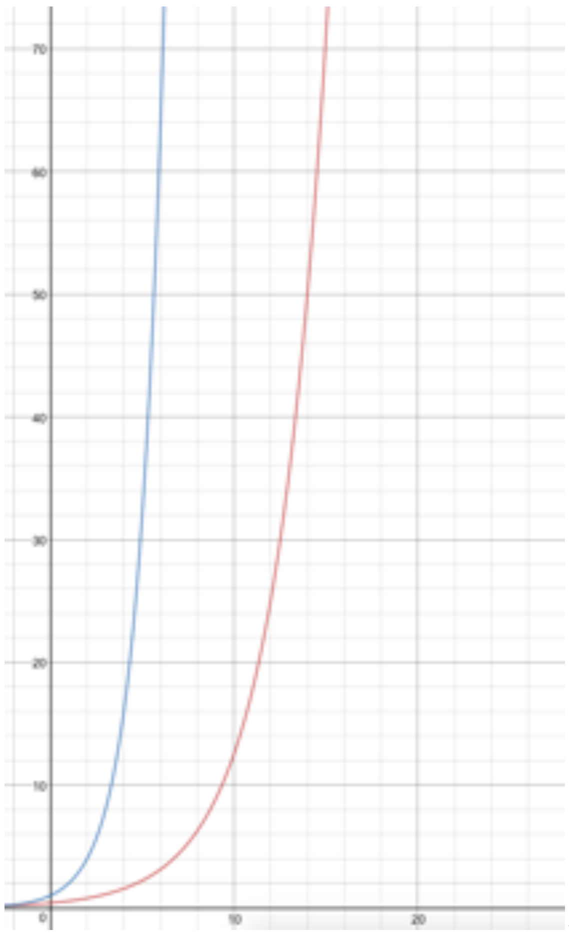


figure 8 - Relationship between number of qubits and iterations needed

However, if we actually develop a search space on QCL's platform, we can now figure out how many qubits and iterations are actually required. The relationship looks very similar, but there seem to be some anomalous results.

Whether these results can be considered anomalous is debatable, however my interpretation is that this is what would look like anyway if we have a discrete number of qubits and number of items that need to be searched through. Whilst it may seem fair to call it a function of an exponential graph, there are slight irregularities that affect the overall curve so we cannot always rely on these graphs to calculate the parameters.

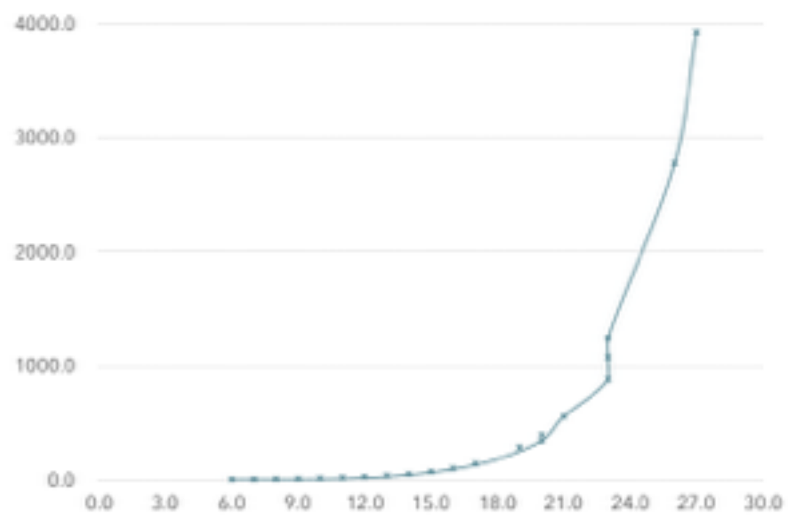


figure 9 - Relationship between number of qubits and number of items

What we can analyse however is how many iterations a single qubit can now has to go through as the number of qubit grows. This supports the theory that qubits can hold number up to an exponential amount compared to the classical bit. A typically processor nowadays has 64bit architecture. What this means is that the size of the ALU is 64 bits big. That means it can support numbers of up to 2^64. That size of really incomprehensible. But in terms of quantum computing, that is a very limiting number. Quantum algorithms are designed to process numbers with 1000+ digits. Another relationship we can also deduce that the number of iterations grows exponentially. Perhaps this can be a problem with classical algorithms but a quantum gate is designed to naturally parallel process decisions. The explanation as to why is explained in 1.2.x.

2.0.3 Program Flow

We have established that quantum algorithms are a product of multiple qubit manipulations by using transforming states and then measuring these states to give us a probabilistic answer. This means the typical quantum programming language has to follow the same kind of ideals in programming paradigms. We have quantum gates which control the quantum computation but we have to follow the classical logical flow of programming.

2.1 Grover's Search Algorithm

2.1.1 Designing a Query function

Before we start going through the list of unordered items, we will have to use a device called the Oracle. We do not need to know the innards of this device but what this device does is tell us if the item is in the list and thus if the problem is solvable. Mathematically speaking, we have a list of N items. y(x) is the item we are searching for and x is the index of the item in list with range, $0 < N - 1$. As we previously mentioned, increasing the number of qubits increases the amount of value stored exponentially. This means we will need at least a certain number of qubits to hold all the items. n is the number of qubits required to hold N. This can be calculated by $N = 2^2$. The list can have multiple solutions, the number of solutions = M.

Next, we need to set up a query condition. This involves the number of qubits required as n. We can now call the condition to be B^n , this can now be implemented on all the eigenstates of the total number of qubits as a superposition. This superposition is the transformation of the qubit states. The states in all the qubits is unified and at 0. The Hadamard transformation therefore applies a superposition.

By knowing how many items there are in the unordered list, we have worked out the number of qubits required. We can also now work out the number of iterations it should theoretically take to find the item we are looking for. To do so, we can use a function of the cotangent and the number of items we have. The simplest way to explain is that we have N number of items and so we have to find the square root reciprocal of it. So we take N, square root it to$\sqrt{N}$. The reciprocal is therefore $(1/(\sqrt{N}))$. As this is a function of N, we can say $(1/(\sqrt{N}))$ is the amplitude We can therefore use this value in an inverse trig function to find the number of iterations required. So we take $(1/(\sqrt{N})$ and insert it into the arccos/arcsin function and multiply by 1/2.

$$\text{Number of Iterations} = \frac{\arccos\left(\frac{1}{\sqrt{N}}\right)}{2\arcsin\left(\frac{1}{\sqrt{N}}\right)}$$

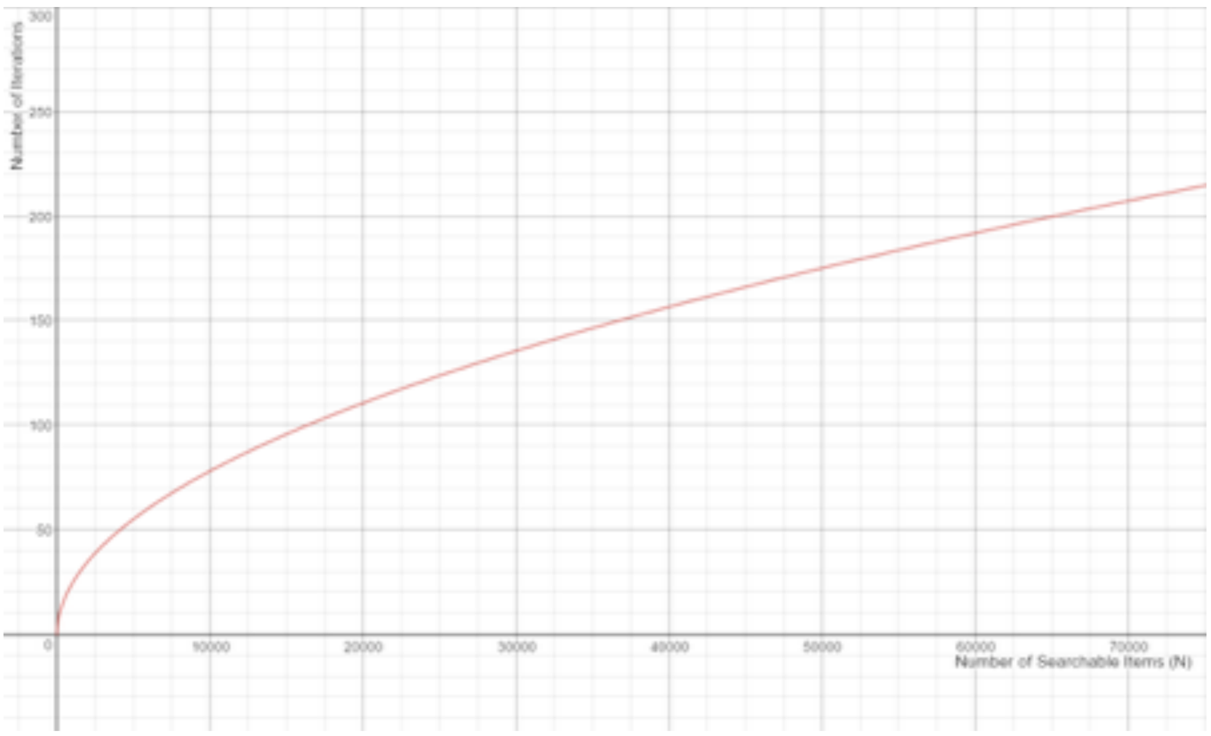


figure 10 - Relationship between number of iterations and number of items

2.1.2 Initialising Search Space and Qubits

We know how many qubits are being used so there are n number of qubits with the equal state of 1/√N after a standard function has been applied to them. This is done by applying the Walsh-Hadamard transformation matrix onto each qubit with the collapsed state of 0

((pi/8) * squareroot(N))

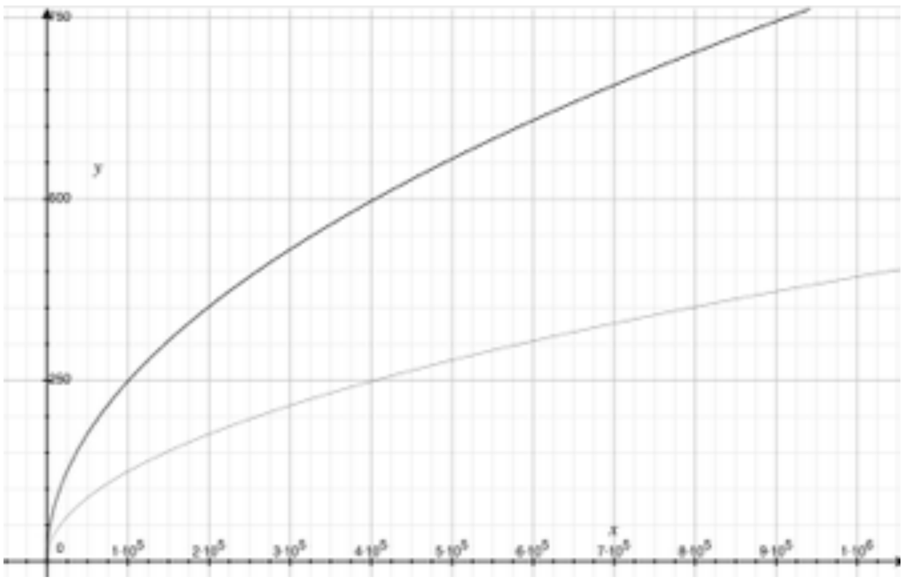


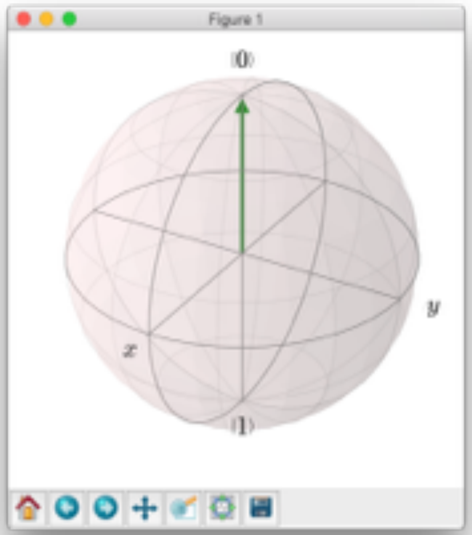
figure 11 - Average relationship between number of iterations and number of items (if probability of noise is > 0.5 and == 0.5)

As we can see the graph has the same relationship as the transformed arccos/arcsin function we previously used.

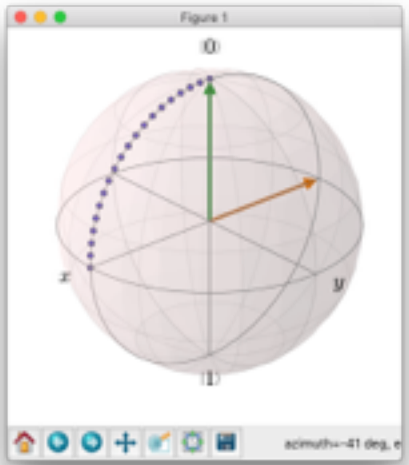
2.1.3 Main Loop - Phase Inversion and Creating Equal Superposition

When we first initialise a qubit, we know it is always at a basis state. This is either 0 or 1. For this instant, we could say the basis is 0.

```
from qutip import *
#modules initialised and called
import numpy as np
#numpy called as np to perform pure mathematical calculations
import math
import time

b= Bloch()
#a class which holds various bits of data regarding a 3 dimensional spherical grid
#This is stored in the named memory location b

startingvector = [0,0,1]
#choose an arbitrary state for the starting position of the qubit
b.add_vectors(startingvector)
```



```
#vector appended to the class as a directional vector
xz = [np.sin(th) for th in np.linspace(0,(math.pi/2), 20)]
#set of x coordinates to move from start to finish in 20 steps

yz = np.zeros(20)
#as we are rotating about y, we do not need the y coordinates to move
zz = [np.cos(th) for th in np.linspace((0), (math.pi/2), 20)]
# set of z coordinates to move from start to finish in 20 steps
vectorpoints = [xz, yz, zz]
#set of vectors are now stored on a single multidimensional array

b.add_points(vectorpoints)
```

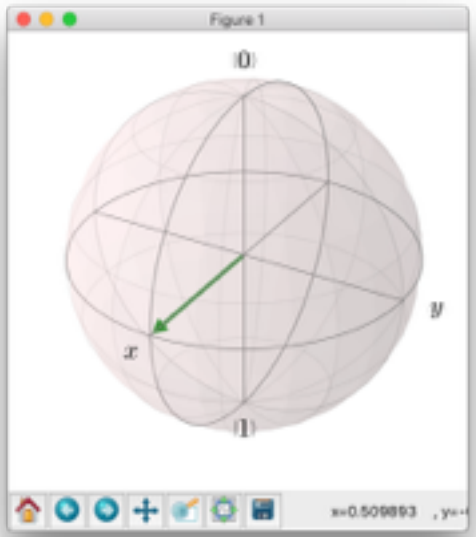


2.1.4 Main Loop - Diffusion Operator

Once we have initialised an equal superposition, we can now feed qubits through a diffusion operator. This is made up of a combination of quantum gates, as well as classical based gates. This is the main component of the algorithm as we are modifying the state of individual qubits. In a classical based simulation, we have to program it in a cyclical nature, therefore we use a for loop starting from 1 to the total number of iterations required, unlike a classical based search algorithm, we can not find the item in fewer iterations by chance, instead we have to iterate the loop for the number of iterations regardless of finding the item or not.

The qubit is first given a deterministic property and initialised:

```
from qutip import *
#modules initialised and called
import numpy as np
#numpy called as np to perform pure mathematical calculations
import math
import time

b= Bloch()
#a class which holds various bits of data regarding a 3 dimensional spherical grid
#This is stored in the named memory location b

startingvector = [1,0,0]
```



```
#choose an arbitrary state for the starting
position of the qubit
b.add_vectors(startingvector)
#vector appended to the class as a directional
vector

b.show()
```

Let’s assume we are now already in an equal superposition. The main loop algorithm goes as:

- Query
- CPHASE
- Undo Query
- Hadamard
- NOT
- CPHASE
- Undo NOT
- Undo Hadamard

and we repeat this m number of times where m is the number of iterations needed to complete.

The query phase occurs as a controlled not gate. Think of it as an XOR gate. We invert the state of our primary qubit if the control qubit has a basis state of 1. If this occurs, we flip the state vector or apply a phase difference of pi about the x axis.

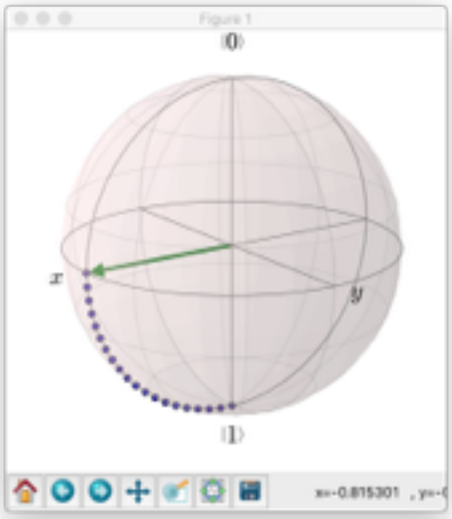
The next process is the CPHASE gate. This is similar to CNOT but in this situation both inputs need to be in a basis state of 1 in order for the qubits to have a rotation of state by pi/4 about the z axis. The basis state becomes $e^{i\phi}| 1 \rangle$. This means the probabilities do not actually change but we modify the phase difference in the qubit.

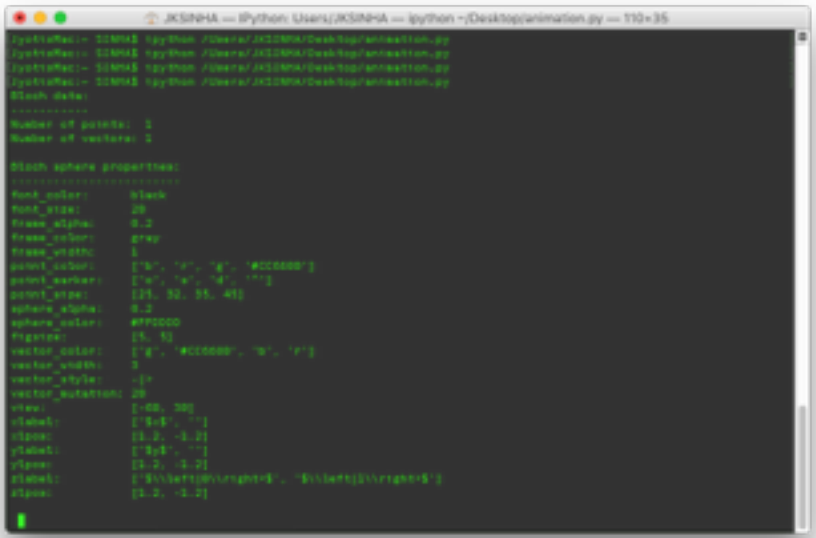
We then undo the query which means using the inverse of the CNOT. This is possible as CNOT is a unitary matrix therefore CNOT x CNOT’ is equal to the identity matrix.

We have to again apply the Hadamard transformation. This is broken into two parts, first rotating the vector on the y axis by a value of pi/2

```
xz = [np.sin(th) for th in np.linspace((startingvector[0] * (math.pi)/2),math.pi,
20)]
#set of x coordinates to move from start to finish in 20 steps

yz = np.zeros(20)
#as we are rotating about y, we do not need the y coordinates to move
```

```
zz  =  [np.cos(th)  for  th  in  np.linspace((startingvector[2]  *  (math.pi)/2),
(math.pi), 20)]
# set of z coordinates to move from start to finish in 20 steps
vectorpoints = [xz, yz, zz]
#set of vectors are now stored on a single multidimensional array

b.add_points(vectorpoints)
#vectorpoints array is now fed into this function
#vectorpoints now added to class b

b.show()
```

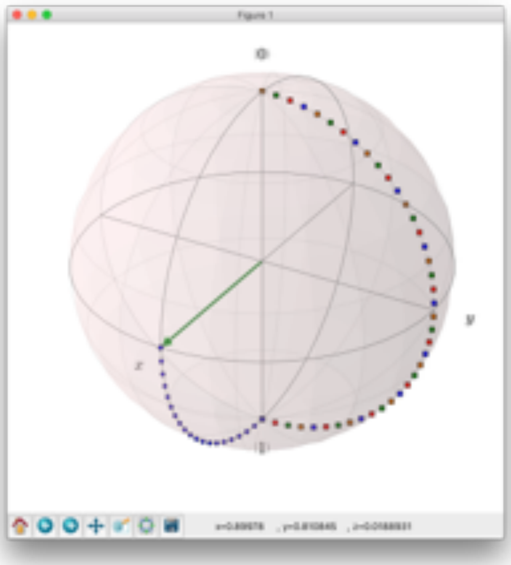


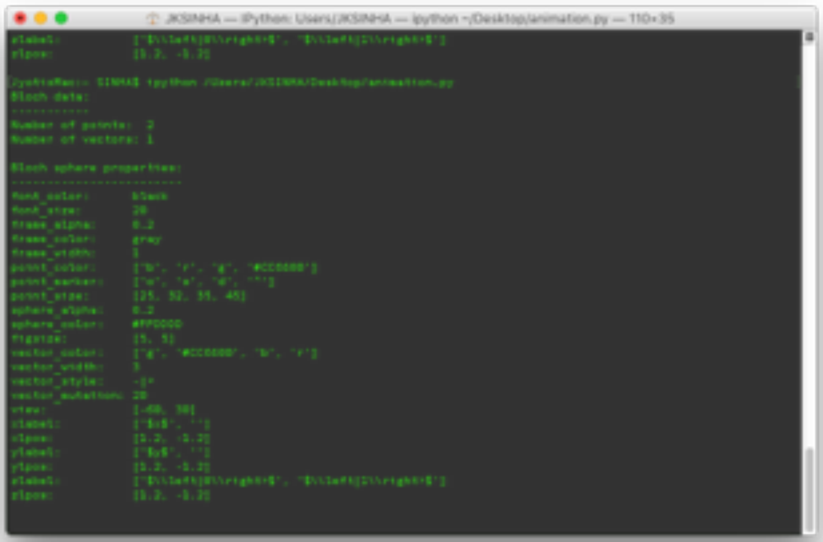

JKSINHA — IPython: Users/JKSINHA — ipython ~/Desktop/animation.py — 110×35

```
JyotisMac:~ SINHA$ ipython /Users/JKSINHA/Desktop/animation.py
JyotisMac:~ SINHA$ ipython /Users/JKSINHA/Desktop/animation.py
JyotisMac:~ SINHA$ ipython /Users/JKSINHA/Desktop/animation.py
JyotisMac:~ SINHA$ ipython /Users/JKSINHA/Desktop/animation.py
Bloch data:
-----------
Number of points:  1
Number of vectors: 1

Bloch sphere properties:
------------------------
font_color:      black
font_size:       20
frame_alpha:     0.2
frame_color:     gray
frame_width:     1
point_color:     ['b', 'r', 'g', '#CC6600']
point_marker:    ['o', 's', 'd', '^']
point_size:      [25, 32, 35, 45]
sphere_alpha:    0.2
sphere_color:    #FFDDDD
figsize:         [5, 5]
vector_color:    ['g', '#CC6600', 'b', 'r']
vector_width:    3
vector_style:    -|>
vector_mutation: 20
view:            [-60, 30]
xlabel:          ['$x$', '']
xlpos:           [1.2, -1.2]
ylabel:          ['$y$', '']
ylpos:           [1.2, -1.2]
zlabel:          ['$\\left|0\\right>$', '$\\left|1\\right>$']
zlpos:           [1.2, -1.2]
```

We now have to flip the vector direction. To reuse code we have already written, we can say a flip is the same as a rotation of pi at about the x axis. We randomly chose the x axis as the rotation point as it is simply arbitrary.

```
x2z = np.zeros(40)
#x coordinates now unchanged
y2z = [np.sin(th) for th in np.linspace((math.pi), 0, 40)]
#second set of y coordinates to move from the phase difference to inversion


z2z = [np.cos(th) for th in np.linspace((math.pi), 0, 40)]
#second set ofz coordinates to move from the phase difference to inversion

newvectorpoints = [x2z,y2z,z2z]
#set of vectors for inversion are now stored on a single multidimensional array

b.add_points(newvectorpoints, 'm')
#newvectorpoints array is now fed into this function
#newvectorpoints now added to class b
print (b)
#printing information about class b at this moment in time
b.show()
```




JKSINHA — IPython: Users/JKSINHA — ipython ~/Desktop/animation.py — 110×35

```
zlabel:          ['$\\left|0\\right>$', '$\\left|1\\right>$']
zlpos:           [1.2, -1.2]

Jyothis-Mac:~ SINHA$ ipython /Users/JKSINHA/Desktop/animation.py
Bloch data:
-----------
Number of points:  2
Number of vectors: 1

Bloch sphere properties:
------------------------
font_color:      black
font_size:       20
frame_alpha:     0.2
frame_color:     gray
frame_width:     1
point_color:     ['b', 'r', 'g', '#CC6600']
point_marker:    ['o', 's', 'd', '^']
point_size:      [25, 32, 35, 45]
sphere_alpha:    0.2
sphere_color:    #FFDDDD
figsize:         [5, 5]
vector_color:    ['g', '#CC6600', 'b', 'r']
vector_width:    3
vector_style:    -|>
vector_mutation: 20
view:            [-60, 30]
xlabel:          ['$x$', '']
xlpos:           [1.2, -1.2]
ylabel:          ['$y$', '']
ylpos:           [1.2, -1.2]
zlabel:          ['$\\left|0\\right>$', '$\\left|1\\right>$']
zlpos:           [1.2, -1.2]
```

We then apply a NOT gate which means simply flipping the unit vector values regardless of its state. The CPHASE is then applied again and we undo the NOT and Hadamard transformations in that order. The loop then starts again, causing quantum entanglement to theoretically occur. This means that the state of superposition in the qubits are conserved due to the laws of conservation in spin states.

2.2 Technical Features

2.2.1 Entanglement and probability

A pair of quantum bits get entangled due to being passed through a range of transformations and gates. They lose their entanglement when decoherence occurs (where the external environment starts interacting with the bits). We cannot really explain why entanglement occurs yet but we can say we have observed in lots of different instances. An example can be a decay occurring in a sub atomic particle. Due to the laws of conservation, if two particles are released, they must have a strong correlation between them. Also due to the laws of conservation, if we have a spin-zero particle and it splits to two particles, the spin state is shared so it becomes spin 1/2 for both. Now if we break the superposition and measure the state of one qubit, we know that the measured state of the other qubit would be opposite to what we found. As we do not know the state of any of these qubits before measured, this makes quantum computing essentially more secure than classical computing, but it also raises issues such as duplicating certain superpositions.

2.2.2 Decoherence

The idea of quantum entanglement is only really relevant if the qubits maintain a state of coherence. If the quantum computer is not fully isolated from the environment, noise is introduced. This causes quantum entanglement to break down as qubits lose their spin states. Thats why researchers like Henrik Dreyer are important as they are introducing error correction designs which solve this problem. Using his algorithm, we can see how the addition of noise affects time efficiency and what the success probability is. This graph shows how errors being introduced in gates affects success in application so clearly noise is a huge problem and unless we can fix them, we won't be able to see a working quantum computer any time soon.

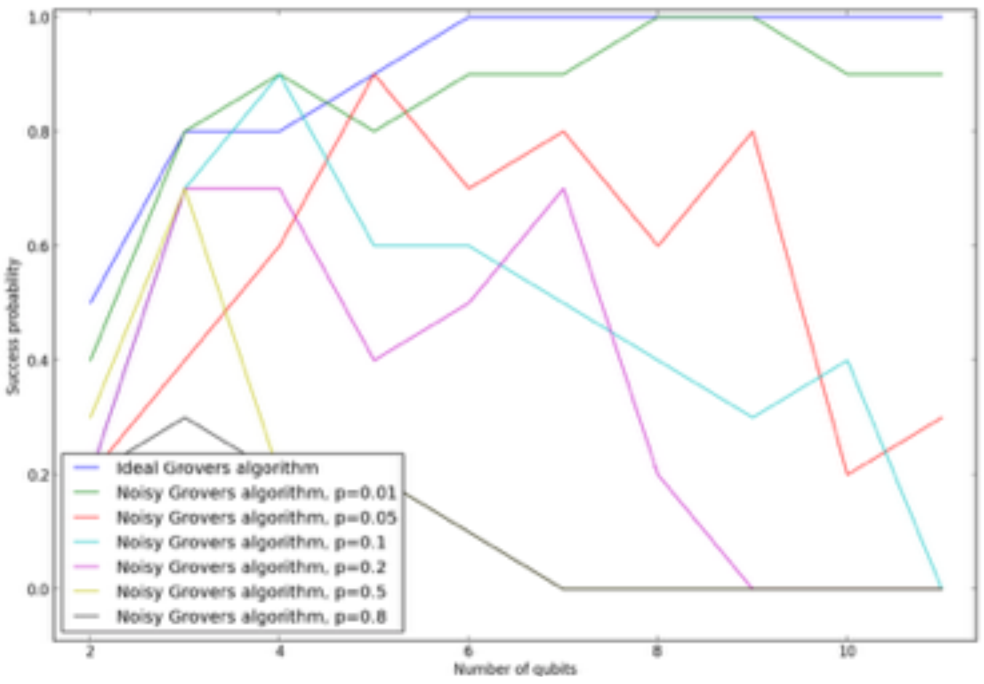


figure 12 - Dreyer's model for the success probability and a range of noisiness in system.

Part III - The Future

3.1 The Future of Quantum Computing

3.1.1 Abstract

By being able to calculate and explain how we are using quantum mechanical properties to create algorithms intrinsically different to classical algorithms, we know that quantum computing has a huge potential ahead of it. We are at a stage where algorithms are being developed and testing that can essentially bring down huge classical computing based infrastructure. This section will discuss different ways in which the computing industry will change by the further development and democratisation of quantum algorithms. These can be:

- Shor's Factorising Algorithm / RSA Encryption
- Searching Unordered Items / Travelling Salesman Problem
- Democratisation of Quantum Computers

3.1.2 Shor's Factorising Algorithm / RSA Encryption

Similar to Grover's Algorithm, Shor's algorithm focuses on using the power of superposition and quantum entanglement to implement an integer factorisation algorithm. Despite multiple classical algorithms already existing, this algorithm promises to find solutions in polynomial time. Therefore just like Grover's, this algorithm becomes more efficient the greater the number. This is important as we can now use a more refined version of this algorithm to factorise larger numbers. As this algorithm can now find solutions in measurable time, it will render encryption keys useless in the future as RSA encryption is based off 'unfactorisable' integers. Speculation can also show quantum computers could also be therefore used to create a more secure way to store data by using quantum based encryption algorithms. The algorithm was first published in 1994 by IBM scientist and by 2001, it was able to factorise basic integers, the first number being 15. Like Grover's algorithm, this algorithm is being tested on a range of different physical implementations, first using NMR based computers and further developing technologies using photonic qubits and manmade diamond structures. As of now, the largest number tested is still 15.

To mirror progress made in Shor's algorithm implementation, a new method called Adiabatic Quantum Computing was developed. The benefits of this style is largely due to remove the issue of quantum noise. Still, with billions of dollars invested by Google, ETH Zurich and Lockheed Martin, we are yet to see a real advantage to quantum computers in that sense due to the early stages of these algorithms. The largest number ADC was able to factorise is 56153 since November 2014.

The issue with these mathematical based algorithms is that economically, there are large compromises and maintenance costs that plague the future of quantum computation. The earliest example of an industry standard quantum computer is currently D-Wave Two. To ensure the qubits behave like the algorithms assume them to and to minimise quantum noise, these computers can only really operate at 0.02 Kelvin.

3.1.4 Grovers Search Algorithm (unordered) / Travelling Salesman Problem

The travelling salesman problem is a simple concept but it is difficult to implement, this is due to the nature of the problem ranking highly in the NP-Hardness scale. As it is an NP-hard problem, the time to find the solution increases exponentially as the search number or factorable number increases. This is why quantum computers are considered to be able to solve such a problem that has been plaguing mathematicians since we implemented binary based algorithms. Perhaps a refined version of this algorithm will be implemented on large, non indexed databases and also allow indexing of unordered lists and data. This is why companies like Google are investing huge amounts of money to develop this, it is not a definite, but we could say they are making an investment to produce a model that searches through the database a lot faster, at a more efficient implementation. Sensationalist news articles like 'Google’s new quantum computer is '100 million times faster than your PC' by the Telegraph add a lot of confusion to the reader. Statements such as '100 million times faster' claimed by technology journalists spread a huge amount of misinformation, this information is then relayed to journalistic giants such as The Telegraph which claim quantum computers to currently being better than they truly are at this stage. The truth is, the growth and progress of the algorithm is incremental and slow. Just on the 31st of March, which is less than a week since writing this, researchers at University of Maryland have developed a 3-qubit implementation. This means the algorithm can search through a list of 8 items. More upsettingly, their gates only have a 80.1% success probability. The process of refining Grover's algorithm, being able to truly simulate quantum noise and design error-correction to solve it are issues that we have often overlooked. To the average user, the progress of quantum computers will have no effect on their lives as it is highly impossible.

3.1.5 Democratisation of Quantum Computers

To the average user, the progress of quantum computers will have no effect on their lives as the power of quantum computers cannot make classical algorithms run faster. This means for simple tasks, such as creating documents or watching videos, classical computers will still be in use. On the other hand, the eventual democratisation of quantum computing will let average users be able to write their own algorithms or use existing algorithms to find solutions to their problems. We can back this claim but looking at IBM's press release. The IBM Q consists of 5 qubits which means that the computer is quite limited to only 32 data points. This can be time shared with users on IBM's Cloud Platform and a further SDK will be provided with 20 qubits available. Due to the rule of exponents, the simulation will theoretically be able to handle 1048576 values. They are already working on developing 50 qubit systems for the future.

In terms of asking whether quantum computer will replace classical algorithms. I think we have a clear answer. Quantum computers only benefit problems that are classed as NP-Hard or NP-Complete. We have not designed any efficient classical based algorithms for these problems as the only solution we have is to brute force trial and error these problems. That’s why Turing Machines like our phones, computers and calculators will never cease to exist as the infrastructure we currently have work far better with classical computers. Quantum computers will not provide any improvements to media consumption but they will be able to solve most typical computers cannot. Perhaps in the future, we could develop systems that allows them to work in conformity but at the time of writing, we are still trying to get numbers to divide and multiply successfully.

Appendices

Bibliography

BENOIST, E. (2006). *Effet Tunnel*. [GIF].(figure 3) Available at: https://commons.wikimedia.org/wiki/File:EffetTunnel.gif

Big Think, (2011). *How to Program a Quantum Computer, Michio Kaku*. [video] Available at: https://www.youtube.com/watch?v=rUWfod_8JsM [Accessed 2 Dec. 2016].

Chuang, I. and Nielsen, M. (2010). Quantum computation and quantum information: 10th anniversary edition. 10th ed. Cambridge: Cambridge University Press.

Corporation, I., Moore, G. and Moore, G. (2005). *Excerpts from A Conversation with Gordon Moore*.
Dreyer, H. (2014). Simulating Grover's algorithm in a noisy environment using Qutip. Cambridge Quantum Computing Limited.

Dwyer, Tyler. http://ece.ubc.ca/~tdwyer/ [accessed 20 February 2017]

Figgatt, C. (31 March 2017) Complete 3-Qubit Grover Search on a Programmable Quantum Computer. University of Maryland. Available at: https://arxiv.org/pdf/1703.10535.pdf

Glosser.ca. (2012). [SVG file]. (figure 1) Available at: https://commons.wikimedia.org/wiki/File:Bloch_Sphere.svg

Hua, M., Tao, M. and Deng, F. (2017). Quantum state transfer and controlled-phase gate on one-dimensional superconducting resonators assisted by a quantum bus. [online] Nature. Available at: http://www.nature.com/articles/srep22037 [Accessed 5 Nov. 2016].

IBM. IBM Building First Universal Quantum Computers for Business and Science. Available at: https://www-03.ibm.com/press/us/en/pressrelease/51740.wss [accessed 18 March 2017]

J. R. Johansson, P. D. Nation, and F. Nori: "QuTiP 2: A Python framework for the dynamics of open quantum systems.", Comp. Phys. Comm. **184**, 1234 (2013) [DOI: 10.1016/j.cpc.2012.11.019].

J. R. Johansson, P. D. Nation, and F. Nori: "QuTiP: An open-source Python framework for the dynamics of open quantum systems.", Comp. Phys. Comm. **183**, 1760–1772 (2012) [DOI: 10.1016/j.cpc.2012.02.021].

Jozsa, R. (2014). QUANTUM COMPUTATION Lecture notes (for lectures 1 to 10). Undergraduate. University of Cambridge.

Molloy, Mark (09 December 2015). http://www.telegraph.co.uk/technology/news/12042781/Google-D-Wave-quantum-computer-is-100-million-times-faster-than-your-PC.html[Accessed 20 February. 2016]

Moore, G. (1965). Cramming more components onto integrated circuits. *Electronics*, [online] 38(8). Available at: https://arstechnica.co.uk/wp-content/uploads/sites/3/2016/03/gordon_moore_1965_article.pdf [Accessed 25 Jan. 2017].

Moran, C. (2016). Quintuple: a Python 5-qubit quantum computer simulator to facilitate cloud quantum computing. California Institute of Technology.

Mutiara, A., Ningtyas, D., Mutiara, A. and Ningtyas, D. (2014). *SIMULATING GROVER'S QUANTUM SEARCH IN A CLASSICAL COMPUTER*.

Omer, B. (2002). Classical Concepts in Quantum Programming. University Vienna, Austria.

Omer, B. (2002). *Quantum Computing Language*. Austria: University Vienna, Austria.

Ömer, B. (n.d.). *Grover's Database Search*. [online] Tph.tuwien.ac.at. Available at: http://tph.tuwien.ac.at/~oemer/doc/quprog/node17.html [Accessed 5 Nov. 2016].

Preskill, J. (2017). *Physics 219 Course Information*. [online] Theory.caltech.edu. Available at: http://www.theory.caltech.edu/people/preskill/ph229/#lecture [Accessed 11 Dec. 2016].

Rioux, F. (n.d.). *Elements of Dirac notation*. [online] Available at: http://www.users.csbsju.edu/~frioux/dirac/dirac.pdf [Accessed 25 Oct. 2016].

Simmons, M. and Bachor, H. (2016). Scientists are exploiting the laws of quantum mechanics to create computers with an exponential increase in computing power.. [online] Nova. Available at: http://www.nova.org.au/technology-future/quantum-computing [Accessed 9 Oct. 2016].

Veritasium, (2013). *How to Make a Quantum Bit*. [video] Available at: https://www.youtube.com/watch?v=zNzzGgr2mhk [Accessed 20 Dec. 2016].

Interview

Interviewer : Aarambh Sinha (AS)
Interviewee : Henrik Dreyer (HD)

AS : My project essentially analyses the effectiveness of the algorithm by looking at multiple implementations. I have seen implementations in QCL and I am also looking at Python's Sympy library I have not seen any implementations based on QuTiP. Have you also seen these? If so, how would you comment on their effectiveness?
HD : Qutip is basically just a set of functions on top of Pythons numpy and scipy libraries that allows you to define new data types (like qubits). I do not think it is the fastest/easiest quantum simulation library out there (that award would probably go to Microsoft's Liquid).

AS : Why did you choose to use QuTiP specifically?
HD : we were pursuing some quantum error correction in CQCL at the time and we just needed something that is nice to simulate noise with.

AS : How did you simulate quantum noise?
HD : You need to assume a specific noise model for your simulation.

AS : How did you control variables such as decoherence?
HD : I think I used to have each gate be an application of a unitary matrix onto a vector (the quantum state), but with some probability there would be an extra bit-flip (Pauli X matrix) on one of the qubits. That is a super simplistic noise model and not very realistic.

AS : How did you try to increase the complexity of the noise creating algorithms?
HD : More realistic noise models include the depolarising channel or amplitude damping channel in particular (these are usually applied to density matrices, rather than pure states).

AS : What were your inital set of results like and were they time effective?
HD : The initial results were not very surprising (i.e. the more frequent the noise, the worse the success probability).

AS : I think I have gained enough conceptual knowledge in this area for the timebeing. I just wanted to say, thanks once again for the great work you are doing for this field.
HD : Forgive me for saying many words that you may not be familiar with, I just think these would be interesting questions to look at.
Maybe things become clearer if I can find the paper again (Simulating Grover's algorithm in a noisy environment using Qutip).

Source Evaluation

Source	Type	Comments
Quantum Computation And Quantum Information	Book	This served as a clear starting point for this project. The majority of my knowledge in this field came from this book as it explains theories in a simplistic way. Where it fell short was failing to discuss mathematical proofs and explain algebraic concepts in detail.
Henrik Dreyer	Interview/Thesis	Speaking to Henrik Dreyer allowed me to put my theoretical understanding into a visual format. His explanations of simulating noise gave a unique insight into the problems with decoherence that is often forgotten. His python based model allowed me to understand the algorithm in a clearer sense. I did however notice a few errors and contradicting information but I was able to see beyond it and correct it.
Tph.tuwien.ac.at.	Website	This became my most useful resource as I was able to use their Quantum Computing Language to gather data on time efficiency and see their source code for the algorithm itself. The website further explained how the quantum gates effect the qubit states. This kind of information is simply not available with any other resource.
Quantum Computers Explained - Limits of Human Technology	Video	A somewhat colloquial form of teaching gave me ideas of how to ensure that my thesis remains understandable to a non-specialised audience. The visualisations allowed me to understand how and why researchers do certain processes. These are not explained in books and resources. The algorithm for searching seems overgeneralised and so it did not contribute towards my understanding of the subject.
QuTiP	Software	As my programming for visualisation was based of QuTiP's architecture, this acts as one of the most important aspects of this thesis. The learning curve was not too steep as it follows syntax of Python. There were a few quirks with the package as I found it difficult to render states in real time. This is a major flaw as I was not able to program in real time.

Appendices

Writing Plan

Introduction and Brief Explanation of Quantum Mechanics/Computing

- 2000 words
- Introduce the history and theory behind quantum mechanics
- Explain and show how we can derive equations such as Schrodinger’s
- Discuss ideas of quantum tunnelling, entanglement and superposition

Conducting Efficiency Tests and Designing Algorithm

- 2000 words
- Show multiple implementations on different platforms
- Give an abstract explanation to the different components of the algorithm
- Do calculations and manipulate qubits on Python

Analysis and Effective Future of Quantum Algorithms

- 1000 words
- By using the data collected from the implementations, I will study the effective run times and loops to analyse whether such algorithms will replace classical algorithms
- Study current quantum computers and make an educated guess as to what the future will look like

Gantt Chart

	9.0	9.5	10.0	10.5	11.0	11.5	12.0	12.5	1.0	1.5	2.0	2.5	3.0	3.5	4
Foundations 1	🟦	🟦													
Foundations 2		🟦	🟦	🟦											
Algorithm 1				🟦	🟦	🟦									
Algorithm 2						🟦	🟦	🟦							
Algorithm 3								🟦	🟦	🟦	🟦				
Analysis 1											🟦	🟦			
Analysis 2											🟦	🟦	🟦		
Presentation														🟦	🟦
Documentation	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Review		🟦		🟦		🟦		🟦			🟦			🟦	🟦

key : 9.5 represents (.5) halfway through (9) September