EXPLORATIONS IN QUANTUM COMPUTING FOR FINANCIAL APPLICATIONS

by

Jesse Gare
B. Sc., McMaster University 2004

THESIS PROJECT
SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ARTS

In the
Faculty
of
Business Administration
Financial Risk Management Program

SIMON FRASER UNIVERSITY

Summer 2008

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENCE

Title of Project:

Explorations in Quantum Computing for Financial Applications

Author: ______________________________
Jesse Gare

____
(Date Signed)

APPROVAL

Name:	**Jesse Gare**
Degree:	**Master of Arts**
Title of Thesis:	**Explorations in Quantum Computing for Financial Applications**

Supervisory Committee:

Chair:

____

Dr. Geoffrey Poitras
Supervisor
Professor of Business Administration

____

Dr. John Heaney
Second Reader
Associate Professor of Finance

Date Defended/Approved: ______________________________

ABSTRACT

Quantum computers have the potential to increase the solution speed for many computational problems. This paper is a first step into possible applications for quantum computing in the context of computational finance. The fundamental ideas of quantum computing are introduced, followed by an exposition of the algorithms of Deutsch and Grover. Improved mean and median estimation are shown as results of Grover’s generalized framework. The algorithm for mean estimation is refined to an improved Monte Carlo algorithm. Quantum random number generation is also described.

DEDICATION

To Sasha, for being the coolest girl in the world!

ACKNOWLEDGEMENTS

I want to thank Dr. Poitras for his patience.

I want to thank all of the excellent instructors in the Financial Risk Management program who really came through and pioneered a top notch first year for the program. Special thanks to Drs. Pavlov, Jones, Theunissen, and Goddard for excellence in teaching and for all the extra support they provided.

I want to thank Dr. William Macready and his colleagues at D-Wave Systems for meeting with me to discuss the potential applications of their work. I wish them luck with their vision of commercializing a quantum computer. It truly is a noble goal.

I want to thank Drs. John Berlinsky and Brian King at McMaster University for introducing me to the wonders of quantum mechanics, and quantum computing in particular.

I want to give a huge thanks to the lovely and talented Sasha Wilson, for her motivation and all-around awesomeness.

Last, but not least, I want to thank to my parents: Robert and Sharon Gare. None of this would have been possible without their constant support and encouragement.

TABLE OF CONTENTS

INTRODUCTION

Motivation

This paper is an introduction to the field of quantum computing for financial practitioners, with an explanation of the basics of quantum computing and a description of modern algorithms that can be used to solve computational problems in finance. We start with an introduction to the field of quantum computing. The introduction follows the approach and notation used in the standard text on quantum computing (Nielsen & Chuang, 2000).

In 1994, Peter Shor, a researcher at Bell Labs, published an algorithm for factoring large composite numbers (Shor, 1994). This algorithm was exponentially faster than any known algorithm at the time. It was also unique since it would only work on a computer that could use fundamental properties of quantum mechanics. This breakthrough initiated a flood of new interest into the field of *quantum computing*, and during the following decade, many advances were made. In 1996, Lov K. Grover published a quantum algorithm for searching an unsorted database (Grover, 1996), and a few years later, he generalized his method to create a framework for the invention of new algorithms (Grover, 1998). This generalized framework allowed the creation of an algorithm for finding the mean of a statistical distribution (Grover, 1998), which can be modified for use in Monte Carlo simulations (Abrams & Williams 1999). This improved algorithm can be applied in finance to get a quadratic speed-up for any Monte Carlo simulation.

Another interesting application of quantum computing in finance is to allow true random number generation (Zak & Williams, 1999), which can be used as the first step in a Monte Carlo simulation (Glasserman, 2003), or any other situation where market factors are modelled as random variables. This application is considered since randomness is the basic mathematical concept used to model financial risk (Powers, 2008).

What is Quantum Computing?

Quantum computing is a method for performing computational tasks by the use of quantum mechanical systems. It is fundamentally different from the familiar paradigm of classical computing because quantum mechanical systems have properties that cannot be efficiently simulated using conventional classical information processing methods. Some of the most useful quantum properties are superposition, interference, entanglement, non-determinism, and non-clonability. We can exploit each of these properties to realize fundamentally new computational advantages.

In a conventional computer, the basic unit of information processing is a binary digit (usually referred to as a “bit”) which can be in one of two states: zero or one. The bit is used to represent one piece of information, and a collection of bits (known as a register) can be used to represent more complicated information. Physically, inside the computer, these bits are realized as transistors (or switches) which can also take one of two states: on or off. The physical registers in the computer are just arrays of these switches, so a register taken as a whole is considered to be in only one state at a time. For example, in a register of 3 bits there are eight possible states, where the zeros and ones represent the states of the individual bits: 000, 001, 010, 011, 100, 101, 110, and 111. These eight register states could be used to represent the numbers from one to eight, or any other piece of information that is distinguished by three true/false decisions.

Like the conventional bit, the quantum bit (or "qubit") can also take one of two states: $|0\rangle$ or $|1\rangle$. Here, the "$|\ \rangle$" surrounding the zero and one is used to show that we are talking about a quantum state, instead of a regular classical state. The "$|\ \rangle$" enclosure is known as Dirac notation, and it has become the standard for denoting quantum states, and we will use it to denote qubit states. We can think of a variable in this notation as a column vector, in which case the "$\langle\ |$" enclosure is the transpose of the column vector (the equivalent row vector) for our purposes. See Appendix A for a more thorough description of this notation.

There are similarities between bits and qubits, but there are also fundamental differences. Unlike the conventional bit, the qubit can exist in a kind of mixture of both states $|0\rangle$ and $|1\rangle$ at the same time. This is possible to realize in a quantum system due to the quantum phenomenon known as superposition, in which a component of a quantum system can be prepared such that its configuration is fundamentally uncertain. By fundamentally uncertain, we mean there is no possible way to measure how the component is configured without altering its configuration. Mathematically, the possible states of the qubit are linear combinations of the states $|0\rangle$ and $|1\rangle$:

$$|\Psi\rangle = a|0\rangle + b|1\rangle,$$

where $|\Psi\rangle$ is the state of the qubit, and a and b are complex numbers. In other words, the state of the qubit is a vector in a two-dimensional complex vector space. Since the state of the qubit is fundamentally uncertain when it is in a superposition like this, we cannot examine a qubit to determine which state it occupies. Due to the peculiarities of quantum mechanics, if we measure the qubit, we get either the result $|0\rangle$, with probability $|a|^2$, or the result $|1\rangle$, with probability $|b|^2$.

This property of superposition can be exploited for computational benefits. Consider a register of qubits, which is just an array of qubits combined together. Since the qubit can be arranged into a state that is an equal superposition of both of its basis states, namely the state:

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

We can consider this state as being both $|0\rangle$ and $|1\rangle$ simultaneously, therefore representing 2 pieces of information simultaneously. Now, consider a register of three such qubits, each in an equal superposition of its 2 states. It would represent all of the 8 states: $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, $|111\rangle$ simultaneously. In general, an n-qubit register could store 2^n states simultaneously, whereas the conventional n-bit register could only store one state at a time. Since a quantum register can be placed into a superposition of all of its classical states, it makes a quantum computer into a massively parallel computer. A quantum computer can work on *all* possible classical states simultaneously in the time it would take to work on just one of the states in a classical computer. Therefore, there might be a way in which we can manipulate this quantum register so it would end up in a quantum state that, when measured, would reveal an answer to a desired computation. This is, in fact, what quantum computing is all about.

Some Quantum Computing Basics

In order to understand how we can manipulate quantum registers to output solutions to computational problems, we have to learn about how quantum systems evolve. The most succinct way to learn this is to recall the four fundamental postulates of quantum mechanics. Note that the postulates described in the following sections will not be stated in their most general form since we are only concerned with finite dimensional problems. This simpler version of quantum mechanics will be good enough for our purposes, since quantum computers have a finite number of states, and we therefore only need to consider finite dimensional spaces. This also greatly simplifies the mathematical formalisms needed. Most of the descriptions and notation used in this section are borrowed from the standard text on quantum computing (Nielsen & Chuang, 2000).

The First Postulate of Quantum Computing Mechanics

Postulate #1 explains that we can associate a mathematical concept (namely a Hilbert space) with any isolated physical system. We call this the state space, because it contains all of the possible states for our physical system. The system's particular configuration is then represented by a state vector, which is a member of the state space. We have already seen an example of a state space associated with the qubit, which is in fact the simplest quantum mechanical system. The qubit has a 2-dimensional state space. Since the basis states $|0\rangle$ and $|1\rangle$ form an orthonormal basis for the state space, any arbitrary state vector in the state space can be written as $|\Psi\rangle = a|0\rangle + b|1\rangle$. There is one more detail associated with postulate #1, which is the condition that state vectors must be unit vectors in the system's state space. For the qubit state, this just means that $|a|^2 + |b|^2 = 1$. Previously, we saw that $|a|^2$ and $|b|^2$ are just the probabilities of measuring states $|0\rangle$ or $|1\rangle$ respectively. Therefore, the condition $|a|^2 + |b|^2 = 1$ is just the familiar condition that probabilities must add to one.

The Second Postulate of Quantum Computing Mechanics

Postulate #2 states that the evolution of a closed quantum system is described by a *unitary transformation*. More precisely, the state $|\Psi_1\rangle$ at time t_1 is related to the state $|\Psi_2\rangle$ at time t_2 by a unitary operator U, which depends only on times t_1 and t_2. That is:

$$|\Psi_2\rangle = U(t_1, t_2)|\Psi_1\rangle$$

By a unitary operator, we mean a matrix U, which has the property that $UU^* = I$, where the $*$ superscript denotes the conjugate transpose (i.e. transpose the matrix, and take the complex conjugate of all the elements), and I in this case refers to the identity matrix which has the same dimensions as the matrix U. This postulate is very important for quantum computing, because it implies that qubits (or registers of qubits) can only be transformed by unitary operators. Therefore, in a quantum computer, the analogues of classical logic gates are unitary operators. Thus, we can visualize quantum circuits using diagrams similar to the logic diagrams of classical computing. We will see such diagrams later in this section.

One interesting and very useful unitary operator is the Hadamard operator, which is usually denoted by H, where:

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

If we identify the state $|0\rangle$ with the column vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and similarly $|1\rangle$ with $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, then:

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}},$$

and $$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

Thus, the Hadamard operator has the remarkable property that it transforms the pure basis states to equal superposition states. This is very useful in many quantum algorithms, because often we would like to work with equal superpositions of the states, but usually only have the basis states to start with.

The Third Postulate of Quantum Computing Mechanics

Postulate #3 explains an alternate way in which quantum systems can evolve. If the quantum system is measured (i.e. its state is observed), then it immediately undergoes a discontinuous and unpredictable jump into one of its basis states. That is, the superposition is destroyed. In the case of the single qubit, which is represented by the state vector $|\Psi\rangle = a|0\rangle + b|1\rangle$, a measurement of this state gives the result $|0\rangle$, with probability $|a|^2$, or the result $|1\rangle$, with probability $|b|^2$.

The Fourth Postulate of Quantum Computing Mechanics

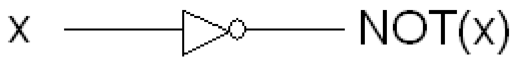
To understand how the entire quantum register evolves, we need to consider composite quantum systems. That is, not just single qubits, but collections of qubits that make up a quantum register. Postulate #4 deals with composite systems. Postulate #4 says that the state space of a composite physical system is the *tensor product* of the state spaces of the component physical systems. The tensor product is a way of putting vector spaces together to form larger vector spaces. The tensor product between two state vectors $|\Psi\rangle$ and $|\Phi\rangle$ is written as $|\Psi\rangle \otimes |\Phi\rangle$, or sometimes as $|\Psi\rangle|\Phi\rangle$,$|\Psi\Phi\rangle$ or $|\Psi,\Phi\rangle$. To get a good idea of how the tensor product works, it helps to see how the tensor product of two matrices A and B is defined. Supposing A is an m-by-n matrix, and B is a p-by-q matrix, then the matrix representation of the tensor product is:

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1n}B \\ A_{21}B & A_{22}B & \cdots & A_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B & A_{m2}B & \ldots & A_{mn}B \end{bmatrix},$$

and therefore, $A \otimes B$ is an mp-by-nq matrix. The tensor product can also be generalized to tensor exponentiation, for example: $|\Psi\rangle^{\otimes 3} = |\Psi\rangle \otimes |\Psi\rangle \otimes |\Psi\rangle$. Now, suppose we had a quantum register of n qubits, and we wanted to construct a unitary operator which, when applied to our register would give us an equal superposition of all possible n-qubit states. Since the Hadamard operator (H) gives us an equal superposition for one qubit, the natural operator which gives us the equal superposition of all n qubits is $H^{\otimes n}$.

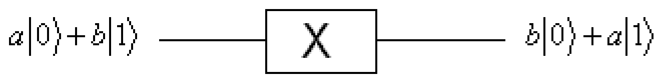
Quantum Circuits

As mentioned above, the second postulate gives us a method for creating circuit diagrams, much like the logic diagrams of classical computing. For example, the logic diagram for the simplest transformation in classical computing (the "not" transformation) looks like this:



where x denotes a binary variable (0 or 1), and the symbol in the middle is known as the *not gate*, which transforms the binary value to its opposite value. This is a simple schematic diagram that helps us understand what is happening with the logic variables. If x started as zero, it would end up as one, and if x started as one, it would end up as zero.

There is a similar "not" transformation for qubits, but it is not quite as simple since the qubit can take on a whole continuum of states instead of just two. The quantum "not" gate is usually denoted by X, because its matrix representation is the same as the Pauli-X spin matrix, which is well known in quantum mechanics. The effect this X gate has on a general qubit is as follows:



that is, it effectively swaps the probability amplitudes (the a and the b).

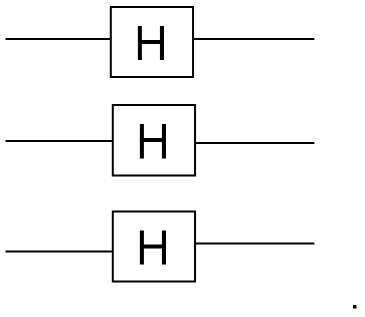
These "circuit" diagrams are one way to represent the evolution of qubits in a quantum algorithm. Another way is to use vector and matrix representations, so that the evolution of the algorithm is found through matrix multiplication (where the matrices are unitary as stated in postulate #2). If we identify the state $|0\rangle$ with the column vector $\begin{bmatrix}1\\0\end{bmatrix}$, and similarly $|1\rangle$ with $\begin{bmatrix}0\\1\end{bmatrix}$, then the general qubit will be given by $\begin{bmatrix}a\\b\end{bmatrix}$, and any transformation on a single qubit will be a 2-by-2 matrix. For example, the X gate given above would be represented by $X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$. The quantum circuit as shown above would be expressed by the matrix multiplication:

$$\begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}.$$

The problem with this method of representation becomes apparent when we deal with systems with many qubits. Due to postulate #4, and the use of the tensor product, the unitary matrices used in the computations will grow exponentially with respect to the number of qubits, such that it will not be feasible to write out all of the matrices involved. For example, the Hadamard operator for a single qubit is given by $H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ in our standard representation. However, the Hadamard operator on three qubits is: $H^{\otimes 3} = H \otimes H \otimes H$, which has the matrix representation:

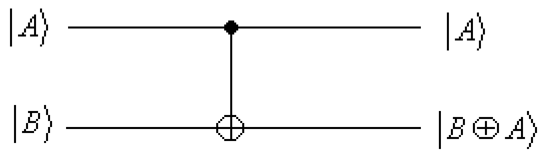
$$H^{\otimes 3} = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix},$$

whereas the quantum circuit representation is given by:



This is why the quantum circuit method is generally a better method for visualizing quantum algorithms. We will use it often in later sections.

Since we are considering multi-qubit quantum gates, we should mention another very important one, known as the controlled-NOT gate. Its circuit representation is as follows:



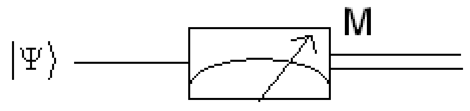
where the solid dot indicates that the top qubit "controls" the bottom qubit, and the $\oplus$ symbol is the same as the XOR (exclusive OR) operation from classical logic, which is equivalent to modulo-2 addition (that is, find the remainder when $A+B$ is divided by 2). The action of this gate can be described in the following way: if the top (control) qubit is 0 then the bottom (target) qubit is left alone, however, if the top qubit is 1, then the target qubit is flipped. That is:

$|00\rangle \rightarrow |00\rangle$; $|01\rangle \rightarrow |01\rangle$; $|10\rangle \rightarrow |11\rangle$; $|11\rangle \rightarrow |10\rangle$.

In terms of a matrix representation if we let $|00\rangle$ be represented by $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $|01\rangle$ be represented by $\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $|10\rangle$ be represented by $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, and $|11\rangle$ be represented by $\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, then the controlled-NOT operator is: $U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$. Or, in terms of a simple equation in Dirac notation: $|A, B \oplus A\rangle = U_{CN}|A, B\rangle$.

The controlled-NOT gate is very important, because it has been proven that this gate, along with 1-qubit rotation gates (of which X is one example), is sufficient for *universal* quantum computation (DiVincenzo, 1995). This means that any algorithm created by the use of quantum computing can be broken down into steps involving the controlled-NOT gate and one-qubit rotations. In other words, since we can represent the evolution of a quantum system by a unitary operator, then all unitary operators can be broken down into products of smaller unitary operators such as 1-qubit rotations, and the controlled-NOT gate.

There is one more tool we need in our toolbox of quantum circuits, which is a symbol for quantum measurement (as mentioned in postulate #3). For a measurement of a quantum state, we will use the following symbol:



.

This symbol will usually come at the end of the algorithm, since measurement destroys any superposition, so we hope to have finished all the necessary steps that require superposition for computational advantages. Thus, the usual order of a quantum algorithm, is as we have explained it here. First, prepare the state (which may come from initializing all the qubits to $|0\rangle$, and then applying the Hadamard operator to form equal superpositions). The second step will be to evolve the states (by employing some unitary operators). The third and final step is usually to measure the states, to get a certain output. This is the general order of a quantum algorithm.

Quantum Computing in Finance

Now that we have acquired a basic understanding of quantum computing, one might wonder how to apply these methods to solve financial problems. Mathematical finance, like any other field of applied mathematics, has computational problems which can be categorized into computational classes. These problems are classified according to the resource requirements needed to solve them. Computational resources include time, space, and energy. Usually, for a certain computer system, energy requirements remain a constant, but space and time needs can vary drastically. By space, we mean the amount of computer memory needed to solve the problem. As mentioned before, a quantum computer has the potential to reduce the space requirements due to the way it can work with many numbers simultaneously through superpositions.

The time requirement is the most variable, but also the resource we can best quantify. We can characterize the time requirement by defining a function between the sizes of the input to the number of time steps it takes a given algorithm to run. The general size of this function is taken as a measure of the time resource. If the time to run grows as a function $g(n)$ of the input size (n)

then we say the algorithm runs in order $g(n)$ or $O(g(n))$. This method measures the upper bound of the asymptotic running time, and is known as asymptotic notation, or big-O notation, and it can be used to classify problems into different computational classes. We can also use the notation $\Omega(g(n))$ to denote the asymptotic lower bound for the running time of an algorithm.

One area in which quantum computing can reduce the time requirements of a computation is in the area of derivative pricing. The usual computational methods used to price derivatives are:

1) The binomial tree approach
2) The Monte Carlo approach, and
3) Solving partial differential equations (PDEs) using a finite difference scheme.

These three methods could all potentially benefit from an implementation on a quantum computer. However, the Monte Carlo approach is the most natural candidate due to its dependence on randomness. Implementing an improved Monte Carlo algorithm on a quantum computer will be the major focus of this paper.

An Example Solution to a Financial Problem

We will now look at a simple quantum algorithm to see how we can exploit the power of quantum mechanics to compute the solution to a problem in less time than would be possible using a classical computer. In the quantum computing literature the following algorithm is known as Deutsch's algorithm (Deutsch, 1985), and it is often posed as a solution to a financial problem.

Let us suppose we have discovered a method that can predict whether the market price of a given financial instrument will rise or fall tomorrow, depending on the outputs of some complicated predictor function $f(x,M)$. In this function, x is a binary variable (i.e. it can only take the values 0 or 1), the output is also a binary variable (i.e. $f(x,M)$ can only take the values 0 or 1), and M is a variable that includes relevant market data (which will be taken as a constant for any given market day, so we can suppress this variable and simply write $f(x)$). Our predictive method works as follows: if $f(0)=f(1)$ the instrument's market price will increase tomorrow, and if $f(0)\neq f(1)$ the market price will decrease. Our dilemma is this: the function $f(x)$ is so complicated that it takes the majority of a day to calculate its value on the fastest super-computer available today. Since it takes most of a day to calculate $f(0)$ and most of a day to calculate $f(1)$, by the time we have calculated both $f(0)$ and $f(1)$ (both of which are needed in order to compare their values) at least an entire day will have passed. At this point, it will be too late to execute any trades to take advantage of our prediction. However, as it turns out, if our super-computer had some quantum abilities, it could calculate our prediction in time to make a handsome profit!

The basic idea of Deutsch's algorithm is that we do not actually need to calculate both $f(0)$ and $f(1)$ in order to compare their values. We are actually only concerned with a joint property of $f(0)$ and $f(1)$ (whether they are equal or not) and quantum computers excel at answering questions about joint properties.

Let us define an operator, U, that transforms the general 2-qubit register as follows:

$$U : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$$

(recall that $\oplus$ is the exclusive-OR operator). So the result of this transformation is that the first qubit, $|x\rangle$, is unchanged, and the second qubit is unchanged unless $f(x)=1$, in which case its value is flipped. In order to value the function at both inputs simultaneously, we need to work with superpositions of the states $|0\rangle$ and $|1\rangle$. If we apply the Hadamard operator to the state $|1\rangle$, we get $H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. If we use this as the initial second qubit, and apply the U operator, we get:

$$U : |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \mapsto |x\rangle \otimes \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}.$$

Now, let us consider what happens for the two possible values of $f(x)$:

$$\text{if } f(x)=0: \ |x\rangle \otimes \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |x\rangle \otimes \frac{(-1)^0\left(|0\rangle - |1\rangle\right)}{\sqrt{2}},$$

$$\text{if } f(x)=1: \ |x\rangle \otimes \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = |x\rangle \otimes \frac{|1\rangle - |0\rangle}{\sqrt{2}} = |x\rangle \otimes \frac{(-1)^1\left(|0\rangle - |1\rangle\right)}{\sqrt{2}}.$$

Therefore, it is clear that we can write:

$$|x\rangle \otimes \frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}} = |x\rangle \otimes \frac{(-1)^{f(x)}(|0\rangle - |1\rangle)}{\sqrt{2}}.$$

This is the most important result for Deutsch's algorithm, because it allows us to summarize the effect of the operator U as a sign change on the second qubit. Now, if we also allow the first qubit to be an equal superposition of $|0\rangle$ and $|1\rangle$, we will be able to use the fact that U only needs to evaluate the function once, to give properties of both $f(0)$, and $f(1)$. If we apply the Hadamard operator to the state $|0\rangle$, we get $H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$. If we use this as the initial first qubit, and apply the U operator, we get:

$$U : \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|1\rangle - |0\rangle}{\sqrt{2}} \mapsto \frac{|0\rangle}{\sqrt{2}} \otimes \frac{(-1)^{f(0)}(|0\rangle - |1\rangle)}{\sqrt{2}} + \frac{|1\rangle}{\sqrt{2}} \otimes \frac{(-1)^{f(1)}(|0\rangle - |1\rangle)}{\sqrt{2}}$$

$$= \left(\frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right).$$

If we apply the Hadamard operator to the first qubit again, we get:

$$\left(\frac{\left((-1)^{f(0)} + (-1)^{f(1)}\right)|0\rangle + \left((-1)^{f(0)} - (-1)^{f(1)}\right)|1\rangle}{2} \right) \otimes \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

If we measure this first qubit, according to postulate #3, we get:

$|0\rangle$ with probability $\left| \frac{(-1)^{f(0)} + (-1)^{f(1)}}{2} \right|^2$,

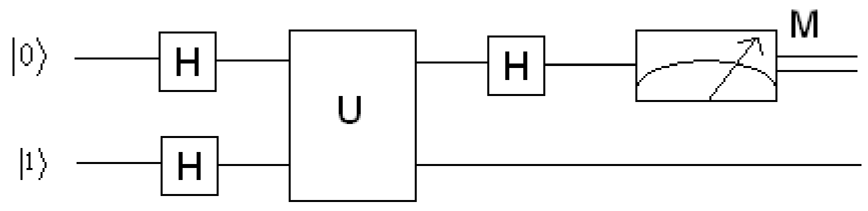
and $|1\rangle$ with probability $\left| \frac{(-1)^{f(0)} - (-1)^{f(1)}}{2} \right|^2$.

These probabilities are the results we need. Notice that if $f(0) = f(1)$ we get state $|0\rangle$ with probability 1, and if $f(0) \neq f(1)$ we get state $|1\rangle$ with probability 1. Therefore, using Deutsch's algorithm, we only need to evaluate the function $f(x)$ once, and we get a result that allows us to determine with certainty if the value of $f(x)$ is equal for the two inputs or if $f(x)$ is not equal, thus allowing us to predict the movement of our specified financial instrument. Now all we need to do is discover such a function $f(x)$ which has these properties!

Deutsch's algorithm very clearly shows the potential benefits of a quantum computer. It employs the quantum mechanical properties of superposition and interference, and it employs the four basic steps used in most quantum algorithms:

1. Initialize the quantum registers.
2. Put the registers in superpositions of states.
3. Evolve the registers using unitary operators.
4. Measure the states to get some result.

Deutsch's algorithm can be summarized in quantum circuit notation like so:



.

In the next section, we will explore some other quantum algorithms that are much more powerful than Deutsch's algorithm. We will also see how to use these algorithms to solve some real-life finance problems (as opposed to the contrived problem posed in this section).

QUANTUM ALGORITHMS

A Global Perspective

Quantum computing is still in its infancy, and all of the algorithms presented in this section are merely first steps into the world of possibilities unleashed when using quantum mechanics for computational advantages. However, even though this field is just beginning, it is useful to note that large financial institutions have much to gain by being early adopters of new computational technologies. For this reason, when quantum computing matures, it is likely that some of its first real-world applications will be in the financial arena. This section highlights some generic quantum algorithms, which, when refined and specialized, will allow computational acceleration to the solution of many financial problems.

Grover's Search Algorithm

Suppose you have a problem with the following properties:

1. The only way to solve the problem is to guess an answer and check if it's correct,
2. There are N possible answers to check, and
3. Every possible answer takes the same amount of time to check.

One example of this problem is the searching of an unsorted database. For example, if you have a telephone book with N entries in which all the names are randomly ordered, and you want to find a telephone number which corresponds to a certain name, guessing and checking is the best you can do. There is no way in which the search can be sped up. Since this search will take on average $\frac{N}{2}$ guesses with the possibility of taking up to N guesses, this method of guessing and checking is an $O(N)$ algorithm, and there is no possible way it can be sped up. However, a surprising result from quantum computing is that there is an algorithm which can find an answer to such problems in only $O\left(\sqrt{N}\right)$ steps!

In this section we will learn about this $O\left(\sqrt{N}\right)$ algorithm which is usually known as Grover's algorithm (after its creator, Lov K. Grover (Grover, 1996)). Suppose there are N elements which we want to search through. We will identify each element by an index in the range from 0 to $(N-1)$. Furthermore, we will assume that $N = 2^n$, so that the index can be encoded into n bits. We will also assume that there is only one solution. We define $f(x)$ as our search function. That is, for any index x, $f(x) = 1$ if x is the solution, otherwise $f(x) = 0$. This function is an indicator function which lets us know whether or not we have found a solution.

As in Deutch's algorithm we will define a unitary operator (U) which has the following action, which allows us to apply our search function:

$$U|x\rangle|w\rangle \rightarrow |x\rangle|w \oplus f(x)\rangle$$

Here, w is an extra work qubit, which is flipped if $f(x)=1$. This unitary operator allows us to check whether x is a solution to our search problem without disturbing the state x in the following way: after preparing the state $|x\rangle|0\rangle$, we can apply the operator U. If w is flipped to 1, then we know that x is a solution to our search problem.

Just as in Deutch's algorithm, it is useful to set the extra qubit to the equal superposition $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, and similarly applying the operator U works as follows:

$$U:|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \rightarrow (-1)^{f(x)}|x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$$

Notice the work qubit doesn't change. In fact, it doesn't change throughout the entire quantum search algorithm. Therefore we don't need to show it, and we will furthermore simply write the effect of U as:

$$U:|x\rangle \rightarrow (-1)^{f(x)}|x\rangle$$

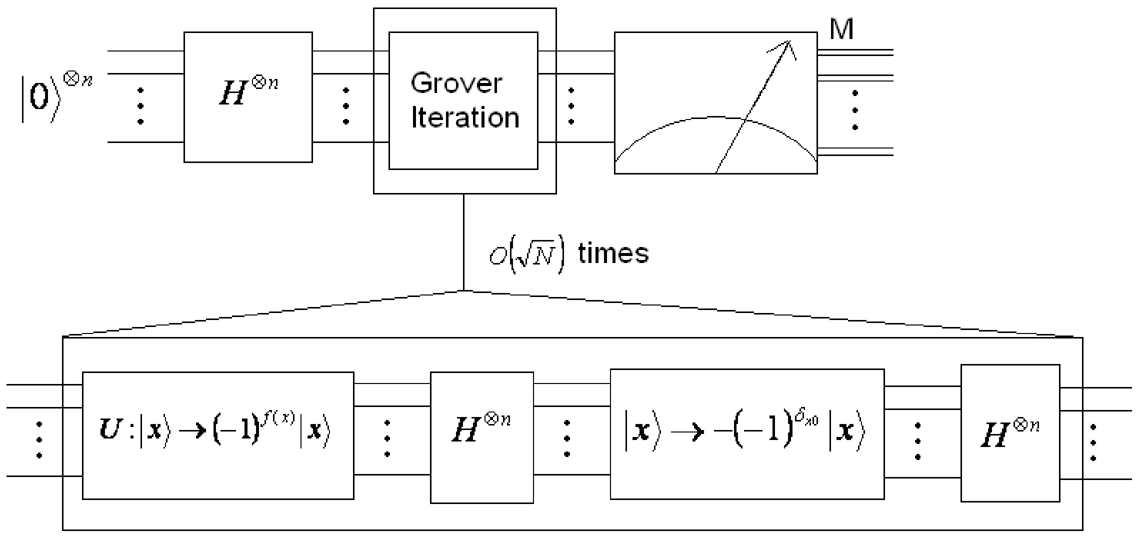
That is, U indicates solution states by flipping the sign.

The algorithm proceeds as follows:

1. Start with the index in the zero state $|0\rangle^{\otimes n}$

2. Apply the n-qubit Hadamard transform $H^{\otimes n}$ to put the index in an equal superposition of all of its states: $|\Psi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$
3. Then apply what is known as the *Grover Iteration* $O(\sqrt{N})$ times which consists of the following steps:
    a. Apply the operator U,
    b. Apply the Hadamard transform $H^{\otimes n}$,
    c. Switch the sign of every basis state except 0. This transformation can be denoted by: $|x\rangle \rightarrow -(-1)^{\delta_{x0}} |x\rangle$, for each computational basis state $|x\rangle$, where δ_{x0} is the Kronecker Delta function which is 1 if both of its indices are the same, and 0 otherwise.
    d. Apply the Hadamard transform $H^{\otimes n}$ again.
4. After $O(\sqrt{N})$ Grover iterations, the index will be equal to the solution value with high probability. We measure the index state to determine the value.

Ignoring the work qubits, the quantum circuit for Grover's algorithm is:



At first glance, it's not entirely obvious how this algorithm works. But a little analysis can clear up the confusion.

First of all, let's consider what operator must be applied to a general state $\sum_{x=0}^{N-1} a_x|x\rangle$ (where the $|x\rangle$s are the computational basis states and the a_xs are their respective amplitudes) to achieve the effect as described in step 3c. After a little thought, we can see that the operator which switches the sign of all the states except the $|0\rangle$ state is $2|0\rangle\langle 0|-I$. To prove this, we will apply this phase shift operator to the general state:

$$\left(2|0\rangle\langle 0| - I\right)\sum_{x=0}^{N-1} a_x |x\rangle = \sum_{x=0}^{N-1}\left(2a_x|0\rangle\langle 0|x\rangle - a_x|x\rangle\right)$$
$$= \sum_{x=0}^{N-1}\left(2\delta_{0x}a_x|0\rangle - a_x|x\rangle\right)$$
$$= \left(2\delta_{00}a_0|0\rangle - a_0|0\rangle\right) + \sum_{x=1}^{N-1}\left(2\delta_{0x}a_x|0\rangle - a_x|x\rangle\right)$$
$$= a_0|0\rangle - \sum_{x=1}^{N-1} a_x|x\rangle$$

This shows that only the $|0\rangle$ state has remained unchanged, all of the other states have opposite signs.

Now, if we combine this phase shift operator from step 3c with the Hadamard operators from steps 3b and 3d, we get the following operator:

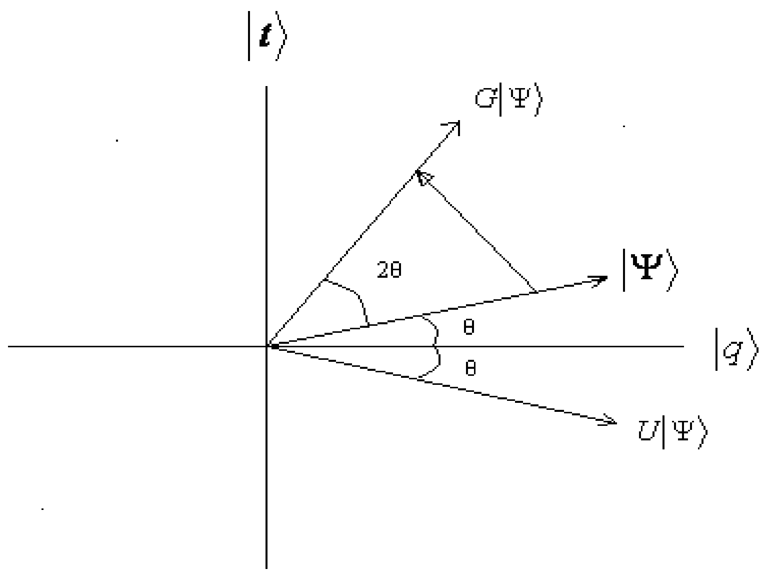
$$H^{\otimes n}\left(2|0\rangle\langle 0| - I\right)H^{\otimes n} = \left(2H^{\otimes n}|0\rangle\langle 0|H^{\otimes n} - H^{\otimes n}H^{\otimes n}\right)$$
$$= 2|\Psi\rangle\langle\Psi| - I$$

Where, as above, the state $|\Psi\rangle$ is the equal superposition state. Now, if we apply the operator U as well, we can express the entire Grover iteration as one operator: $G = \left(2|\Psi\rangle\langle\Psi| - I\right)U$.

Now that we have defined the Grover iteration, we can ask: what does it do? It can be shown (Aharonov, 1999) that the Grover iteration is a rotation in a two dimensional space defined by the solution state. If we define $|t\rangle$ as the target solution state, and $\left|q\right\rangle = \frac{1}{\sqrt{N-1}}\sum_{x\neq t}\left|x\right\rangle$ as a superposition of all of the states which are not solutions, then the Grover iteration G is a rotation in this t-q space. To see this, first realize that we can write the equal superposition state $\left|\Psi\right\rangle$ as a weighted sum of these two other states:

$$\left|\Psi\right\rangle = \sqrt{\frac{N-1}{N}}\left|q\right\rangle + \sqrt{\frac{1}{N}}\left|t\right\rangle$$

Then, the U operator performs a reflection about the state $\left|q\right\rangle$, and the remaining part of the Grover iteration, $2\left|\Psi\right\rangle\left\langle\Psi\right| - I$, performs a reflection about the state $\left|\Psi\right\rangle$. Since the product of two reflections is a rotation, we can see that the Grover iteration rotates the index state towards the solution state. This can be visualized geometrically as follows:



$\left|t\right\rangle$
$G\left|\Psi\right\rangle$
2θ
$\left|\Psi\right\rangle$
θ
θ
$\left|q\right\rangle$
$U\left|\Psi\right\rangle$

The effect of the operation G on the states $|t\rangle$ and $|q\rangle$ is:

$$G|t\rangle = \left(1 - \frac{2}{N}\right)|t\rangle - \frac{2\sqrt{N-1}}{N}|q\rangle$$
$$G|q\rangle = \left(\frac{2\sqrt{N-1}}{N}\right)|t\rangle - \left(1 - \frac{2(N-1)}{N}\right)|q\rangle$$

We can write this transformation in matrix form, where the first row and column correspond to $|q\rangle$, and the second to $|t\rangle$ as:

$$G = \begin{bmatrix} -\left(1 - \frac{2(N-1)}{N}\right) & -\frac{2\sqrt{N-1}}{N} \\ \frac{2\sqrt{N-1}}{N} & 1 - \frac{2}{N} \end{bmatrix} = \begin{bmatrix} \frac{N-2}{N} & -\frac{2\sqrt{N-1}}{N} \\ \frac{2\sqrt{N-1}}{N} & \frac{N-2}{N} \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{N-1}{N}} & -\sqrt{\frac{1}{N}} \\ \sqrt{\frac{1}{N}} & \sqrt{\frac{N-1}{N}} \end{bmatrix}^2$$

It can be shown that this matrix is a rotation operator if we define:

$$\sin(\theta) = \frac{1}{\sqrt{N}}$$
$$\cos(\theta) = \sqrt{\frac{N-1}{N}}$$

then:

$$G = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}^2 = R_{2\theta}$$

Therefore, the Grover iteration is a rotation in t-q space by an angle 2θ. Letting k be the number of iterations of G applied to the initial state $|\Psi\rangle$, the final state will be:

$$G^k|\Psi\rangle = \cos((2k+1)\theta)|q\rangle + \sin((2k+1)\theta)|t\rangle$$

Since we want our final state to have an amplitude close to 1 in the $|t\rangle$ state, the optimal number of Grover iterations to apply will be:

$$\sin((2k+1)\theta) \approx 1 \Rightarrow k \approx \frac{\pi}{4\theta} - \frac{1}{2} = \frac{\pi}{4\sin^{-1}\left(\frac{1}{\sqrt{N}}\right)} - \frac{1}{2} \approx \frac{\pi\sqrt{N}}{4} - \frac{1}{2}$$

Since k must be an integer, we have to round this quantity. A good choice is to round down, in which case we end up with:

$$k = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$$

This shows that Grover's algorithm converges on the solution state $|t\rangle$ in $O(\sqrt{N})$ iterations.

Grover's Generalized Framework

In 1998, Grover published a generalized framework for algorithms in which this $O(\sqrt{N})$ search algorithm is just a special case (Grover, 1998). Two more algorithms came from this generalized framework, which have promise for applications in finance. These algorithms find approximations for the median and mean of a statistical distribution. It takes $\Omega(\varepsilon^{-2})$ to estimate these statistics to a precision ε on a classical computer, but using Grover's generalized framework the calculation only takes $O(\varepsilon^{-1})$.

Suppose we have a function $f(x)$ on the indices $x = 0, 1, \dots N-1$, which is 0 for all inputs except for a target state, t, and we want to find t (many problems can be transformed into solving a function of this type). Assume that we have a unitary transformation, U, and we start with the system in the state $|s\rangle$. If we apply U to $|s\rangle$, the amplitude of reaching $|t\rangle$ is U_{ts} (the element at the t^{th} row and s^{th} column of the matrix representation of U), and the probability of getting t when the system is measured is: $|U_{ts}|^2$. This probability means that it would take $\Omega\left(\frac{1}{|U_{ts}|^2}\right)$ guesses until we found the solution index t. However, Grover's generalized algorithm reaches t in only $O\left(\frac{1}{|U_{ts}|}\right)$ steps.

This generalized algorithm uses a method called *amplitude amplification* to increase the probability of the solution state, while decreasing the probabilities of all the non-solution states. To do this, we need to define the operator I_x which is effectively a diagonal matrix with all terms equal to 1, except the xx term which is -1. This is used to identify certain states for amplification. We also need to define I_t, which isn't as simple since we don't know the value of t prior to running the algorithm. However, this can be set up using the our indicator function $f(x)$, and the XOR operator as shown in the previous sections. The algorithm itself is extremely simple, and can be summarized as applying $O\left(\frac{1}{|U_{ts}|}\right)$ repetitions of the operation sequence:

$$G \equiv -I_s U^{-1} I_t U \ ,$$

to reach the state $U^{-1}|t\rangle$. Then we only need to apply U once more to cancel the U^{-1} (since U is unitary, and $UU^{-1} = I$) . Then, after measurement we are left with the target value t.

Search Starting from the Zero State

Grover's generalized algorithm allows the creation of new algorithms simply by selecting a unitary operator U and a starting state $|s\rangle$. Notice if we select $U = H^{\otimes n}$ (the n-bit Hadamard transform, where $N = 2^n$), and start at $|s\rangle = |0\rangle^{\otimes n}$ (all qubits initialized to $|0\rangle$), we get the original version of Grover's search algorithm. This can be seen by noting that in this case, $I_s = I_0$ is the operator that takes $|x\rangle \rightarrow (-1)^{\delta_{x0}}|x\rangle$, and I_t is the operator that takes $|x\rangle \rightarrow (-1)^{f(x)}|x\rangle$ for each state $|x\rangle$. Noting also that $\left(H^{\otimes n}\right)^{-1} = H^{\otimes n}$, we can see that $G = -I_s U^{-1} I_t U$ is the same as the Grover iteration for the original search algorithm we have shown above.

Search Starting from Any State

The search algorithm does not need to start from the zero state. It could start from any one of the $N = 2^n$ computational basis states, since the probability of going from any of these states to the target state is $|U_{ts}|^2 = \frac{1}{N}$. Starting at any of the basis states, and repeatedly applying the Grover iteration would give us an equally efficient $O(\sqrt{N})$ algorithm.

Median Estimation

Suppose we are given $N = 2^n$ numbers, denoted by $x_0, x_1, \ldots x_{N-1}$, then we will define the ε-approximate median of these numbers to be the number x_i such that the number of x_j less than it, and the number of x_j more than it, are both less than $(1+\varepsilon)\frac{N}{2}$ (where is ε is a small positive number) . The best classical algorithm to get to get such a value x_i is $\Omega\left(\frac{1}{\varepsilon^2}\right)$. But the quantum algorithm, using Grover's generalized framework is $O\left(\frac{1}{\varepsilon}\right)$.

As Grover puts it in his paper: we will consider a certain “threshold” θ, such that the number of values below θ is $\frac{N}{2}(1+\varepsilon)$. Then, given the bound $|\varepsilon| < 2\varepsilon_0$, the task is to find an estimate ε_e such that $\left||\varepsilon_e| - |\varepsilon|\right| < \frac{\varepsilon_0}{4}$.

Now, if we associate a quantum state $|j\rangle$ (which will be composed of n qubits), with each of the N numbers, we can consider a unitary transform R which is a selective inversion operation such that: if the value x_j is smaller than θ, then we will invert the amplitude in $|j\rangle$.

If we start with the system in the zero state, and consider the unitary operator $U = H^{\otimes n} R H^{\otimes n}$, after one application of U, the probability of the system being in the state $|0\rangle$ is ε^2. After $O\left(\frac{1}{\varepsilon_0}\right)$ applications of $G = -I_0 U^{-1} I_0 U$, followed by one more application of U and a measurement, the probability of getting 0 is $\left(\frac{\varepsilon}{4\varepsilon_0}\right)^2$. Then, it is possible to estimate ε within the needed error bound of $\frac{\varepsilon_0}{4}$.

Mean Estimation

Grover also mentions a mean estimation algorithm in his generalized framework paper, but we will omit the details since the algorithm described in the next section is even more general and easier to describe. The most important point is that it is an $O\left(\frac{1}{\varepsilon}\right)$ algorithm, just like the median estimation algorithm.

FINANCIAL APPLICATIONS

A Quantum Algorithm for Integration

In (Abrams & Williams, 1999), the authors put forward a quantum algorithm for computing multi-dimensional integrals based on Grover's amplitude amplification method. This method gives a quadratic speed increase over the classical Monte Carlo method for integration, and an exponential speed increase over the classical deterministic integration algorithm.

Classical Integration Methods

Consider the general problem of integration. We seek to integrate a d-dimensional function $g(X)$, where X is a d-dimensional vector. Without loss of generality, we can scale the domain of integration to be the unit hyper-rectangle given by $\Re = [0,1]\times[0,1]\times\ldots\times[0,1]$. Then, we wish to calculate the integral I:

$$I = \int_{\Re} g(X)dX \; .$$

We can approximate the function $g(X)$ by another function $f(A)$, where each element x_i in X is replaced by $\frac{a_i}{M}$ in A, where the a_i values are integers in the range $[1, M]$ such that $f(A) = g\left(\frac{1}{M}A\right)$. This allows us to approximate the integral I with the d-dimensional sum S, as follows:

$$S = \frac{1}{M^d} \sum_{[1,M]^d} f(A).$$

Note that this sum is the same as the average of f over all the possible input vectors A.

We should also note that a sum of this form can be used to calculate the moments of a stochastic process. Consider a stochastic process given by a sequence of values: $w_1, w_2, \ldots w_N$ where each w_i is a random selection from some distribution (for example, we could use the log-normal distribution for stock prices in the Black-Scholes world). We might want to find some properties of the function $v(W)$, where W is a vector made from the elements of the sequence. Suppose we want to approximate the moments (mean, variance, skewness, kurtosis, etc.) of this function. We can easily transform this function into a function of the type g by making a change of variables. We can write each w_i as a function $w_i(r_i, w_1, w_2, \ldots w_{i-1})$ where this r_i is a random selection from the interval $[0,1]$. Then, we can write $v(R)$, where R is a vector built from the random r_i values. Since each $r_i \in [0,1]$, we have a function of the same form as g above, and we can approximate the mean by S. We can get higher moments by doing the same with the functions $v^2, v^3,$ etc.

We have so far described the deterministic algorithm for evaluating the integral I. Since we need to make M^d function evaluations, it is obvious that this algorithm is $O(M^d)$. Obviously, this is very impractical for integrals of high dimensionality. The classical Monte-Carlo method for evaluating the integral doesn't evaluate the function at every point. In the Monte-Carlo algorithm, we randomly choose the A vectors and build an estimator $\hat{S}$. After m random choices of A^i vectors, we get:

$$\hat{S} = \frac{1}{m} \sum_{i=1}^{m} f(A^i) .$$

This method depends on the law of large numbers, which tells us: $\lim_{m \to \infty} \hat{S} = S$ with probability 1. Also, the central limit theorem can be applied to tell us that we can determine S with accuracy ε in $O\left(\frac{1}{\varepsilon^2}\right)$ steps. Note, that the Monte-Carlo method doesn't depend on the dimensionality, and is therefore much more practical for integrating functions with a high number of dimensions.

Quantum Integration

In this section, we will describe the quantum algorithm for integration using amplitude amplification (Abrams & Williams, 1999). Suppose we have an estimate for S, which we will call E, and let us define the difference: $D = S - E$. We will define a new function: $f' = f - E$. Then:

$$D = S - E = \left(\frac{1}{M^d} \sum_{[0,M-1]^d} f(A) \right) - E = \frac{1}{M^d} \sum_{[0,M-1]^d} f'(A) .$$

So, we can interpret D as the average value of f'. The essential quantum part of the algorithm is to estimate the average value of f', and we iterate to find better estimates. Supposing we have $d\log_2 M + 1$ qubits, we can label the states $|r\rangle|a_{1,}a_2,\ldots a_d\rangle = |r\rangle|A\rangle$, where the first qubit $|r\rangle$ is a work qubit and the rest describe the input vector A. Here, we need $n = \log_2 M$ qubits to represent the integer values for each element on function input. This implies that $M = 2^n$, for some integer n, which gives a restriction on which values of M we can use for our approximation function f.

The quantum computer is initialized into the zero state:

$$|0\rangle|00\ldots0\rangle = |0\rangle^{\otimes(1+d\log_2 M)},$$

and the Hadamard transformation $H^{\otimes d\log_2 M}$, is applied to the function qubits to obtain an equal superposition of all possible values for A:

$$|\Psi_1\rangle = \frac{1}{\sqrt{M^d}} \sum_{[0,M-1]^d} |0\rangle|A\rangle,$$

Then, we rotate the first qubit by an amount f'. The state is then:

$$|\Psi_2\rangle = \frac{1}{\sqrt{M^d}} \sum_{[0,M-1]^d} \left(\sqrt{1-f'(A)^2}|0\rangle|A\rangle + f'(A)|1\rangle|A\rangle\right).$$

Next, we perform the inverse of the Hadamard transform used in the first step. We can now see that the amplitude of the state $|1\rangle|00\ldots0\rangle$ will be D (since each state $|1\rangle|A\rangle$ contributes amplitude $\frac{1}{\sqrt{M^d}} f'(A)$ to the state $|1\rangle|00\ldots0\rangle$). Therefore, an estimate for D could be obtained by repeating the above process, measuring, and counting the frequency of the result $|1\rangle|00\ldots0\rangle$. We would get an accuracy of ε in $O\left(\frac{1}{\varepsilon^2}\right)$ measurements.

The method described thus far only has the same asymptotic speed as the classical Monte-Carlo method. However, we can speed it up by introducing some amplitude amplification to increase the accuracy of our estimate. We can consider the steps applied so far as a single unitary operation U, which has an amplitude $|U_{ts}|$ between the starting state $|s\rangle = |0\rangle|00\ldots0\rangle$, and the target state $|t\rangle = |1\rangle|00\ldots0\rangle$. Therefore, we can use Grover's generalized framework for amplitude amplification to increase the probability of measuring this target state. However, there is a limit on the amount of amplification which can be applied for each estimate of D. For this reason, we will need to run the entire algorithm a few times. As we get better estimates for D, we can apply more amplitude amplification. The number of total iterations will be small, so the complexity will be dominated by the amplitude amplification at the last iteration, which, as we have seen is $O\left(\frac{1}{\varepsilon}\right)$ to get an estimate with accuracy ε.

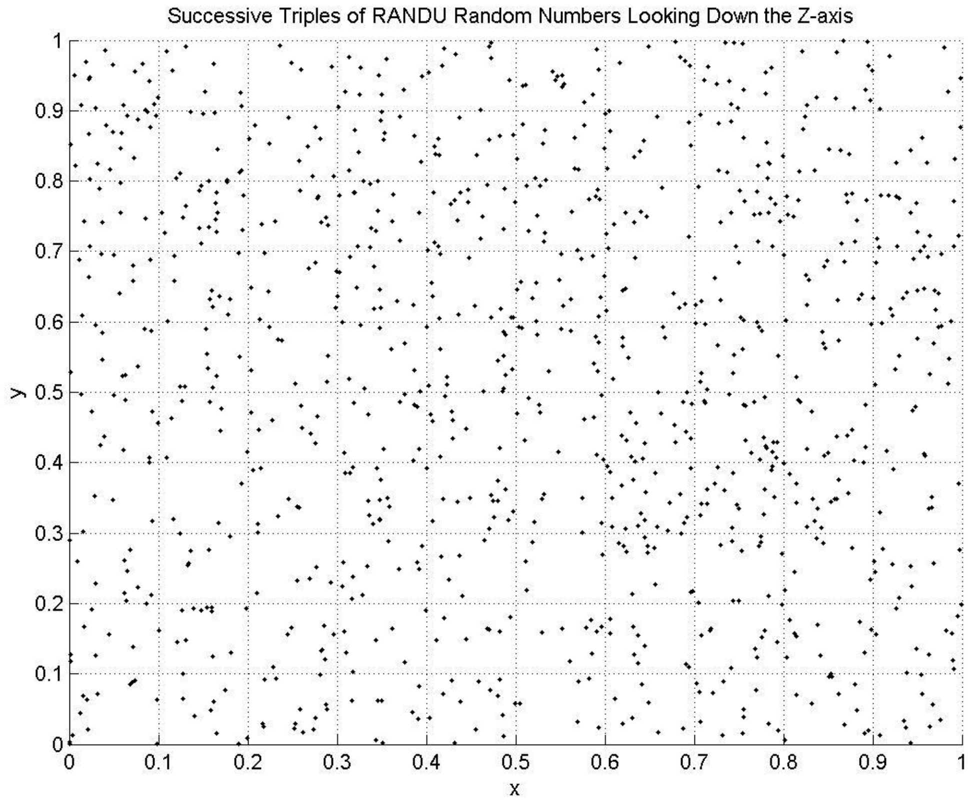
Quantum Random Number Generation

Reliable random number generation is a fundamental part of any Monte Carlo computation. However, the computation of true random numbers is not possible on a classical computer. The "random numbers" which we use for our Monte Carlo computations and stochastic simulations are actually pseudo-random numbers, which are created by a deterministic function, designed to pass many statistical tests of randomness (Zak & Williams, 1999). Sometimes numbers generated in this way can contain subtle correlations, and can therefore cause undesired results. In the case of derivative pricing using Monte Carlo, this could cause undesired correlations in the asset price paths, which could cause unintended correlations in prices.

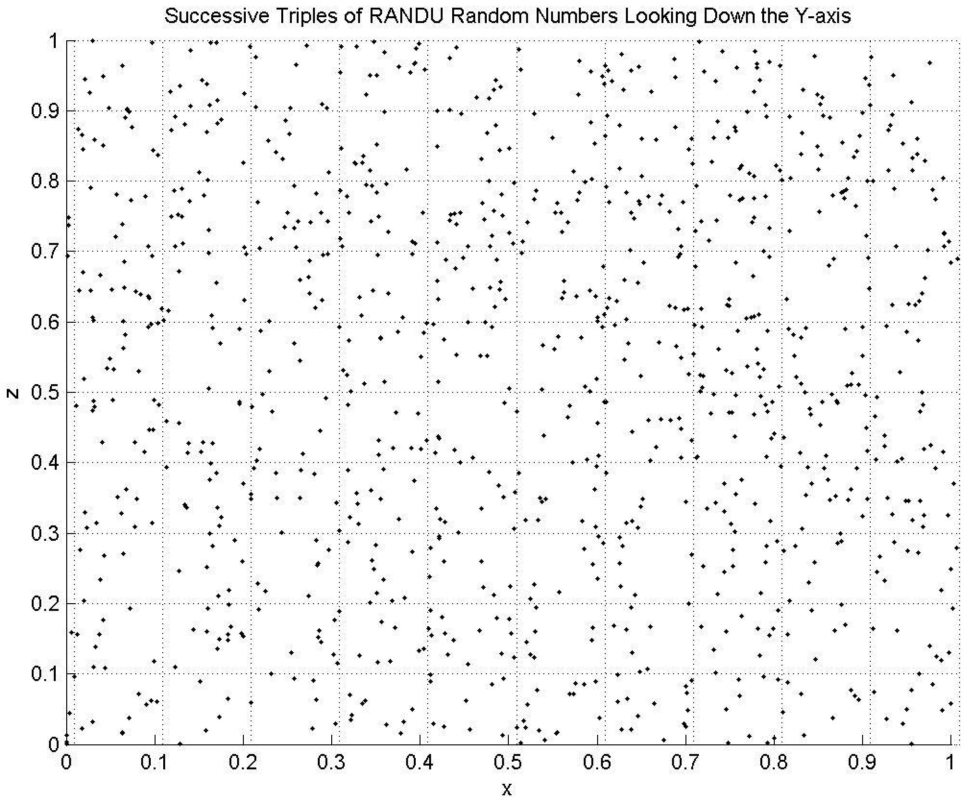
As an extreme example, we can consider the RANDU linear congruential generator (Zak & Williams, 1999). This algorithm for calculating pseudo-random numbers was common on IBM mainframes of the 1960s. A linear congruential generator is defined by:

$$N_{k+1} = (lN_k + m) \bmod n$$

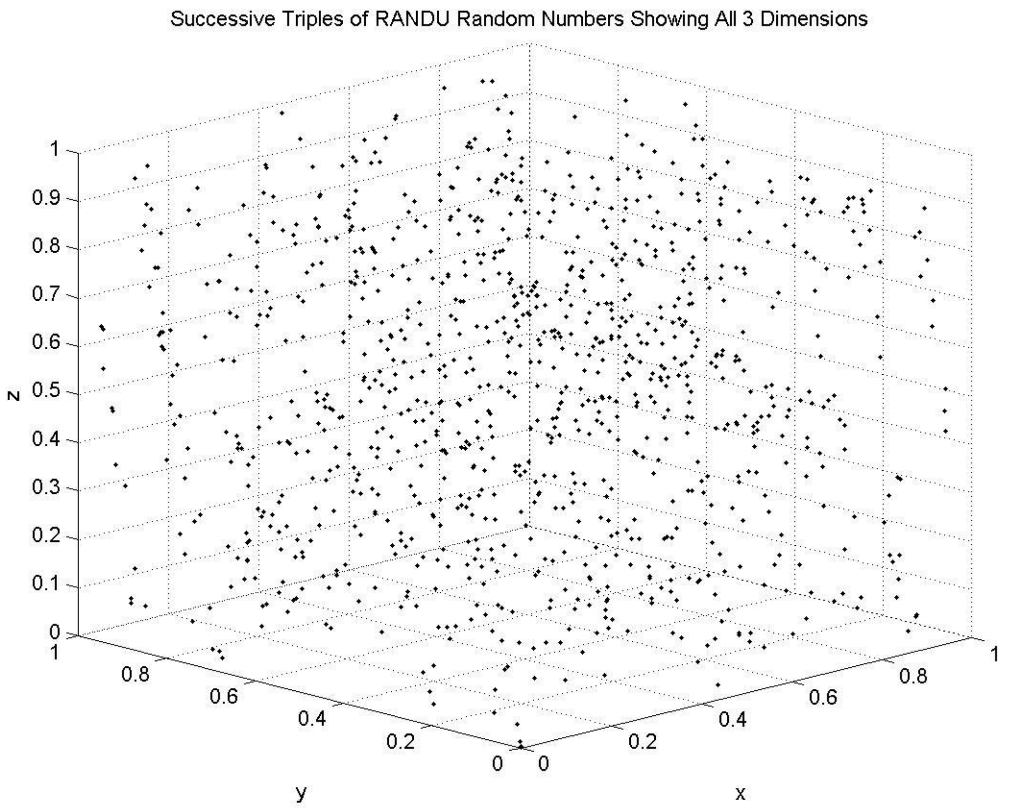
where l, m, n are integers. The sequence of numbers generated, N, appear to be a set of random numbers in the range 0 to $n-1$. These numbers would pass many tests of randomness. However, if we use $l = 65539$, $m = 0$, $n = 2^{31}$, and $N_1 = 1$ (as was used in the RANDU algorithm), and we take 3 successive triples produced by the generator to be the (x, y, z) coordinates of a 3-dimensional space, we can inspect the numbers from different angles to find a disturbing result.



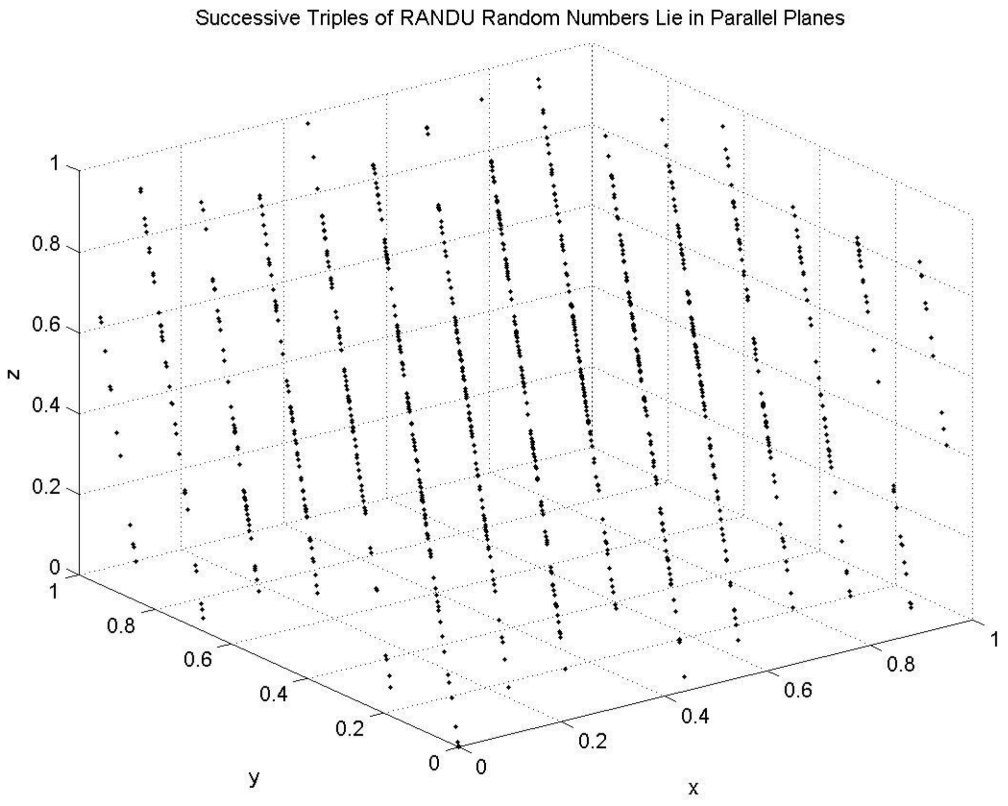
When shown from this angle, the random numbers seem to have no pattern. These numbers look like they would be useful random numbers in a 2-dimensional space.



From this angle, the numbers also look random - so far, they appear to be random looking directly down two of the axes.



Here we look at the plotted triples of random number by the side, and it still looks very disordered. However, if we change the angle slightly, we can see the disturbing truth.



As we can clearly see in this last case, all the numbers actually lie in a set of parallel planes. Therefore, these numbers are not random at all, and could give misleading results if used in a numerical simulation of a stochastic process.

Although pseudo-random number generators are better today than they were in the 60s, hidden correlations could still be lurking. For example, as recently as 1992 there was a problem found in a supposedly good random number generator (Ferrenberg, 1992). The lesson is that the only way to be truly confident about the randomness in our numbers is to use a random source for generating them.

The quantum algorithm for random number generation is extremely simple, and gives us true random numbers, due to the inherent randomness of the quantum system. We can get a random number in the range 0 to $2^n - 1$ by preparing an n-qubit register in the zero state: $|0\rangle^{\otimes n}$, applying the Hadamard transform to each qubit, and then measuring the result, which will randomly give us one of the 2^n possible states. We could then use a function transformation to transform these uniform random numbers into random numbers from any other distribution.

These true quantum random numbers could be used as the first step in a Monte-Carlo simulation - in which random numbers are calculated. As described in (Glasserman, 2003), we could replace the usual pseudo-random number generation by quasi-random numbers which are not actually random, but have properties that allow the entire space to be efficiently filled. Likewise, we could replace the usual pseudo-random number generation with this true quantum random number generation to use a set of random numbers which are theoretically free from any unwanted correlations like the hidden correlations demonstrated above.

CONCLUSION

The importance of Monte-Carlo in computational finance cannot be overestimated. Monte-Carlo simulation is the only practical method for consistently analysing portfolios of financial instruments dependent on multiple market factors. It is therefore an indispensable tool for risk management. However, the quantum algorithm for Monte-Carlo as described above only gives a modest improvement in speed. At the time of writing, the state of the art in quantum computer hardware is nowhere near able to implement even the simplest version of the algorithm. Therefore, the expense of developing the quantum computer technology far outweighs the potential benefits of the slight speed increase available, especially after considering that this increase could also be obtained simply by the addition of extra classical computational resources.

To catalyze the development of quantum computer technology, an exponentially faster algorithm with wide applications in finance needs to be discovered. Quantum algorithms which are exponentially faster than any known classical algorithms do exist, such as Shor’s factoring algorithm (Shor, 1994). Unfortunately, these algorithms do not have any practical or pervasive use in finance.

The quantum random number generation algorithm seems to be the most practical algorithm for use in finance. Although the speed increase is not necessarily important, the ability to generate true random numbers may prove to be more important in the future. As derivatives markets become more efficient, price differences caused by subtle correlations in random number generators may lead to potential arbitrage opportunities. Pricing using true random numbers may prove to be the only way to remove these inaccuracies.

Although known applications for quantum computers in finance are quite weak at the time of writing, technological advancements might come at any time, and it is a good idea to be prepared for the possible implications of quantum computing technology, since the inability to adapt to new technology is a source of risk which should not be underestimated.

APPENDICES

Appendix A: Dirac Notation and Tensor Products

The standard notation used in the quantum mechanics literature is to denote quantum states in the following form:

$$|X\rangle,$$

where X is the label for the quantum state, and the "$|\ \rangle$" enclosure indicates that it is a quantum state. This notation is known as Dirac notation, after the inventor Paul Dirac, who was one of the pioneers in theoretical quantum mechanics.

Since a quantum state is a vector, we could have used many other notations to denote a vector, some examples are:

$$\vec{X}\ ,\ \overline{X}\ ,\ \underline{X} \text{ or just } X\ .$$

However, one advantage of the Dirac notation is that it can be used both to describe a column vector (which are the basis states we use in quantum computing) and the transpose – a row vector. If we have a real, finite dimensional quantum state, we can use $|X\rangle$ to denote the column vector for the state, and $\langle X|$ to denote the equivalent row vector, or transpose. The "$\langle\ |$" enclosure is known as a "bra", where the "$|\ \rangle$" enclosure is known as a "ket". This is why the notation is sometimes known as bra-ket notation.

If you multiply the bra by the ket, you get the inner product, which is a very useful operation in linear algebra. Usually, in the bra-ket multiplication, one of the lines are removed, so:

$$\langle X | Y \rangle \equiv \langle X \| Y \rangle .$$

Another way to multiply is the ket by the ket, or the bra by the bra. These are defined to denote the *tensor product* of the two states:

$$| X \rangle | Y \rangle \equiv | X \rangle \otimes | Y \rangle ,$$

which is the way larger quantum states are built from smaller quantum states. In this notation, we have many different ways of writing the tensor products between two states, since it is very common:

$$| X \rangle | Y \rangle \equiv | X , Y \rangle \equiv | XY \rangle .$$

These are all equivalent, and usually the context will determine which form is used. To make these ideas more clear, it might help to see some examples.

Consider the computational basis states for a single qubit. These are denoted by $|0\rangle$, and $|1\rangle$. Since quantum states are vectors, we can define them explicitly as follows:

$$|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \text{ and } |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Then, the inner products of all combinations of these states are:

$$\langle 0 | 0 \rangle = \langle 0 \| 0 \rangle = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 ,$$

$$\langle 0|1\rangle = \langle 0\|1\rangle = \begin{bmatrix}1 & 0\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix} = 0,$$

$$\langle 1|0\rangle = \langle 1\|0\rangle = \begin{bmatrix}0 & 1\end{bmatrix}\cdot\begin{bmatrix}1\\0\end{bmatrix} = 0,$$

$$\langle 1|1\rangle = \langle 1\|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}\cdot\begin{bmatrix}0\\1\end{bmatrix} = 1.$$

Therefore, we can see that the inner product of the computational basis states is the Kronecker delta:

$$\langle x|y\rangle = \delta_{xy},\ x \in \{0, 1\}.$$

The definition of the tensor product of two matrices is as follows: supposing X is an m-by-n matrix, and Y is a p-by-q matrix, then the matrix representation of the tensor product is:

$$X \otimes Y = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} \\ X_{21} & X_{22} & \cdots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} \end{bmatrix} \otimes \begin{bmatrix} Y_{11} & Y_{12} & \cdots & Y_{1q} \\ Y_{21} & Y_{22} & \cdots & Y_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ Y_{p1} & Y_{p2} & \cdots & Y_{pq} \end{bmatrix} = \begin{bmatrix} X_{11}Y & X_{12}Y & \cdots & X_{1n}Y \\ X_{21}Y & X_{22}Y & \cdots & X_{2n}Y \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1}Y & X_{m2}Y & \cdots & X_{mn}Y \end{bmatrix},$$

and therefore, $X \otimes Y$ is an mp-by-nq matrix.

Therefore, if we have two vectors: suppose $|X\rangle$ is an m-element column vector (or an m-by-1 matrix), and $|Y\rangle$ is an n-element column vector (or an n-by-1 matrix), then the tensor product between them is:

$$|X\rangle|Y\rangle = |X\rangle \otimes |Y\rangle = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} \otimes \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} X_1|Y\rangle \\ X_2|Y\rangle \\ \vdots \\ X_m|Y\rangle \end{bmatrix},$$

Which is an mn-by-1 vector. The tensor product can also be generalized to tensor exponentiation, for example: $|X\rangle^{\otimes 3} = |X\rangle \otimes |X\rangle \otimes |X\rangle$.

The tensor product is useful in quantum computing when building registers of qubits from single qubits. For example, if we consider all the possible combinations of the computational basis states for the single qubit, which are defined as:

$$|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \text{ and } |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

then we get:

$$|0\rangle|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0\begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$|0\rangle|1\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0\begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

$$|1\rangle|0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0\begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

$$|1\rangle|1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\begin{bmatrix}0\\1\end{bmatrix}\\1\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}.$$

This set of vectors can be used as a basis for the 2-qubit registers, since they are orthonormal and span the space. In this case, it makes sense to use the $|xy\rangle$ form for the tensor product, to define this new basis, and if we interpret all these two digit numbers as binary digits, we can re-label them as:

$$|0\rangle = |00\rangle = |0\rangle|0\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix},$$

$$|1\rangle = |01\rangle = |0\rangle|1\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix},$$

$$|2\rangle = |10\rangle = |1\rangle|0\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix},$$

$$|3\rangle = |11\rangle = |1\rangle|1\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}.$$

The utility of Dirac's notation becomes very noticeable when we build even larger registers of qubits.

REFERENCE LIST

Aaronson, S. (2008) The Limits of Quantum. *Scientific American, 298 (3)*, p. 62.

Abrams, D. S. & Williams, C. P. (1999) Fast quantum algorithms for numerical integrals and stochastic processes. *arXiv:quant-ph*, article 9908083v1. Retrieved March 7, 2007, from http://arxiv.org/PS_cache/quant-ph/pdf/9908/9908083v1.pdf.

Aharanov, D. (1999) Quantum computation. In D. Stauffer (Ed.), *Annual Reviews of Computational Physics VI*. Singapore: World Scientific.

Bacon, D. & Leung, D. (2007) Toward a World with Quantum Computers. *Communications of the ACM, 50 (9)*, p. 55.

Deutsch, D. (1999) Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond. A, 400*:97.

DiVincenzo, D. P. (1995) Two-bit gates are universal for quantum computation. *Phys. Rev. A, 51*(2), p. 1015-1022.

Ferrenberg, A. M., & Landau, D. P. (1992) Monte Carlo simulations: hidden errors from “good” random number generators. *Physical Review Letters, 69*(23), p. 3382-3384.

Glasserman, P. (2003) *Monte Carlo Methods in Financial Engineering*. New York: Springer-Verlag.

Grover, L. K. (1996) A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual Symposium on the Theory of Computing (STOC)*, p. 212-219.

Grover, L. K. (1998) A framework for fast quantum mechanical algorithms. *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC)*, p. 53-63.

Knights, M. (2007) The Art of Quantum Computing. *Engineering & Technology, 2 (1)*, p. 30.

Nielsen M., & Chuang, I. (2000) *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press.

Powers, M. R. (2008) The Nature of Randomness. *Journal of Risk Finance, 9 (1)*, p. 5.

Shor, P. (1994) Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, p. 124-134.

Williams, C. P., & Clearwater, S. H. (2000) *Ultimate Zero and One, Computing at the Quantum Frontier*. New York: Springer-Verlag.

Zak, M., & Williams, C. P. (1999) Quantum Recurrent Networks for Simulating Stochastic Processes. *Springer-Verlag Lecture Notes in Computer Science, Volume1509*, Heidelberg: Springer-Verlag.