Option Pricing using Canonical Amplitude Estimation

Constantin Kurz MSc.
Thomas Jung MSc.
Deepankar Bhagat MSc.
Adil Acun PhD.

September 13, 2022

Foreword

This report was originally intended as a report for the course Introduction to Quantum Computing at the University of Twente. Both the report and the course are provided by ING employees. Initially, the report was meant to show maturity in quantum computing of the students attending the course. However, it became quickly interesting to upstream the report into a tutorial for, mostly colleagues, who would like to have a step-by-step introduction to quantum computing and to execute Monte Carlo simulations on such computers. The report is going to be a living document and will be updated from time to time as the field of Monte Carlo simulations on quantum computers advances and the author's experiences too. Lastly, the Monte Carlo simulations are applied in the domain of (quantitative) finance, thus, covering the financial problems too in the report. We hope that this report meets its requirements as a standalone document for the quantum computing enthusiasts in the financial world.

1 Introduction

Banks and other financial institutes employ vast computational resources for pricing and risk management of financial assets and derivatives. Without the computational resources evaluating complex mathematical models today's financial markets would be in grave danger and could not operate efficiently or at the scale it does nowadays. Examples of assets on financial markets are stocks, bonds and commodities. The assets are a base for more complex financial instruments called derivatives. A financial derivative, as it name suggests, derives its price from the future price or a price trajectory of at least one asset. As the nature of risky assets is stochastic, determining a fair price of a derivative (contract) is a challenge. A mathematical model that can be used for this pricing model is called the Black-Scholes-Merton model. In Section 2 this model is discussed in more depth and its applications on derivatives is addressed.

The Black-Scholes-Merton model is only solved analytically for very simple derivatives. For example, a simple derivative is a derivative whose payoff only relies on the future price at maturity time. A more complex derivative would be one where the payoff depends on the trajectory towards some future price at maturity time. Unfortunately, these complex derivatives have path-dependence embedded in the payoff function and combined with the stochastic nature of the underlying asset(s), it becomes infeasible (or perhaps impossible) to find an analytical solution for the pricing problem. Due to the stochastic nature of the underlying risky asset(s), it is wise to revert to a method where the expectation value of the future price (or the trajectory) is calculated by repeatedly and randomly sampling from a probabilistic distribution describing the underlying asset price (development). This method, called the Monte Carlo method, has long been explored in physics, chemistry, biology, finance, meteorology and many other domains where analytical solutions are infeasible and stochastic behaviors are observed. Random sampling from a distribution yield a higher accuracy (i.e., lower error) as the number of samples is growing. The downside of increasing the number of samples is the increased cost of computational power. A ten-fold improvement in accuracy suggests a hundred-fold longer runtime. Monte Carlo simulations can become rapidly intractable due to the scaling illustrated in the previous sentence. Thus, perhaps by changing the nature of deterministic processing units in computers to stochastic processing units, such as quantum processing units, a better scaling might be attained. Therefore, this report explores the opportunities and challenges of quantum computers regarding Monte Carlo simulations applied in the financial industry. For a more in-depth introduction on Monte Carlo simulations, please consult Section 3.

A quantum algorithm and computer capable of performing Monte Carlo simulations of financial derivatives (and the pricing problems) in a consistent and reliable way would be of utmost importance. This document discusses and shows which algorithms on quantum computers might be helpful in risk analytics and derivatives pricing.

1.1 Introduction to Quantum Computing

Gated quantum computing is similar to classical computing with logical circuits. A logical circuits is composed of bits and gates. A bit is a binary digit carrying one of the following two values exclusively: 0(*false*) or 1 (*true*). A quantum bit, abbreviated as *qubit*, is a quantum state with two base states $|0\rangle$ and $|1\rangle$ and does not exclude the presence of both states simultaneously (which is called the *superposition* principle). How and how much a base state is contributing to the qubit is determined by the amplitude (α, and β in the equation below) of each base state. Upon measuring a quantum state, hence forcing it into a classical realm, the superposition principle collapses and only one single deterministic state must be acquired. The resulting state is retrieved by a stochastic process in which the probability distribution of the base states is characterized by the absolute value squared of amplitudes. As the sum of all probability coefficients of a probability distribution must be one, the sum of all absolute value squared of amplitudes must be one too. A qubit is then defined as the equation below following both an algebraic and a vector representation respectively:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,\ \text{with } (\alpha,\beta) \in \mathcal{C},\ \text{and } |\alpha|^2 + |\beta|^2 = 1 \tag{1}$$

$$|\psi\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} \tag{2}$$

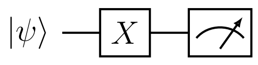
Logical gates take in a Boolean input and return a Boolean output as followed from a Boolean function embedded in that gate. For example, a NOT-gate has a Boolean formula embedded in it that inverts the binary value of an input bit. To illustrate, an input bit with value 0 would become an output bit with value 1. A quantum gate follows the same principle. The quantum gate has an input quantum state (or an amplitude vector), follows a linear algebraic formula (or a matrix) and returns an output quantum state (again, an amplitude vector). One notable differenc between classical and quantum computing is the number of bits and qubits put in and returned. In classical computing the number of input and output bits may differ, for example, the AND-gate, would require two input bits and only returns a single output bit. In quantum computing a quantum gate performs on operation on (a set of) qubits and solely transforms the states of these qubits only. More technically, a quantum gate is a unitary matrix. The unitary property, i.e., the complex conjugate transpose of a matrix multiplied by the original matrix is an identity matrix, ensures that the norm of the quantum state vector is preserved (and remains one). All unitary matrices are square too. The property of being square results in the fact that the number of qubits is preserved during the operation. An example of a quantum gate deployed on a single qubit is given below where the amplitudes of the base states are swapped:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\text{Let } |\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

$$X\,|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} = \beta\,|0\rangle + \alpha\,|1\rangle$$

The above operation can be represented as a quantum circuit as seen below. The qubit is shown as a quantum register (and is often initialised with state $|0\rangle$). Gates are blocks put on a quantum register and perform an operation on the targetted qubit. The first gate denotes the X-gate operation as exemplified above. The second gate is a measurement gate. The measurement gate collapses the quantum state into a single deterministic base state according to the probability distribution derived from the amplitudes of the quantum state.

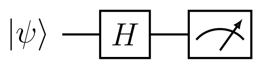


Qubits are technically quantum states, hence, are governed by a wave function as followed from/by Schrödinger's equation. The wave nature allows for *interference* where amplitudes of a base state may constructively or destructively interfere. Below is an example of interference upon using an Hadamard gate on a generic qubit. The quantum circuit is also shown below.

$$H\,|\psi\rangle = \sqrt{\frac{1}{2}} \begin{bmatrix} 1 & 1 \\ 1 & \text{-}1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} =$$

$$\sqrt{\frac{1}{2}} \begin{bmatrix} \alpha + \beta \\ \alpha - \beta \end{bmatrix} =$$

$$\sqrt{\frac{1}{2}}(\alpha + \beta)\,|0\rangle + \sqrt{\frac{1}{2}}(\alpha - \beta)\,|1\rangle$$



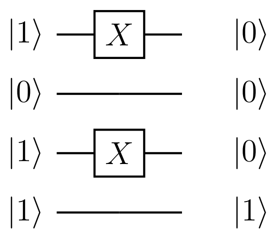
If $\alpha = \beta$, then the $|1\rangle$-state would be diminished completely. Even though the qubit started in a superposition of two base states, due to interference, the resulting qubit is now characterized by a single base state only. This does not mean the other base state is removed entirely, it is solely absent (until some other operation would redistribute the amplitudes such to acquire a non-zero amplitude on both base states).

A single qubit holds only two discrete base states representing two bits of information. Often, more bits of information is needed, thus more qubits are needed. With Boolean algebra and classical bits, multi-bit systems are 0's and 1's concatenaded horizontally from right to left. The right-most bit is the least significant bit, whereas the left-most bit is the most significant bit. For example, the integer number 13 becomes the bit-string 1101, where the left-most qubit is valued $2^3 = 8$ and right-most qubit $2^0 = 1$. Qubits are represented in various ways, all meaning the same.

$$|1101\rangle$$
$$|1, 1, 0, 1\rangle$$
$$|1\rangle\,|1\rangle\,|0\rangle\,|1\rangle$$
$$|1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle$$

A four-qubit state holds $2^4 = 16$ bits of information. Therefore, the vector representing a four-qubit state has 16 elements.

In a circuit, typically, but not always, the least significant bit is the uppermost qubit. Now suppose a circuit is initialised with the state $|1101\rangle$ and some gates account for a final state $|1000\rangle$ (where an X-gate is put on the second and last qubit from the left).



Algebraically, these operations are represented as below by only picking the first and last qubit representation from above. Please note that the sequence of the Kronecker products ($\otimes$) is crucial. Here, I is an identity matrix which preserves the same state (as multiplying any number by one with simple arithmetics).

$$(I \otimes X \otimes I \otimes X)\,|1101\rangle = |1000\rangle$$

$$\begin{gathered}(I \otimes X \otimes I \otimes X)(|1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle) = \\ I\,|1\rangle \otimes X\,|1\rangle \otimes I\,|0\rangle \otimes X\,|1\rangle = \\ (|1\rangle \otimes |0\rangle \otimes |0\rangle \otimes |0\rangle)\end{gathered}$$

A Kronecker product is a tensor product between two tensors. Usually, the tensors used in quantum computing are often scalars, vectors and matrices. Below, two examples are given of Kronecker products to illustrate what it does. Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ and $|\phi\rangle = \gamma|0\rangle + \delta|1\rangle$. The vector representation of these two states are respectively:

$$|\psi\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix}, \quad |\phi\rangle = \begin{bmatrix}\gamma \\ \delta\end{bmatrix}$$

Then the Kronecker product of the quantum states result in a composite quantum state:

$$\begin{aligned}|\psi, \phi\rangle = |\psi\rangle \otimes |\phi\rangle &= \begin{bmatrix}\alpha \\ \beta\end{bmatrix} \otimes \begin{bmatrix}\gamma \\ \delta\end{bmatrix} \\ &= \begin{bmatrix}\alpha \begin{bmatrix}\gamma \\ \delta\end{bmatrix} \\ \beta \begin{bmatrix}\gamma \\ \delta\end{bmatrix}\end{bmatrix} \\ &= \begin{bmatrix}\alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta\end{bmatrix}\end{aligned}$$

A similar example is given for two 2×2-matrices called U and V.

$$U = \begin{bmatrix}a & b \\ c & d\end{bmatrix}, \quad V = \begin{bmatrix}e & f \\ g & h\end{bmatrix}$$

The Kronecker product of U and V is then:

$$\begin{aligned}U \otimes V &= \begin{bmatrix}a & b \\ c & d\end{bmatrix} \otimes \begin{bmatrix}e & f \\ g & h\end{bmatrix} \\ &= \begin{bmatrix}a\begin{bmatrix}e & f \\ g & h\end{bmatrix} & b\begin{bmatrix}e & f \\ g & h\end{bmatrix} \\ c\begin{bmatrix}e & f \\ g & h\end{bmatrix} & d\begin{bmatrix}e & f \\ g & h\end{bmatrix}\end{bmatrix} \\ &= \begin{bmatrix}ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh\end{bmatrix}\end{aligned}$$

A pattern that one sees above is that Kronecker products are about putting the rightmost tensor inside of the leftmost tensor in the equation multiplying the rightmost tensor by every scalar of the leftmost tensor.

The inverse of a Kronecker product involves finding A and B from $C = A \otimes B$. There are no unique solutoins for this problem. However, one may consult the Pitsianis-Van

Loan algorithm for the approximation of A and B by a minimisation function (if the two matrices are real valued). For the sake of illustration, consider the vector below:

$$x = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 0 \end{bmatrix}$$

The goal is to find two vectors u and v such that $x = u \otimes v$. Just by trial and error one would find:

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad v = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This means that a 4-dimensional vector can be decomposed in two 2-dimensional vectors. Regarding qubits, this would mean that a composed two-qubit system is decomposed in two independent qubits, as each independent qubit has only two elements in its vector representation.

It is possible to have a multi-qubit state in which it is not possible to invert the Kronecker product. Consider the following problem:

$$|\psi\rangle = \sqrt{\frac{1}{2}}\,|00\rangle + \sqrt{\frac{1}{2}}\,|11\rangle = \sqrt{\frac{1}{2}} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \qquad \text{find } |u\rangle \text{ and } |v\rangle \text{ for which holds } |\psi\rangle = |u\rangle \otimes |v\rangle$$

It is not possible to find valid quantum states $|u\rangle$ and $|v\rangle$ for this problem, even though $|\psi\rangle$ is a valid two-qubit state. It is then said that the two qubits are strongly correlated, or entangled, meaning that that $|\psi\rangle$ acts like a single particle composed of two strongly correlated qubits. Suppose $|\psi\rangle = \sqrt{\frac{1}{2}}\,|00\rangle + \sqrt{\frac{1}{2}}\,|11\rangle$ to be measured and the leftmost qubit is measured. There is a probability of $\frac{1}{2}$ that the leftmost qubit will collapse in a 0-state or in a 1-state. However, it would collapse into a 0-state, then the remaining qubit is also going to be in a 0-state as it is the only option left. With that, the superposition of the second qubit immediately collapses when the first qubit is measured. One bit of measurement hence gives two bits of information, one can say.

Entanglement is useful in many occassions in quantum computing. Suppose an initial two state system to be $|x\rangle\,|0\rangle$ and that there is a quantum gate that embeds a function $f(x)$ that provides $|x\rangle\,|f(x)\rangle$. By entanglement, any input x is connected to its corresponding output $f(x)$. If the input-register is a superposition of many states, i.e., many inputs, then the output-register will host a superposition of many outputs at the same time. It is therefore not a surprise that including superposition and entanglement into a quantum circuit leverages speedups. With interference included in the post-processing of the superposed and entangled circuit, algorithms can be designed that might give speedups to classical counterparts.

2 Black-Scholes-Merton-Model and European Call Options

The Black-Scholes-Merton (BSM) model considers the pricing of financial derivatives (*options*). The original model assumes a single benchmark asset (*stock*), the price of which is stochastically driven by a Brownian motion, see fig 1. In addition, it assumes a risk-free investment into a bank account (*bond*).

The Black-Scholes-Merton model consists of two assets, one risky (the stock), the other one risk-free (the bond). The risky asset is defined by the stochastic differential equation for the price dynamics given by

$$dS_t = S_t r dt + S_t \sigma dW_t, \tag{3}$$

where r is the drift, σ the volatility and dWt is a Brownian increment. The initial condition is S0. In addition, the risk-free asset dynamics is given by

$$dB_t = B_t r dt, \tag{4}$$

where r is the risk-free rate (market rate).
the risky asset stochastic differential equation can be solved as

$$S_i = S_0 \cdot \exp^{\sigma \cdot W_i + (r - \frac{\sigma^2}{2})T} \tag{5}$$

$$W_i = \mathcal{N}(\mu = 0, \sigma^2 = T). \tag{6}$$

The risk-free asset is solved easily as

$$B_t = e^{rt}. \tag{7}$$

This risk-free asset also is used for 'discounting', i.e. determining the present value of a future amount of money. One of the simplest options is the European call option. The European call option gives the owner of the option the right to buy the stock at maturity time $T \geq 0$ for a pre-agreed strike price K.
The payoff is defined as

$$f(S_T) = \max(0, S_T - K). \tag{8}$$

The task of pricing is to evaluate at present time $t = 0$ the expectation value of the option $f(S_T)$ on the stock on the maturity date. The pricing problem is thus given by evaluating the risk-neutral price

$$\Pi = e^{-rt}\mathbb{E}[f(S_T)], \tag{9}$$

where e^{-rT} is the discount factor, which determines the present value of the payoff at a future time, given the model assumption of a risk-free asset growing with r.

The asset price a maturity T follows a log-normal distribution wit probability density ([5])

$$P(S_T) = \frac{1}{S_T\sigma\sqrt{2\pi T}} exp(-\frac{(\ln S_T - \mu)^2}{2\sigma^2 T}, \tag{10}$$

where σ is the volatility of the asset and $\mu = (r - 0.5\sigma^2)T + \ln(S_0)$.

3 Monte Carlo

In this section we discuss how Monte Carlo Simulations are working and compare them to quantum computer.

3.1 Classical Monte Carlo

In general monte carlo data are generated if an experiment or an calculation doesn't have sufficient training data. To generate data using the Monte Carlo method we can following these steps ([5]):

1. At first we need a set of random variables $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$. In our case this values describe the asset price and other sources of uncertainties.
2. With this information's we can now create additional information. In our case we can build M random price paths $\{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_N\}$.
3. At the end we want to have the expected value of the payoff $\mathbb{E}_P$, for this we can calculate the payoff of every path $F(\mathbf{X}_i)$ and then build the average of the results:

$$\mathbb{E}_P[F(\mathbf{X})] = \frac{1}{M}\sum_{i=1}^{M} F(\mathbf{X}_i) \tag{11}$$

In this work we will use the "Black-Scholes-Merton" model to create the paths of the stock price. After the Black-Scholes-Merton method the value of the stock changes each step after a log-normal distribution shown in figure 1.

To calculate the end result we split the time frame T in n steps, with each step the size of $dt = \frac{T}{n}$. We need also the volatility σ and the drift r of the stock.

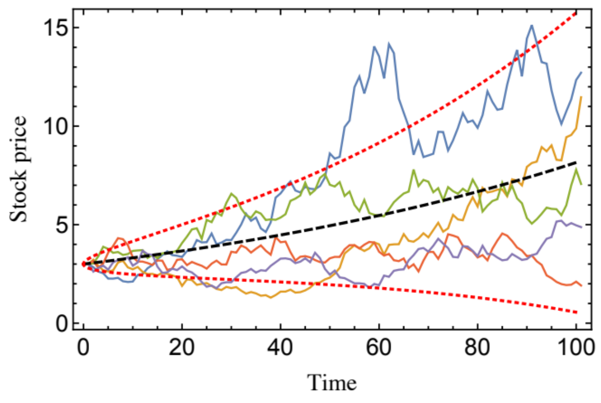


Figure 1: Image of five different paths of the stock price. The stock price is in dollar and the time is in days. The dashed black line is the mean of the samples and the dotted red lines are the first standard deviation. The parameters are: $S_0 = \$3, r = 0.1, \sigma = 0.25$ [4]

Now Π is the true option price and $\hat{\Pi}$ is the approximation of the option price calculated via the Monte Carlo data, with M samples. With this values and the condition that the variance of the payoff function is $\leq \lambda^2$ the probability that $|\Pi - \hat{\Pi}| \leq \epsilon$ can be determined by Chebyshev's inequality ([3]).

$$P[|\Pi - \hat{\Pi}| \leq \epsilon] \leq \frac{\lambda^2}{M\epsilon^2} \tag{12}$$

So to achieve an error of ϵ

$$M = \mathcal{O}(\frac{\lambda^2}{\epsilon^2}) \tag{13}$$

samples are needed. Section 3.2 shows that quantum computer can improve this from ϵ^2 to ϵ.

3.1.1 Example

In this subsection a easy Monte Carlo method is shown. Therefor we create only four different stock path and only the last value is of interest, so we get this values:

$$\{\mathbf{X}_1 = 1.5, \mathbf{X}_2 = 2.3, \mathbf{X}_3 = 2.8, \mathbf{X}_4 = 2.11\}$$

We use $S_0 = 2$ as initial value, $K = 2.1$ as strike price and this

$$F(\mathbf{X}_i) = \max\{0, X_i - K\} \tag{14}$$

as the payoff function. With this kind of information's we can now calculate our expected payoff

$$\begin{aligned}\mathbb{E}_P[F(\mathbf{X})] &= \frac{1}{4}(0 + 0.01 + 0.2 + 0.7)\\ &= 0.2275\end{aligned}$$

3.2 Quantum Monte Carlo

In the quantum field three problems has to be solved.([5])

1. Load the probability Estimation (the log-normal distribution) in to the quantum computer. This problem is described in section 3.2.1
2. The payoff function $F(\mathbf{X})$ has to be build with qubits inside the quantum computer. In section 3.2.3 the payoff function is constructed.
3. The quantum computer should calculate at the end the expected value of the payoff $\mathbb{E}_P[F(\mathbf{X})]$, this will also be explained in section 3.2.3.

But before we can describe how to construct the gates to build the payoff function, we need to explain how amplitude estimation is working in an quantum computer. We are doing this in section 3.2.2

3.2.1 Loading the distribution (QuGAN)

In this section we go over the basics of the QuGAN, because we use an already implemented function, which is not QuGAN, of qiskit ([1]) later to load in the log-normal distribution.

QuGAN stands for Quantum Generative Adversarial Networks and it creates fake data which are similar to the real data. The GAN method is used when there are only a few real world data or to bring the data in to a new system. The last one is exactly our purpose here. We want to have the log-normal distribution inside a quantum computer. For this we divide the log-normal distribution in distinct buckets n. The value of n has to be the number of states the k qubits can have $n = 2^k$. So if we have 3 qubits we have 8 distinct buckets. Every state of all the k qubits maps to a real value.

The structure of an QuGAN is shown in figure 2. So a QuGAN goes throw this steps:

1. We load in at first the classical dataset, transform the Dateset and with this dataset initial parameter for the gates can be created.
2. With this parameters the quantum circuit can be created.
3. The quantum circuit is then loaded into a quantum computer.
4. After optimise steps in the quantum computer the quantum state is measured.
5. The quantum state is then transformed in to the classical world and the loss is calculated. Also the Quantum Circuits can create sample states and this sample states are then used to generate classical data points.
6. With the loss the parameters of the circuit can be update. After updating the circuits parameter we go back to step 2.

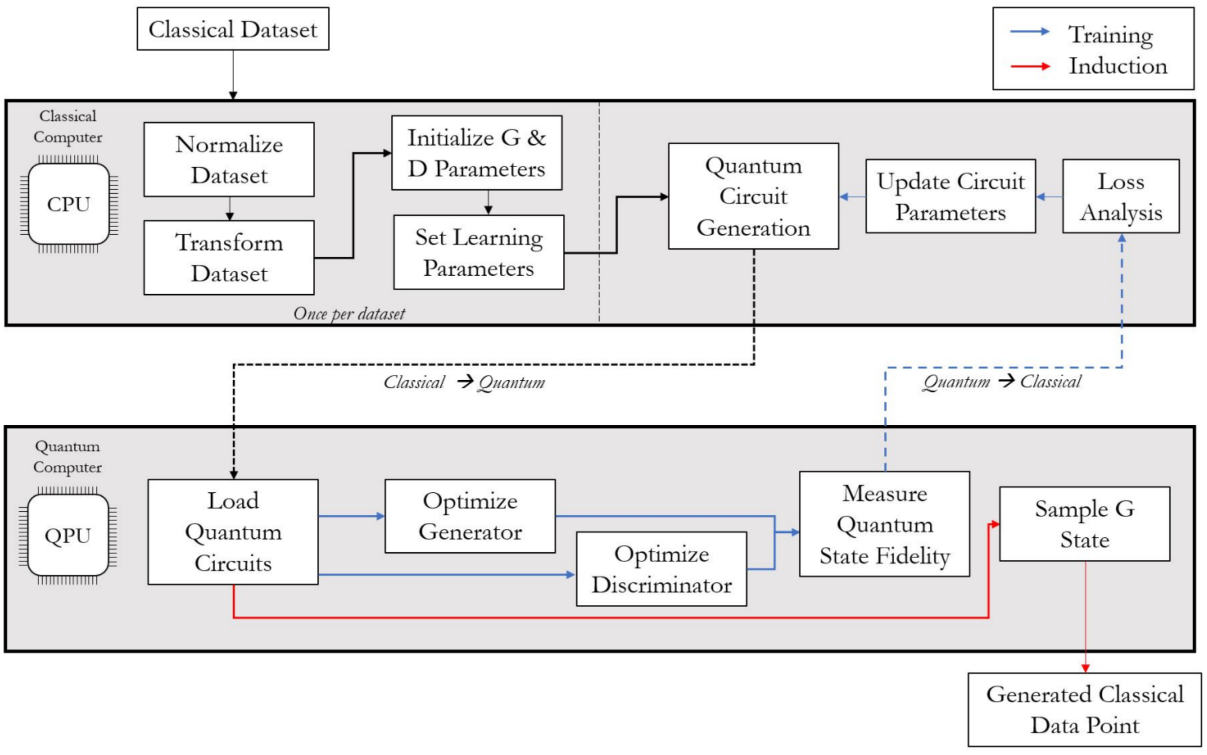


Figure 2: QuGAN System Architecture [6]

3.2.2 Amplitude Estimation

For the Amplitude Estimation (AE) we need an unitary operator A which behaves on an register of $(n+1)$ qubits like this

$$A\left|0\right\rangle_{n+1} = \sqrt{1-a}\left|\psi_0\right\rangle_n\left|0\right\rangle + \sqrt{a}\left|\psi_1\right\rangle_n\left|1\right\rangle . \tag{15}$$

In the equation 15 $\left|\psi_0\right\rangle_n$ and $\left|\psi_1\right\rangle_n$ are some normalized states and a is a unknown variable $\in [0,1]$. With Amplitude Estimation we can estimate the value of a therefor we have to construct our problem so that we have this at the end $a = \mathbb{E}_P[f(\mathbf{X})]$. To make a use of sin and cos to work with RY-Gates we can say that $a = \sin^2(\theta_a)$ is, so that equation 15 turns into this

$$A\left|0\right\rangle_{n+1} = \cos(\theta_a)\left|\psi_0\right\rangle_n\left|0\right\rangle + \sin(\theta_a)\left|\psi_1\right\rangle_n\left|1\right\rangle . \tag{16}$$

$$\theta_p = 2\theta_a$$

$$A = R_Y(\theta_p) \tag{17}$$

With this changes we transformed the amplitude estimation in to a phase estimation. The next step is to determine the Grover operator $\mathcal{Q} = AS_0A^\dagger S_{\psi_1}$.
S_0 is a reflection about the $\left|0\right\rangle$ state and S_{ψ_1} is a reflection about the $|\psi_1>$ state. So we can write S and $\mathcal{Q}$ like this:

$$S_0 = 1 - 2\left|0\right\rangle\left\langle 0\right|$$

$$S_{\psi_1} = 1 - 2\left|\psi_1\right\rangle\left|0\right\rangle\left\langle\psi_1\right|\left\langle 0\right|$$

$$\mathcal{Q} = R_Y(2\theta_p) \tag{18}$$

$$\mathcal{Q}_{2^j} = R_Y(2^j \cdot 2\theta_p) \tag{19}$$

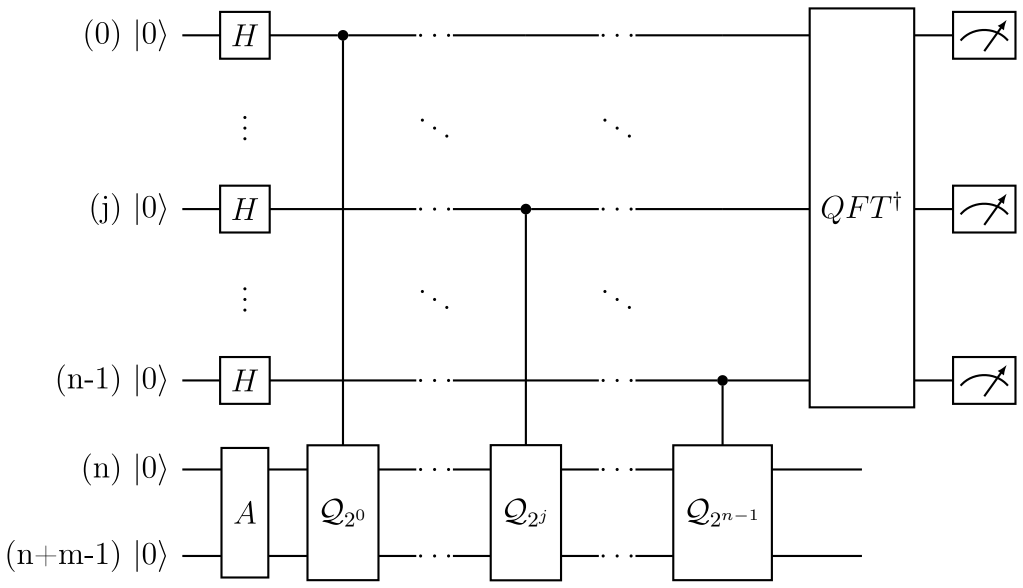


Figure 3: Design of the Amplitude Estimation recreated after [5]

The error ϵ of this method defined as followed

$$|\hat{\Pi} - \Pi| \leq C \cdot (\frac{\sqrt{\mathbb{E}}}{M} + \frac{1}{M^2})$$
$$M = \mathcal{O}(\frac{1}{\epsilon\sqrt{\mathbb{E}}}) \tag{20}$$

As we can see in 13 and 20 quantum computer improved the number error with the same number of samples

3.2.3 Construct the payoff function

A lot of payoff function can be constructed by piece-wise linear functions, in this section we will go over how to construct the piece-wise linear function in to quantum computer.

First of all we need to transform the payoff function $F(\mathbf{X})$ in to new functions $f_b(i) = m_b \cdot i + o_b$, with m_b the slope of the function and o_b the offset of the function. For every break point b of the payoff function we need to create a new function f_b. The new function gets the following as input $i \in \{0, \ldots, 2^n - 1\}$, which represents all states the qubits can have. For example we have three qubits ($i \in \{0, 1, 2, 3, 4, 5, 6, 7\}$) so $|011\rangle$ mapes to $i = 3$. The output of the function can also only be $f(i) \in [0, 1]$ this was shown in section 3.2.2. Now that we build the function we can insert the function in this equation 16 and get this new equation

$$|i\rangle_n |0\rangle = |i\rangle_n (\cos[f(i)] |0\rangle + \sin[f(i)] |1\rangle) \tag{21}$$

So this function can be rebuild in a quantum computer with controlled Y-Rotations CR_y and normal Y-Rotations R_y, the normal Y-Rotation is only used to initially rotate the qubit, which corresponds to the first offset. To do this we need an extra qubit on which we perform the rotations based on x encoded in register $|i\rangle_n$ which is already holding the probability distribution. Similar to the Amplitude Estimation the rotations add a factor 2^j, where the j's are qubits from $|i\rangle_n$-register. (shown in figure 4) Consequently the more

$|j\rangle$ are set to $|1\rangle$, the higher the value of x is. If we add more qubits to $|i\rangle_n$-register, i.e. more grid-points, a higher accuracy can be reached.

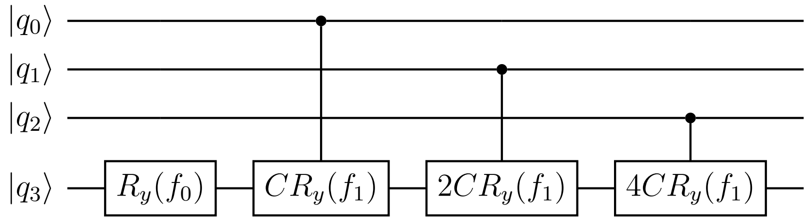


Figure 4: Quantum circuit for a linear function. Base image copied from [5]

Until now we only showed how to implement a linear function but every piece-wise linear function applies only on specific states. Now we have to build a greater or equal operation in a quantum computer, since we are dealing with a *max* function in eq. 25. For this we need also n qubits, one of the qubits has the result at the end. So for this operation we have n data qubits $|q\rangle$ and n qubits for the greater or equal operation $|a\rangle$. To build the operation we can follow this steps (example of a circuit is in figure 6).

1. At first we convert the breakpoint $b \in [0, 2^n - 1]$ in a bit representation t. For this we use a ceil operation on b and convert it then in a bit representation t. Every bit represent one qubit. If the breakpoint lies between two grid points i, the ceil operation ensures that the respective larger grid point is used as breakpoint for controlled application of max-operation.
2. Now we can iterate over all the qubits. (we use here j as iterate parameter)
   - if it is the first qubit $j = 0$ and $t[j] = 1$, we have to perform a CX operation on $|q_j\rangle$ and $|a_j\rangle$.
   - if $t[j] = 1$ an OR-Operation has to be done on $|q_j\rangle$ and $|a_{j-1}\rangle$ the result of the OR-Operation will be saved in $|a_j\rangle$
   - if $t[j] = 0$ an CCX-Operation has to be done on $|q_j\rangle$, $|a_{j-1}\rangle$ and $|a_j\rangle$
3. Now we have in a_{n-1} the result saved $|0\rangle$ if $i < b$ and $|1\rangle$ if $i \geq b$.
4. What we have to do now is to reverse the process except that now $j \in n-1, \ldots, 0$ is. So we do not consider the last qubit because it saves the result. To reverse an CX, CCX and OR-Operation we only have to apply the same operator again.

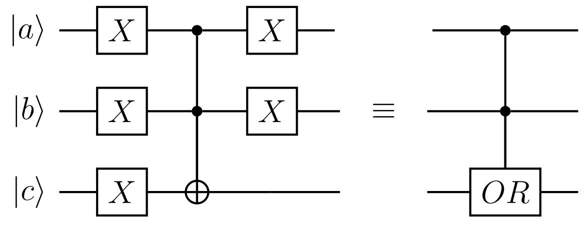


Figure 5: Example of an OR-Operation on qubit $|a\rangle$ and $|b\rangle$. The result is then saved in $|c\rangle$. Base image copied from [5]

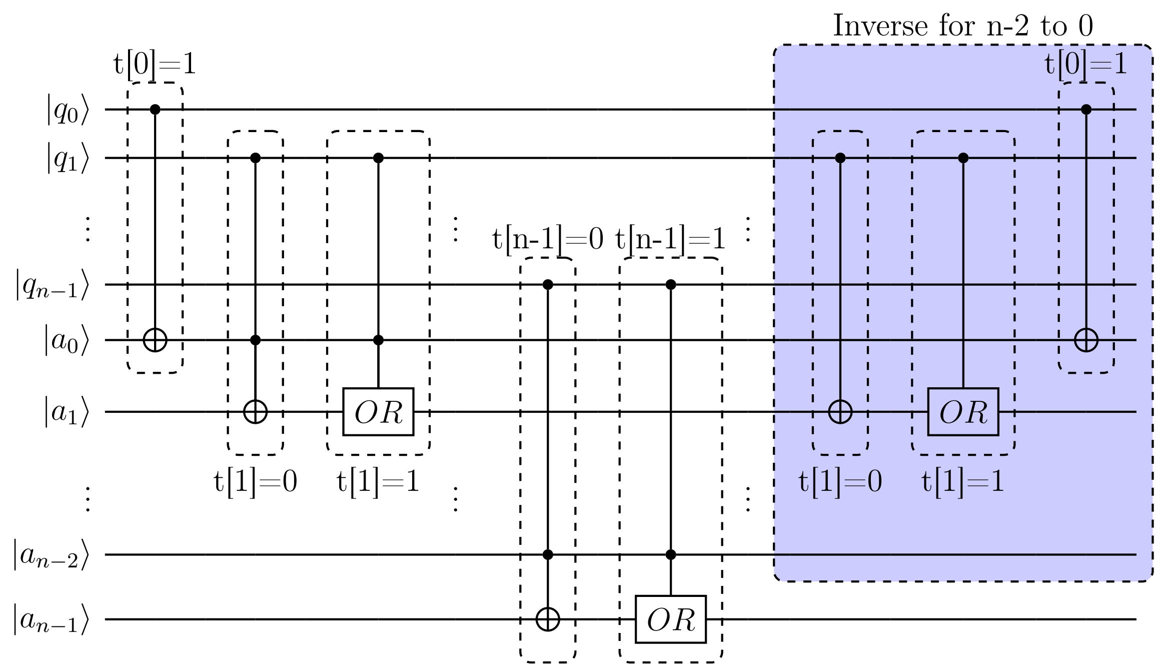


Figure 6: Basic structure of an circuit for the bigger then operation. Base image copied from [5]

So now we have a qubit a_{n-1} (control qubit) which holds the information if $i \geq b$. This control qubit is then used to control if the rotations of breakpoint b are executed or not. After applying the Y Rotations for a breakpoint b the inverse of the greater or equal operation has to applied to the circuit, so that the states are all $|0\rangle$ for the next breakpoint. Note that a smaller-than operations can be realized by adding a bit-flip X-gate to qubit a_{n-1}.

Now that we have shown how to construct a piece-wise linear function in a quantum computer we change θ_P again to better obtain our desired result $\mathbb{E}$.

$$\theta_P = c\tilde{f}_b(i) + \frac{\pi}{4} \tag{22}$$

$$\tilde{f}_b(i) = 2\frac{f_b(i) - f_{\min_b}}{f_{\max_b} - f_{\min_b}} \tag{23}$$

Equation 22 is chosen to center $\sin^2(\theta_P)$ around $\tilde{f}_b(i) = 0$ and 23 so that $\tilde{f}_b(i) \in [-1, 1]$

At the end of the Amplitude Estimation we only have to calculate the result. To obtain an approximation for θ_p, AE applies Quantum Phase Estimation to approximate certain eigenvalues of $\mathcal{Q}$, which are classically mapped to an estimator for a. The Quantum Phase Estimation uses m additional sampling qubits to represent result and $M = 2^m$ applications of $\mathcal{Q}$, i.e., M quantum samples. The m qubits, initialized to an equal superposition state by Hadamard gates, are used to control different powers of $\mathcal{Q}$. After applying an inverse Quantum Fourier Transform, their state is measured resulting in an integer $y \in \{0, ..., M-1\}$, which is classically mapped to the estimator for a, i.e.

$$\tilde{a} = \sin^2(y\pi/M) \in [0, 1]. \tag{24}$$

4 Results

In this section we go over our results of the European Call Option and compared our result from the quantum computer to the classical result. For the Images we used this parameters $S_0 = 2, \sigma = 0.4, r = 0.05, T = 40/365, K = 2, n = 3, c = 0.25$. n is the number of qubits.

The European call option only depends on the price of the stock at the expiration date so we can construct the probability of the value with a log-normal distribution (described in section 3.1). Qiskit has already a function to create a log-normal distribution and to load the distribution into a quantum computer. The function needs μ, σ_{dist} and lower and higher bounds to create the log-normal distribution. The result of the log-normal distribution can be seen in figure 7.

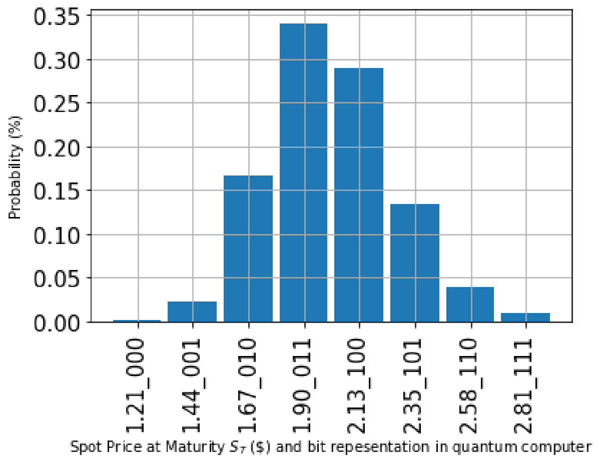


Figure 7: The distinct log-normal distribution (eq. 10), with $\mu = (r - 0.5 \cdot \sigma^2) \cdot T + \log(S_0)$, $\sigma_{\text{dist}} = \sigma \cdot \sqrt{T}$ and as lower l and highest h value we used the 3 standard deviation. As x value the classical value is shown and the corresponding qubit state which are representing the number in our quantum computer.

The equation of the European Payoff function is this

$$F(S(T)) = \max\{0, S(T) - K\} \tag{25}$$

and creates the result shown in figure 8.

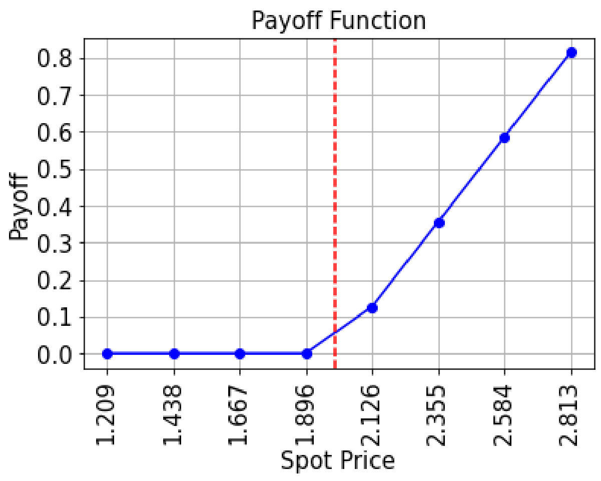


Figure 8: Here the value of the payoff function is shown. The breakpoints are set to $[0, 2.126]$ or $[|000\rangle, |100\rangle]$ thanks to ceil operation described in 3.2.3

With this values we created our breakpoints, slopes and offsets. Important to know is that for our slope at breakpoint 1 we have to subtract slope(0), because the qubit already did a rotation for slope(0), the same for the offset. So we get this breakpoints, slopes and offsets, where we used $b_{\text{classic}} = [0, S_T]$

$$\begin{aligned}
br &= \frac{b_{\text{classic}} - l}{h - l} \cdot (2^n - 1) \\
&= [0.0, 3.45206590600975] \\
\text{slopes}_{\text{temp}} &= [0, \pi c \frac{h - l}{2 \cdot (f(i)_{\max} - f(i)_{\min}) \cdot (2^n - 1)}] \\
\text{slopes}(b) &= \text{slopes}_{\text{temp}}(b) - \sum_{i=0}^{b-1} \text{slopes}(i) \\
&= [0.0, 0.22136774] \\
\text{offsets}_{\text{temp}} &= -\frac{\pi c^2}{2 \cdot (f(i)_{\max} - f(i)_{\min})} \\
\text{offsets}(b) &= \text{offsets}_{\text{temp}} - \text{slopes}_{\text{temp}}(b) \cdot br(b) - \sum_{i=0}^{b-1} \text{offsets}(i) \\
&= [1.17809725, -0.76417604]
\end{aligned}$$

This values then result in the circuit displayed in figure 9 and a probability distribution shown in figure 10

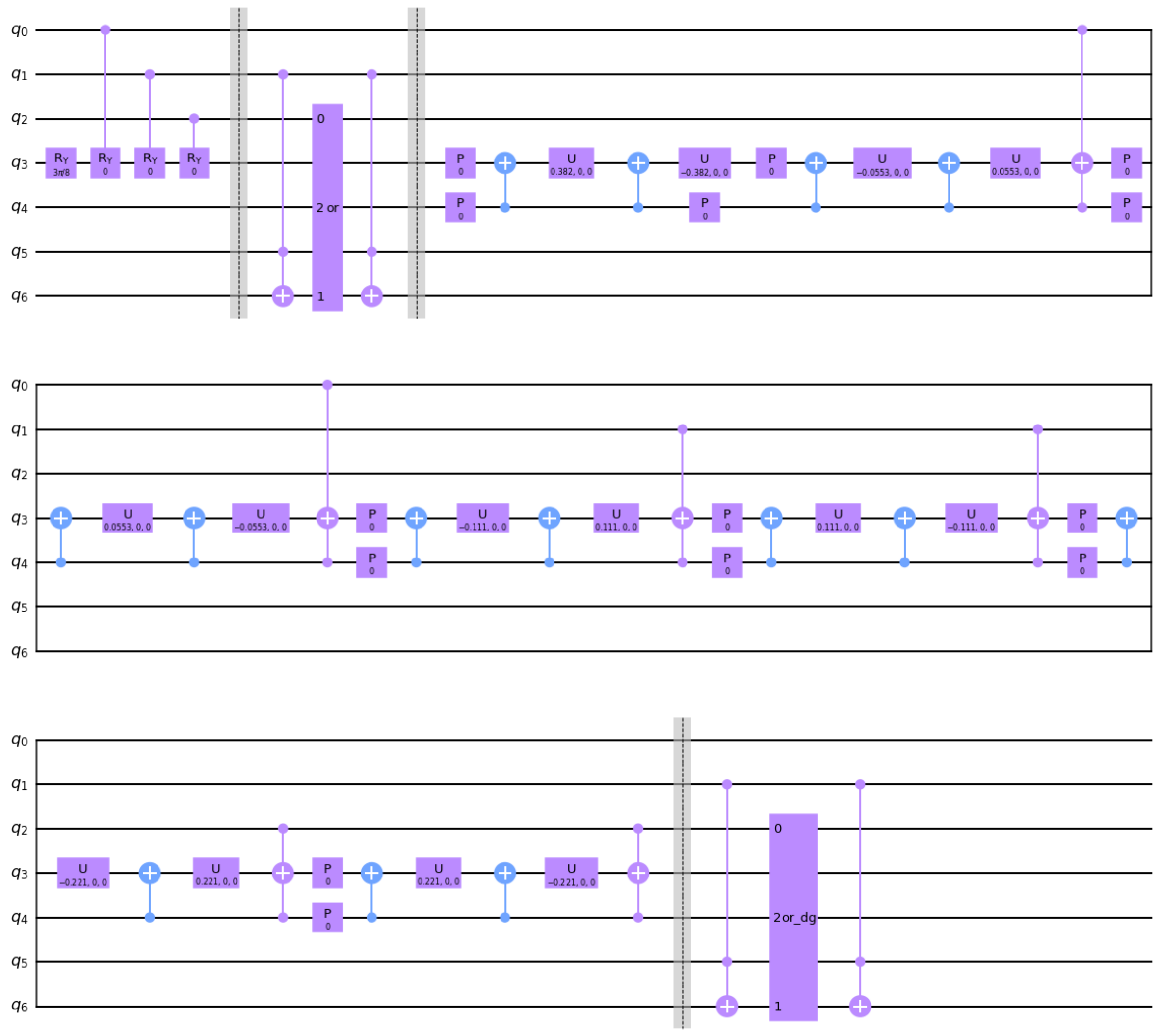


Figure 9: The output of the circuit for the payoff function. The image of the circuit was generated by qiskit.[1]. The circuit is divided in four sections. The first section is the Y-Rotation for the first breakpoint. The second section is then the greater or equal operation for the second breakpoint. The next section is then the rotation for the second breakpoint, here it good to see that the compare qubit is used to decide if the rotation should be executed or not. The last section is now the inverse of the second section.

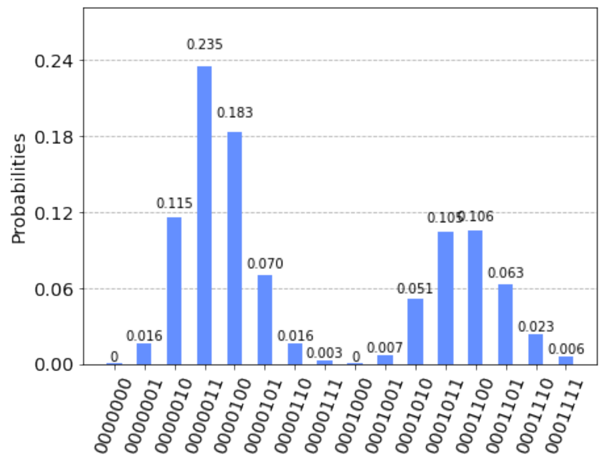


Figure 10: The output after applying the payoff function to our circuit.

The Grover Operator $\mathcal{Q} = AS_0A^\dagger S_{\psi_1}$ can now be used for the Phase Estimation Part of Amplitude Estimation.

For European Call Options the A-gate of the Grover Operator is the circuit shown in figure 9, $A^\dagger$ is simply the complex conjugate of A. S_0 is realized by a bit-flip of the objective qubit sandwiched by H-gates. The objective qubit is qubit 3 in our case, since thanks to the qubit saving implementation of *max* function in 6 a second ancilla qubit for A-gate is not needed.$S_{\psi_1} = 1 - 2\left|\psi_1\right>\left|0\right>\left<\psi_1\right|\left<0\right|$ is implemented using $2\left|\psi_1\right>\left|0\right>\left<\psi_1\right|\left<0\right| - 1$ and a global phase bit, since the negative form can easily be implemented by using multi-controlled Z-gates sandwiched by X-gates on the target qubit. The global phase has no effect on Grovers algorithm in general. The Grover is shown in figure 11

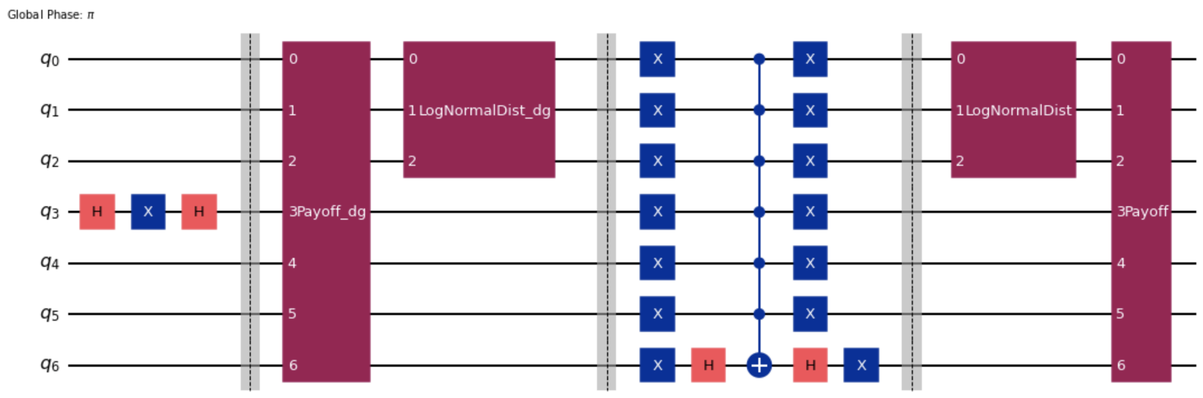


Figure 11: The Grover-operator. S_0 is a bit-flip implemented by a X-gate sandwiched by H-gates. A, $A^\dagger$ is the circuit and its complex-conjugate shown in figure 9. $S_{\psi_1} = 1 - 2\left|\psi_1\right>\left|0\right>\left<\psi_1\right|\left<0\right|$ is implemented using $2\left|\psi_1\right>\left|0\right>\left<\psi_1\right|\left<0\right| - 1$ and a global phase bit.

The Grover-operator can than be applied to the first register $\left|i\right>$ controlled by a second evaluation register $\left|m\right>$. See figure 2 for a schematic overview.

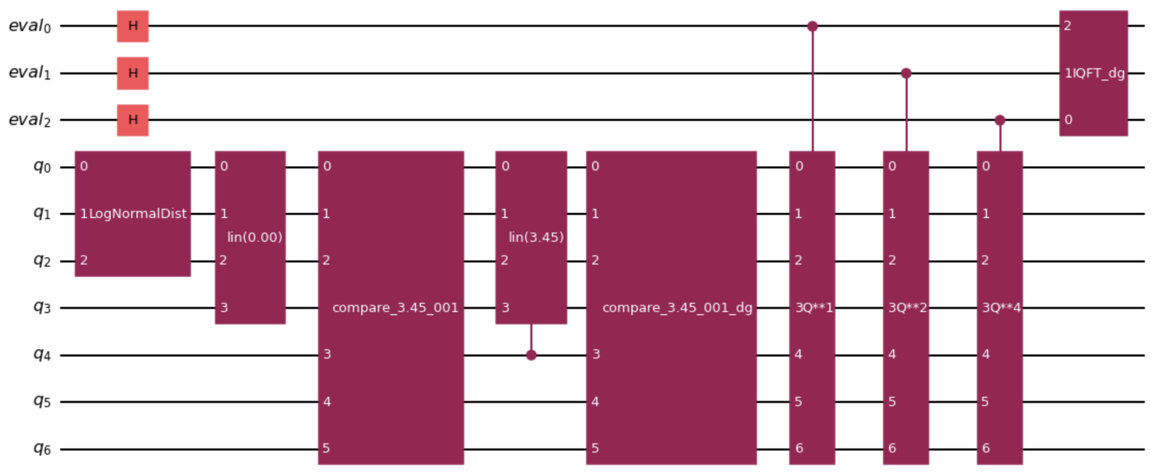


Figure 12: The combined circuit. Note that the objective qubit here is qubit 3. Evaluation qubits are chosen for pragmatical reasons. For results a register consisting of seven qubits have been used.

The combined Circuit is displayed in figure 12. For the obtained results, i.e. the estimate of Amplitude Estimation seven qubits in the evaluation register have been used. Measuring all qubits in the evaluation register gives the following result:

$$\text{Exact Value:} \qquad 0.1133 \tag{26}$$

$$\text{Estimated Value:} \qquad 0.1061 \tag{27}$$

$$\text{95 Percent Confidence interval:} \qquad [0.1157, 0.1209] \tag{28}$$

Therefore the estimated value is close to the exact value. One problem here is that the estimated value does not lie within the confidence interval. Note that the estimated value is the mean of applying eq. 24 to the results of amplitude estimation. Additionally a post processing has been added in order to map from $[0, 1]$ to the domain of stock prices. It is possible to apply a maximum likelihood estimator to the estimated value:

$$\text{MLE estimator value: } 0.1189. \tag{29}$$

The value of the MLE estimator vaue is slightly better than the value of the ordinary estimator. Furthermore it lies within the confidence interval which is not the case for the estimated value.

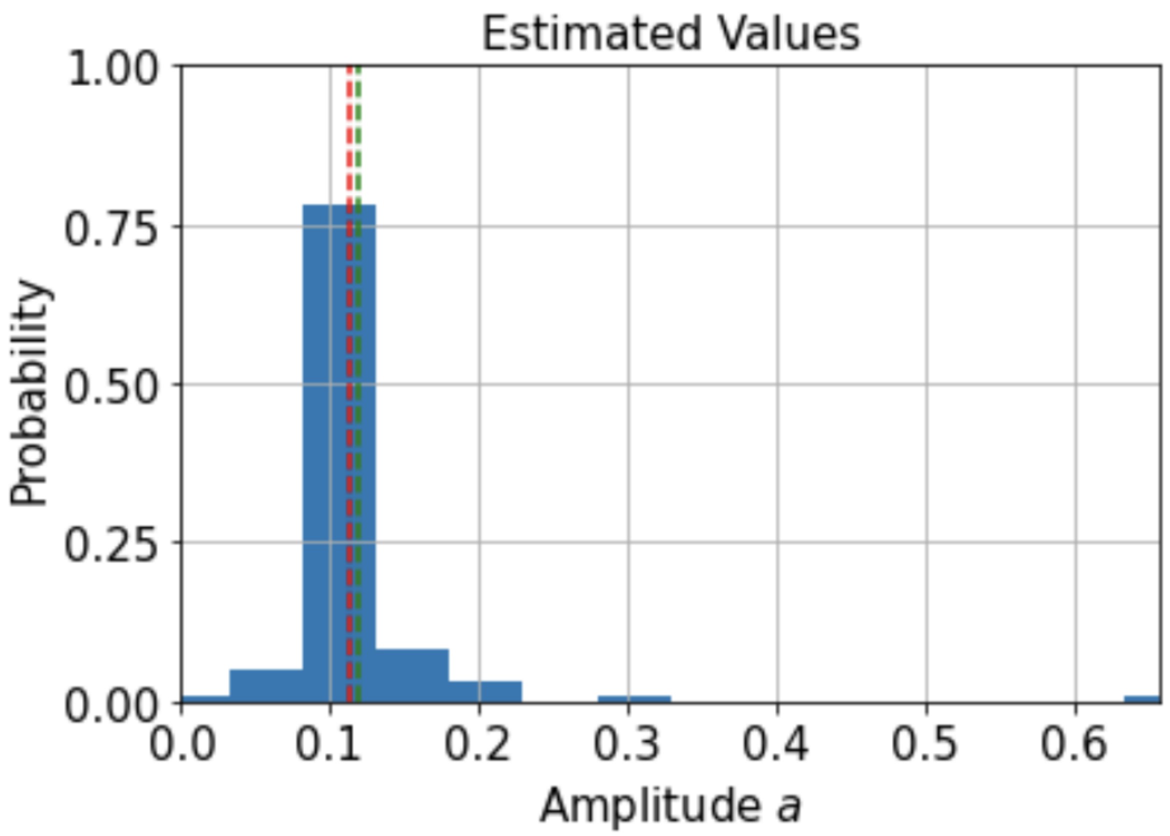


Figure 13: Results of Canonical AE. Red dotted line show the exact value. Green line is the MLE estimator value.

When we now execute the same Circuit using Iterative Amplitude Estimation [2] from qiskit we get the following results:

$$\text{Exact Value:} \quad 0.1133 \tag{30}$$
$$\text{Estimated Value:} \quad 0.1203 \tag{31}$$
$$\text{95 Percent Confidence interval:} \quad [0.1153, 0.1254] \tag{32}$$

Which is comparable to the values from Canonical Amplitude Estimation but comes with a lower cost, since Iterative Amplitude Estimation estimates the amplitude wihtout the usage of Phase Estimation and therefore with less qubits and gates.

5 Outlook and Conclusion

In this report, we worked on reproducing the work of the paper [5]. When we compare our result we get with the quantum computer and compare it with the result of the exact value we are close to the exact result. But do not fit into the confidence interval unless a MLE estimation is used.
After investigating these problems we can apply Amplitude Estimation to even more complicated options types like path dependent once. Also generic functions can be fitted with piece-wise linear function and more research can be done on how to use the fitted linear functions in Amplitude Estimation.

References

[1] M. S. et al. Qiskit: An open-source framework for quantum computing, 2021.

[2] D. Grinko, J. Gacon, C. Zoufal, and S. Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7(1), mar 2021. doi: 10.1038/s41534-021-00379-1. URL `https://doi.org/10.1038%2Fs41534-021-00379-1`.

[3] A. Montanaro. Quantum speedup of monte carlo methods. *pre-print*, 2015. doi: 10.1098/rspa.2015.0301.

[4] P. Rebentrost, B. Gupt, and T. R. Bromley. Quantum computational finance: Monte carlo pricing of financial derivatives. *pre-print*, 2018. doi: 10.1103/PhysRevA.98.022321.

[5] N. Stamatopoulos, D. J. Egger, Y. Sun, C. Zoufal, R. Iten, N. Shen, and S. Woerner. Option pricing using quantum computers. *pre-print*, 2019. doi: 10.22331/q-2020-07-06-291.

[6] S. A. Stein, B. Baheri, D. Chen, Y. Mao, Q. Guan, A. Li, B. Fang, and S. Xu. Qugan: A generative adversarial network through quantum states, 2020.