Option Pricing using Canonical Amplitude Estimation

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

October 5, 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 **??** 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 **??**.

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.

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 **??**. 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

$$dS_t = S_t r dt + S_t \sigma dW_t, \tag{1}$$

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

$$dB_t = B_t r dt, \tag{2}$$

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

$$S_i = S_0 \cdot \exp^{\sigma \cdot W_i + (r - \frac{\sigma^2}{2})T} \tag{3}$$
$$W_i = \mathcal{N}(\mu = 0, \sigma^2 = T). \tag{4}$$

The risk-free asset is solved easily as

$$B_t = e^{rt}. \tag{5}$$

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

$$f(S_T) = \max(0, S_T - K). \tag{6}$$

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

$$\Pi = e^{-rt}\mathbb{E}[f(S_T)], \tag{7}$$

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 ([**?**])

$$P(S_T) = \frac{1}{S_T\sigma\sqrt{2\pi T}} \exp\left(-\frac{(\ln S_T - \mu)^2}{2\sigma^2 T}\right), \tag{8}$$

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 ([**?**]):

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:

$$\mathbb{E}_P[F(\mathbf{X})] = \frac{1}{M}\sum_{i=1}^{M} F(\mathbf{X}_i) \tag{9}$$

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 **??**.

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.

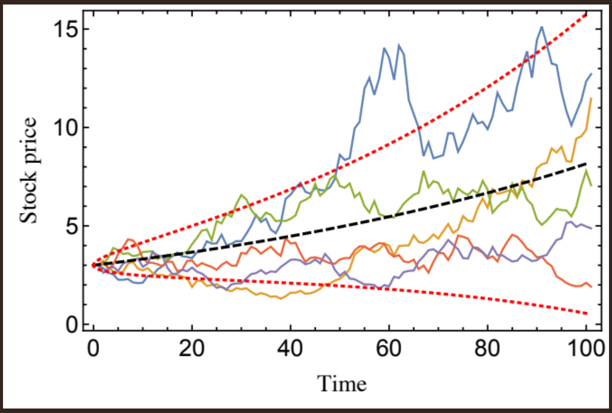

Figure 1: Image of five different paths of the stock price. The stock price is in dollar and the time is in days. The dashed black line is the mean of the samples and the dotted red lines are the first standard deviation. The parameters are: $S_0 = \$3, r = 0.1, \sigma = 0.25$ [?]

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 ([?]).

$$P[|\Pi - \hat{\Pi}| \leq \epsilon] \leq \frac{\lambda^2}{M\epsilon^2} \tag{10}$$

So to achieve an error of ϵ

$$M = \mathcal{O}(\frac{\lambda^2}{\epsilon^2}) \tag{11}$$

samples are needed. Section **??** 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

$$F(\mathbf{X}_i) = \max\{0, X_i - K\} \tag{12}$$

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

Quantum Amplitude Estimation provides a quadratic speed-up over classical monte carlo methods.
Suppose a unitary operator A is acting on a register of $(n+1)$ qubits

$$A\left|0\right\rangle_{n+1} = \sqrt{1-a}\left|\psi_0\right\rangle_n\left|0\right\rangle + \sqrt{a}\left|\psi_1\right\rangle_n\left|1\right\rangle, \tag{13}$$

for normalised states $\left|\psi_0\right\rangle_n\left|0\right\rangle$, $\left|\psi_1\right\rangle_n\left|1\right\rangle$ and unknown $a \in [0,1]$. QAE allows the efficient estimation of a, i.e. the probability of measuring $\left|1\right\rangle$ in the last qubit.
Simply measuring $\left|1\right\rangle$ t times does not give any advantage since the variance of t is defined by a bernoulli distibution

$$t = O(\frac{a(1-a)}{\epsilon^2}), \tag{14}$$

with given accuracy ϵ.
To gain a quantum speed-up more efficient quantum algorithms are used than simply measuring t times. From **??** it can be seen that

$$a = sin^2(\theta_a), \tag{15}$$

what comes from the fact that exchanging a and $1-a$ by sin and cos would result in same quantum behavior:

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

The first efficient quantum algorthim used is Grovers algorithm, where

$$\begin{aligned} Q &= AS_0AS_{\psi_0}\\ S_0 &= 1 - 2\left|0\right\rangle\left\langle 0\right|\\ S_{\psi_0} &= 1 - 2\left|\psi_0\right\rangle\left|0\right\rangle\left\langle\psi_0\right|\left\langle 0\right|. \end{aligned} \tag{17}$$

Q applies a rotation of angle $2\theta_a$ in the complex two-dimensional Hilbert space spannend by $\left|\psi_0\right\rangle\left|0\right\rangle$ and $\left|\psi_1\right\rangle\left|0\right\rangle$. The eigenvalues of Q are (Euler formula) $\exp(\pm i\theta_a)$.
The Grover algorithm therefore is used to encode the angle θ_a as the argument of an exponential function in the quantum register but does not yield an approximation of that angle. To obtain an approximation for θ_a QAE applies Quantum Phase Estimation(QPE) to approximate the eigenvalues of Q. QPE uses m additional sampling qubits to represent the results and $M = 2^m$ applications of Q.
For that the m qubits are initialized to a equal superposition by Hadamard gates and are then used to control the different powers of Q applied to the QAE register. After application of an inverse Quantum Fourier Transformation (QFT) the state of QAE register is measured, resulting in $y \in 0, ..., M-1$ which is classically mapped to the estimator of a

$$\tilde{a} = \sin^2(y\pi/M) \in [0,1]. \tag{18}$$

Focusing on the efficiency of the algorithm described above it can be observed from **??** and **??** that $\tilde{a}$ satisfies:

$$|-\leq (\frac{\sqrt{\pi}}{M} + \frac{\pi}{M^2}) = O(M^-1), \tag{19}$$

with probability of at least $(\frac{8}{\pi^2})$. Result **??** represents a quadratic speed-up in comparision to the classical efficiency **??**.

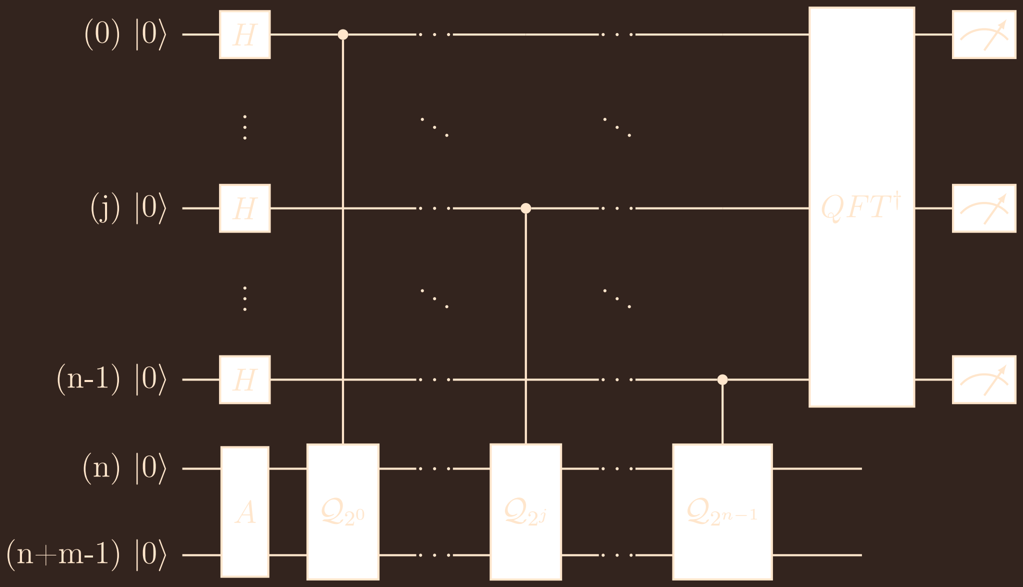

Figure 2: Design of the Amplitude Estimation recreated after [**?**]

3.2.1 European Option Pricing

For option contracts the involved random variables represent the possible values $\{S_i\}$ for $i \in \{0, .., 2^n - 1\}$ the underlying asset can take and the probabilities p_i that those values will be realized. For the unitary A that means: Given an n+1-qubit register N (QAE register), asset prices at maturity S_T, the corresponding probabilites $\{p_i\}$ and the option payoff $f(S_i)$ the operator A becomes

$$A = \sum_{i=0}^{2n+1} \big(\sqrt{1 - f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|0\rangle + \sqrt{f(S_i)}\sqrt{p_i}\,|S_i\rangle\,|1\rangle\big). \tag{20}$$

Comparing **??** and **??** gives for options

$$a = \sum_{i=0}^{2n+1} f(S_i)p_i = E[f(S_T)]. \tag{21}$$

?? is the expectation value of the payoff given the stockprice at maturity and therefore the quantity which we want to approximated with Monte Carlo Simulations. For option pricing using QAE that means that in order to be able to apply QAE the unitary operator A has to be created using quantum circuits, i.e. a efficient method to encode the probabilities p_i for a certain stock price at maturity S_i and the corresponding payoff $f(S_i)$ in the register N.

3.2.2 Loading the distribution

The algorithm for loading the probability distribution ?? reads:

- On an interval, which is described by the minimal and maximal asset price an equidistant discretization is established based on the number of quantum states $M = 2^n - 1$.
- Equidistant x-values are used to compute log-normal probability of the asset prices ?? with constant volatility.
- The probabilites are normalized by the sum of all probabilites
- The square root is applied and resulting numbers initialized in a quantum state.

The algorithm for the distribution loading can then be implemented like:

```
def logNormalDistribution(num_qubits, mu, sigma, bounds: tuple, name):
  qc = QuantumCircuit(num_qubits, name=name)
  x = np.linspace(bounds[0], bounds[1], num=2**num_qubits)
  probabilities = []
  for x_i in x:
      # map probabilities from normal to log-normal reference:
      #
          https://stats.stackexchange.com/questions/214997/multivariate-log-normal-probabilt
      if np.min(x_i) > 0:
          det = 1 / np.prod(x_i)
          probability = multivariate_normal.pdf(np.log(x_i), mu, sigma) * det
      else:
          probability = 0
      probabilities += [probability]
      normalized_probabilities = probabilities / np.sum(probabilities)

  initialize = Initialize(np.sqrt(normalized_probabilities))
  circuit = initialize.gates_to_uncompute().inverse()
  qc.compose(circuit, inplace=True)
  return qc, x, probabilities
```

In tearms of a quantum state the above algorithm implements

$$|\Phi_n\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i}\, |S_i\rangle_n \,. \tag{22}$$

3.2.3 Constructing the payoff

The payoff of a european call option at maturity is defined as

$$f(S_T) = \max\{0, S_T - K\}, \tag{23}$$

with K the strike price and T the maturity date. Numerically the payoff (and any other function) can be realized using piecewise linear functions for $S_T < K$ and $S_T \geq K$ which are dependent on the given stock price S_i. On a quantum computer we are interested in an operator

$$f(i) = \alpha i + \beta \tag{24}$$
$$|i\rangle_n |0\rangle \to |i\rangle_n (\cos[f(i)] |0\rangle + \sin[f(i) |1\rangle]), \tag{25}$$

where $f : \{0, ..., 2^n - 1\} \to [0, 1]$. f is on a quantum computer dependent on the stock price encoded in a quantum state and the piecewise linear functions are implemented usind controlled Y-gates.
?? uses therefore the strategy introduced already in equation **??** to obtain the operator A defined in **??**. $f(i)$ is implemented by using each qubit i of the $|i\rangle_n$(N) register as a control for the Y-rotation with angle $2^i\alpha$ of an ancilla qubit. The constant term is implemented by an initial rotation on the ancilla without controls.

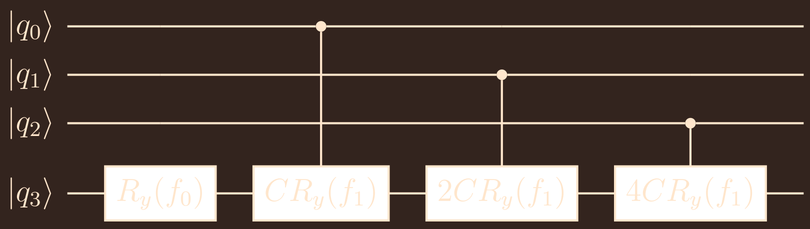

Figure 3: Quantum circuit that creates state in Eq. [**??**]. Three qubits $|i_2 i_1 i_0\rangle$ are here used to be encode $i = 4i_2 + 2i_1 + i_0 \in \{0, ..., 7\}$. The linear function $f(i) = \alpha \cdot i + \beta$ is given by $4\alpha \cdot i_2 + 2\alpha \cdot i_1 + \alpha \cdot i_0 + \beta$.

3.2.4 Implementation of max-function

3.2.5 Implementation of piece-wise linear functions

Implementing the piecewise linear function is not directly straigth forward so a detailed introduction is given in the following. As mentioned above we are dealing with linear function to encode the payoff

$$f(x) = \alpha x + \beta, \tag{26}$$

where $f(x)$ maps x to the payoff by equation **??** and the domain of x consist of of two linear function. One for $S_T \leq K$ and one for $S_T \geq K$. Furthermore the S_i do not occure as a real number but as an element of $\{0, ..., 2^n - 1\}$ and additionally on a quantum computer $f(x)$ has to be realised using rotations in the corresponding Hilbert space. These challenges require us to use a affine mapping $f\hat{(x)}$ which is in contrast to

$$f : [a, b] \to [c, d] \tag{27}$$

defined as

$$\hat{f} : \{0, ..., 2^n - 1\} \to [0, 1]. \tag{28}$$

Note that in this case $d = b - K$ thanks to the definition of the payoff. But how is the transformation from **??** to **??** actually done?
For piecewise function there are of course breakpoints needed within $\{0, ..., 2^n - 1\}$

$$b_p(x) = \frac{x - a}{b - a} \times 2^n - 1. \tag{29}$$

The affine transformation from f to $hatf$ is then

$$\phi(x) = a + \frac{b-a}{2^n - 1} \cdot x \tag{30}$$

$$\hat{f}(x) = \frac{f(\phi(x)) - c}{d - c}. \tag{31}$$

Implemented is $\phi(x)$ first by leveraging on the slope and offset of a linear function

$$\hat{\alpha} = \frac{\alpha}{2^n - 1} \cdot (b - a) \tag{32}$$

$$\hat{\beta} = \beta \tag{33}$$

$$\Rightarrow \frac{\alpha}{2^n - 1} \cdot (b - a) + \beta. \tag{34}$$

$\hat{f}(x)$ is then generated by introducing the slope and offset angles

$$\theta_{\hat{\alpha}} = \frac{\pi}{2} f_c \frac{\alpha}{2^n - 1} \frac{(b-a)}{(d-c)} \tag{35}$$

$$\theta_{\hat{\beta}} = \frac{\pi}{4}(1 - f_c) + \frac{\pi}{2} \frac{\beta - c}{d - c} f_c, \tag{36}$$

where $\pi/2$ is used to map from slope and offest to angles.The reason why f_c occurs has to be discussed in more detail. Equation **??** can be used to create the operator that maps $\sum_i \sqrt{p_i} \left|i\right\rangle_n \left|0\right\rangle$ to

$$\sum_{i=0}^{2^n - 1} \sqrt{p_i} \left|i\right\rangle_n \left[\cos(f_c \cdot \hat{f}(i) + \frac{\pi}{4}) \left|0\right\rangle + \sin(f_c \cdot \hat{f}(i) + \frac{\pi}{4}) \left|1\right\rangle\right], \tag{37}$$

with $f_c \in [0, 1]$. Consequently , $\sin^2 f_c \cdot \hat{f}(i) + \pi/4 isananti-symmetricfunctionaround 1/2. With(i)nowbe$
$\sum_{i=0}^{2^n-1} p_i \sin^2(f_c \cdot \hat{f}(i) + \frac{\pi}{4}),$ (38)is well approximated by

$$P_1 \approx \sum_{i=0}^{2^n - 1} p_i \big(f_c \cdot \hat{f}(i) + 1/2\big) = f_c E f(x)$$

- c$\frac{}{d-c-f_c+\frac{1}{2}.}$The approximation is obtained by making use of the approximation

$$\sin^2(c\hat{f}(i) + \frac{\pi}{4}) = f_c \hat{f}(i) + \frac{1}{2} + \mathcal{O}(f_c^3 \hat{f}^3(i)), \tag{40}$$

which is valid for small values of $f_c \hat{f}(i)$.

1. The payoff function $F(\mathbf{X})$ has to be build with qubits inside the quantum computer. In section **??** the payoff function is constructed.

2. 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 **??**.

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 **??**

3.2.6 Amplitude Estimation

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

$$A\left|0\right\rangle_{n+1} = \sqrt{1-a}\left|\psi_0\right\rangle_n\left|0\right\rangle + \sqrt{a}\left|\psi_1\right\rangle_n\left|1\right\rangle. \tag{41}$$

In the equation **??** $\left|\psi_0\right\rangle_n$ 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 **??** turns into this

$$A\left|0\right\rangle_{n+1} = \cos(\theta_a)\left|\psi_0\right\rangle_n\left|0\right\rangle + \sin(\theta_a)\left|\psi_1\right\rangle_n\left|1\right\rangle. \tag{42}$$

$$\theta_p = 2\theta_a$$

$$A = R_Y(\theta_p) \tag{43}$$

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:

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

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

$$\mathcal{Q} = R_Y(2\theta_p) \tag{44}$$

$$\mathcal{Q}_{2^j} = R_Y(2^j \cdot 2\theta_p) \tag{45}$$

The error ϵ of this method defined as followed

$$|\hat{\Pi} - \Pi| \leq C \cdot (\frac{\sqrt{\mathbb{E}}}{M} + \frac{1}{M^2})$$

$$M = \mathcal{O}(\frac{1}{\epsilon\sqrt{\mathbb{E}}}) \tag{46}$$

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

3.2.7 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 $\left|011\right\rangle$ mapes to $i = 3$. The output of the function can also only be $f(i) \in [0,1]$ this was shown in section **??**. Now that we build the function we can insert the function in this equation **??** and get this new equation

$$\left|i\right\rangle_n\left|0\right\rangle = \left|i\right\rangle_n(\cos[f(i)]\left|0\right\rangle + \sin[f(i)]\left|1\right\rangle) \tag{47}$$

So this function can be rebuild with a quantum computer using 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 **??**) 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.

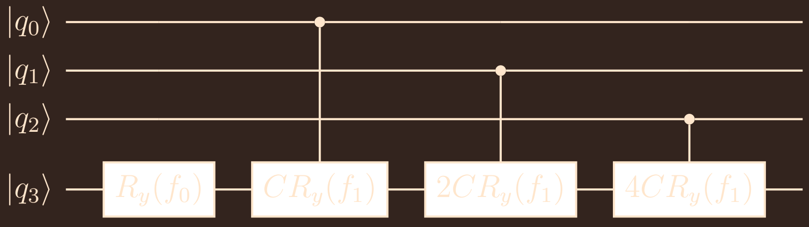

Figure 4: Quantum circuit for a linear function. Base image copied from [**?**]

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. **??**. 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 **??**).

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 $j \in n - 1, \dots, 0$ 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.

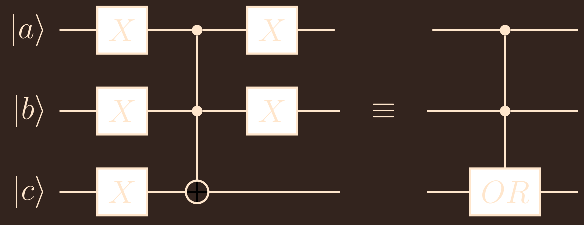

Figure 5: Example of an OR-Operation on qubit $|a\rangle$ and $|b\rangle$. The result is then saved in $|c\rangle$. Base image copied from [?]

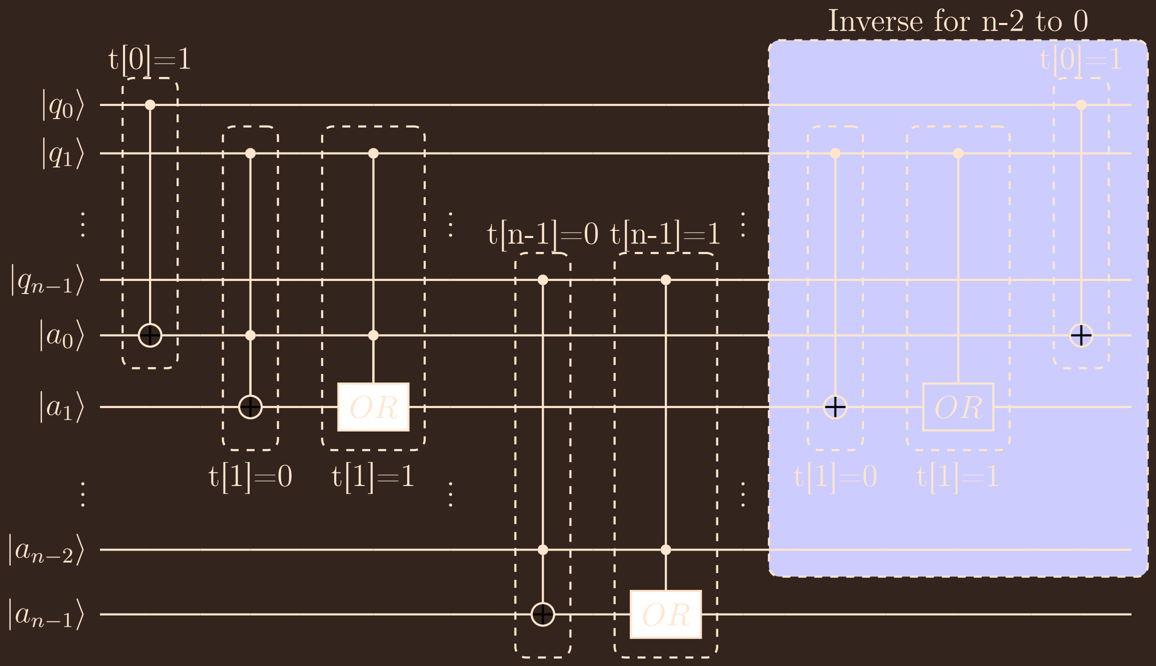

Figure 6: Basic structure of an circuit for the bigger then operation. Base image copied from [?]

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 $\mathbb{E}$.

$$\theta_P = c\tilde{f}_b(i) + \frac{\pi}{4} \tag{48}$$

$$\tilde{f}_b(i) = 2\frac{f_b(i) - f_{\min_b}}{f_{\max_b} - f_{\min_b}} \tag{49}$$

Equation **??** is chosen to center $\sin^2(\theta_P)$ around $\tilde{f}_b(i) = 0$ and **??** 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.

$$\tilde{a} = \sin^2(y\pi/M) \in [0, 1]. \tag{50}$$