

# Option Pricing using Canonical Amplitude Estimation

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

October 12, 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 its 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.

## 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

$$dS_t = S_t r dt + S_t \sigma dW_t, \quad (1)$$

where  $r$  is the drift,  $\sigma$  the volatility and  $dW_t$  is a Brownian increment. The initial condition is  $S_0$ . In addition, the risk-free asset dynamics is given by

$$dB_t = B_t r dt, \quad (2)$$

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

$$S_i = S_0 \cdot \exp^{\sigma \cdot W_i + (r - \frac{\sigma^2}{2})T} \quad (3)$$

$$W_i = \mathcal{N}(\mu = 0, \sigma^2 = T). \quad (4)$$

The risk-free asset is solved easily as

$$B_t = e^{rt}. \quad (5)$$

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

$$f(S_T) = \max(0, S_T - K). \quad (6)$$

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

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

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 at maturity  $T$  follows a log-normal distribution with probability density ([5])

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

where  $\sigma$  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  $\mathbf{X} = \{X_1, X_2, \dots, X_N\}$ . 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  $\{\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_M\}$ .
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:

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

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 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  $\sigma$  and the drift  $r$  of the stock.

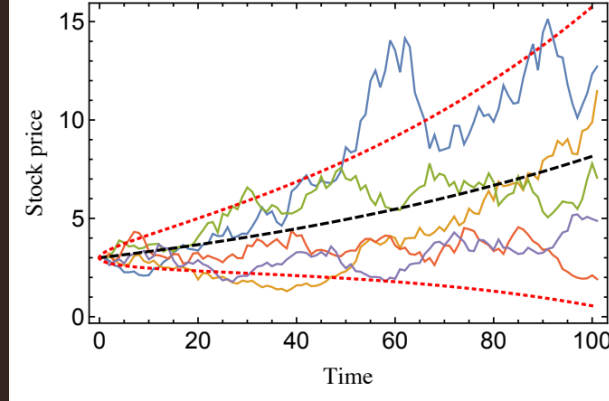


Figure 1: Image of five different paths of the stock price. The stock price is in dollar and the time is in days. The dashed black line is the mean of the samples and the dotted red lines are the first standard deviation. The parameters are:  $S_0 = \$3, r = 0.1, \sigma = 0.25$  [4]

Now  $\Pi$  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]).

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

So to achieve an error of  $\epsilon$

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

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

### 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

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

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

$$A|0\rangle_{n+1} = \sqrt{1-a}|\psi_0\rangle_n|0\rangle + \sqrt{a}|\psi_1\rangle_n|1\rangle, \quad (13)$$

for normalised states  $|\psi_0\rangle_n|0\rangle$ ,  $|\psi_1\rangle_n|1\rangle$  and unknown  $a \in [0, 1]$ . QAE allows the efficient estimation of  $a$ , i.e. the probability of measuring  $|1\rangle$  in the last qubit.

Simply measuring  $|1\rangle$   $t$  times does not give any advantage since the variance of  $t$  is defined by a bernoulli distribution

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

with given accuracy  $\epsilon$ .

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

$$a = \sin^2(\theta_a), \quad (15)$$

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

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

The first efficient quantum algorithm used is amplitude estimation which is the general case of Grovers algorithm, where

$$\begin{aligned}Q &= AS_0AS_{\psi_0} \\ S_0 &= 1 - 2|0\rangle\langle 0| \\ S_{\psi_0} &= 1 - 2|\psi_0\rangle\langle 0|\langle\psi_0|\langle 0|. \end{aligned} \quad (17)$$

$Q$  applies a rotation of angle  $2\theta_a$  in the complex two-dimensional Hilbert space spanned by  $|\psi_0\rangle|0\rangle$  and  $|\psi_1\rangle|0\rangle$ . The eigenvalues of  $Q$  are (Euler formula)  $\exp(\pm i\theta_a)$ .

The Grover algorithm therefore is used to encode the angle  $\theta_a$  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  $\theta_a$  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  $y \in 0, \dots, M - 1$  which is classically mapped to the estimator of  $a$

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

Focusing on the efficiency of the algorithm described above it can be observed from 11 and 19 that  $\tilde{a}$  satisfies:

$$|\tilde{a} - a| \leq \left(\frac{\sqrt{\pi}}{M} + \frac{\pi}{M^2}\right) = O(M^{-1}), \quad (19)$$

with probability of at least  $(\frac{8}{\pi^2})$ . Result 19 represents a quadratic speed-up in comparison to the classical efficiency 14.

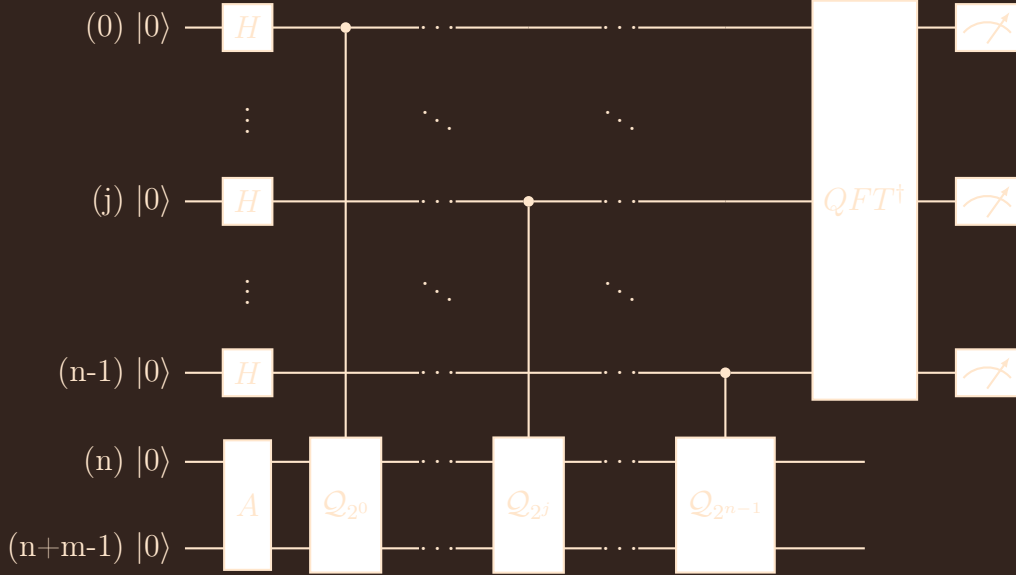


Figure 2: Design of the Amplitude Estimation. Operator  $A$  combines distribution loading and payoff construction as defined in 13.  $Q$  to rotate given state around  $2 * \theta_a$  in Hilbert-space spanned by  $|\phi_0\rangle |0\rangle$  and  $|\phi_1\rangle |1\rangle$ . Inverse QFT is then applied to obtain the estimate  $\tilde{a}$  of  $a$ .

### 3.2.1

As already mentioned amplitude amplification is the general case of Grover's algorithm. It amplifies the probability to measure a "good" part of an algorithm. This algorithm is in our case the circuit  $A$ . Suppose it initializes  $n$  qubits, such that

$$|U\rangle = A |0\rangle_n = \sqrt{1-a} |\phi_0\rangle_n + \sqrt{a} |\phi_1\rangle_n, \quad (20)$$

where  $|\phi_0\rangle, |\phi_1\rangle$  are both normalized  $n$ -qubit states and orthogonal to each other. For example, it could be that the last qubit of  $|\phi_0\rangle |0\rangle$  and  $|\phi_1\rangle |1\rangle$  is. Let us denote  $|\phi_1\rangle$  as the good-state, what means that the probability  $\sqrt{a}$  should be increased.

Furthermore,  $|\phi_0\rangle$  and  $|\phi_1\rangle$  can be viewed as the horizontal and vertical axes of a two-dimensional Hilbert space. The angle between  $|U\rangle$  and the horizontal axis is thus  $\theta_a = \arcsin(\sqrt{a})$ .

### 3.2.2 European Option Pricing

For option contracts, the involved random variables represent the possible values  $\{S_i\}$  for  $i \in \{0, \dots, 2^n - 1\}$  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 probabilities  $\{p_i\}$  and the option payoff  $f(S_i)$  the operator  $A$  becomes

$$A = \sum_{i=0}^{2n+1} (\sqrt{1 - f(S_i)}\sqrt{p_i} |S_i\rangle |0\rangle + \sqrt{f(S_i)}\sqrt{p_i} |S_i\rangle |1\rangle). \quad (21)$$

Comparing 21 and 13 gives for options

$$a = \sum_{i=0}^{2n+1} f(S_i)p_i = E[f(S_T)]. \quad (22)$$

21 is the expectation value of the payoff given the stockprice at maturity and therefore the quantity which we want to approximate 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  $N$ .

### 3.2.3 Loading the distribution

The algorithm for loading the probability distribution 8 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 8 with constant volatility.
- The probabilities are normalized by the sum of all probabilities
- 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-probability
        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 terms of a quantum state the above algorithm implements

$$|\Phi_n\rangle = \sum_{i=0}^{2^n-1} \sqrt{p_i} |S_i\rangle_n. \quad (23)$$

### 3.2.4 Constructing the payoff

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

$$f(S_T) = \max\{0, S_T - K\}, \quad (24)$$

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

$$f(i) = \alpha i + \beta \quad (25)$$

$$|i\rangle_n |0\rangle \rightarrow |i\rangle_n (\cos[f(i)] |0\rangle + \sin[f(i)] |1\rangle), \quad (26)$$

where  $f : \{0, \dots, 2^n - 1\} \rightarrow [0, 1]$ .  $f$  is on a quantum computer dependent on the stock price encoded in a quantum state and the piecewise linear functions are implemented using controlled Y-gates.

?? uses therefore the strategy introduced already in equation 18 to obtain the operator  $A$  defined in 21.  $f(i)$  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.

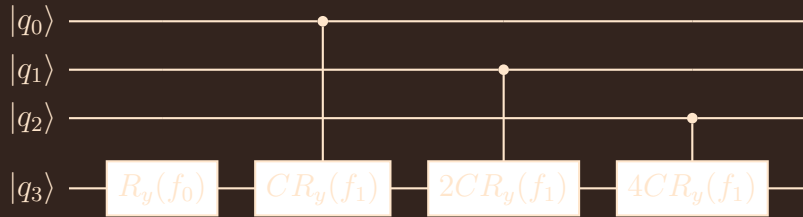


Figure 3: Quantum circuit that creates state in Eq. [25]. Three qubits  $|i_2 i_1 i_0\rangle$  are here used to be encode  $i = 4i_2 + 2i_1 + i_0 \in \{0, \dots, 7\}$ . The linear function  $f(i) = \alpha \cdot i + \beta$  is given by  $4\alpha \cdot i_2 + 2\alpha \cdot i_1 + \alpha \cdot i_0 + \beta$ .

Equation 25 can be used to create the operator that maps  $\sum_i \sqrt{p_i} |i\rangle_n |0\rangle$  to

$$\sum_{i=0}^{2^n-1} \sqrt{p_i} |i\rangle_n \left[ \cos(f_c \cdot \hat{f}(i) + \frac{\pi}{4}) |0\rangle + \sin(f_c \cdot \hat{f}(i) + \frac{\pi}{4}) |1\rangle \right], \quad (27)$$

with  $f_c \in [0, 1]$ .

$f(i)$  is of course only a mapped payoff since the payoff is usually defined on closed intervals

$$f : [a, b] \rightarrow [c, d]. \quad (28)$$



For the introduced piece-wise representation of the payoff we already used the domain and image suitable for calculations on a quantum computer

$$\hat{f} : \{0, \dots, 2^n - 1\} \rightarrow [0, 1]. \quad (29)$$

Hence it is necessary to introduce an affine transformation to realize the transition between 28 and 29

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

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

Consequently,  $\sin^2[f_c \cdot \hat{f}(i) + \pi/4]$  is an anti-symmetric function around  $1/2$ . With  $\hat{f}(i)$  now being set into 27 the probability to find the ancilla qubit in state  $|1\rangle$ ,

$$P_1 = \sum_{i=0}^{2^n-1} p_i \sin^2(f_c \cdot \hat{f}(i) + \frac{\pi}{4}), \quad (32)$$

is well approximated by

$$P_1 \approx \sum_{i=0}^{2^n-1} p_i (f_c \cdot \hat{f}(i) + 1/2) = f_c \frac{E[f(x)] - c}{d - c} - f_c + \frac{1}{2}. \quad (33)$$

The approximation is obtained by making use of the following Taylor expansion

$$\sin^2(c\hat{f}(i) + \frac{\pi}{4}) = f_c \hat{f}(i) + \frac{1}{2} + \mathcal{O}(f_c^3 \hat{f}^3(i)), \quad (34)$$

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

From 33  $E(f(x))$  can be recovered since Quantum Amplitude Estimation allow to efficiently estimate  $P_1$  and because  $f_{min}$ ,  $f_{max}$  and  $f_c$  are known.

### 3.2.5 Implementation of max-function

Mimicking the linear behavior of the payoff is done by the approach outlined in Sec. 3.2.4, but still a way to express the max operation on a quantum computer is missing. In the following one approach is shown by comparison between the values encoded in the basis states of Eq. 23.

In principle we use  $n$  additional ancilla qubits  $|a\rangle$  in addition to the already given  $n$  data qubits in the  $|i\rangle$ -register to build a greater-equal operation.

1. Convert the breakpoint of piece-wise linear function  $b \in [0, 2^n - 1]$  in bit representation  $t$ . The breakpoint indicates the threshold from which a actual payoff is gained and is therefore equal to the strike price  $K$ . To prevent the breakpoint from lying in between two grid points a ceil operation is applied on  $b$  to set it to next grid point before the conversion is done. During the conversion every qubit is represented by a bit.
2. Iterate over all qubits ( $j$  is used as iteration parameter)
  - if  $j = 0$  and  $t[j] = 1$ , perform a  $CX$  operation on  $|q_j\rangle$  and  $|a_j\rangle$ .

- if  $t[j] = 1$  an *OR*-Operation has to be applied to  $|q_j\rangle$  and  $|a_{j-1}\rangle$  and save the result of the *OR*-Operation in  $|a_j\rangle$
  - if  $t[j] = 0$  an *CCX*-Operation has to be applied to  $|q_j\rangle$ ,  $|a_{j-1}\rangle$  and  $|a_j\rangle$
3.  $|a_{n-1}\rangle$  is now  $|0\rangle$  if  $i < b$  and  $|1\rangle$  if  $i \geq b$  and therefore a quantum version of a max-function.
  4. Reverse the process by simply applying the complex conjugate of all operations except for the last qubit which does store the outcome of max-operation.

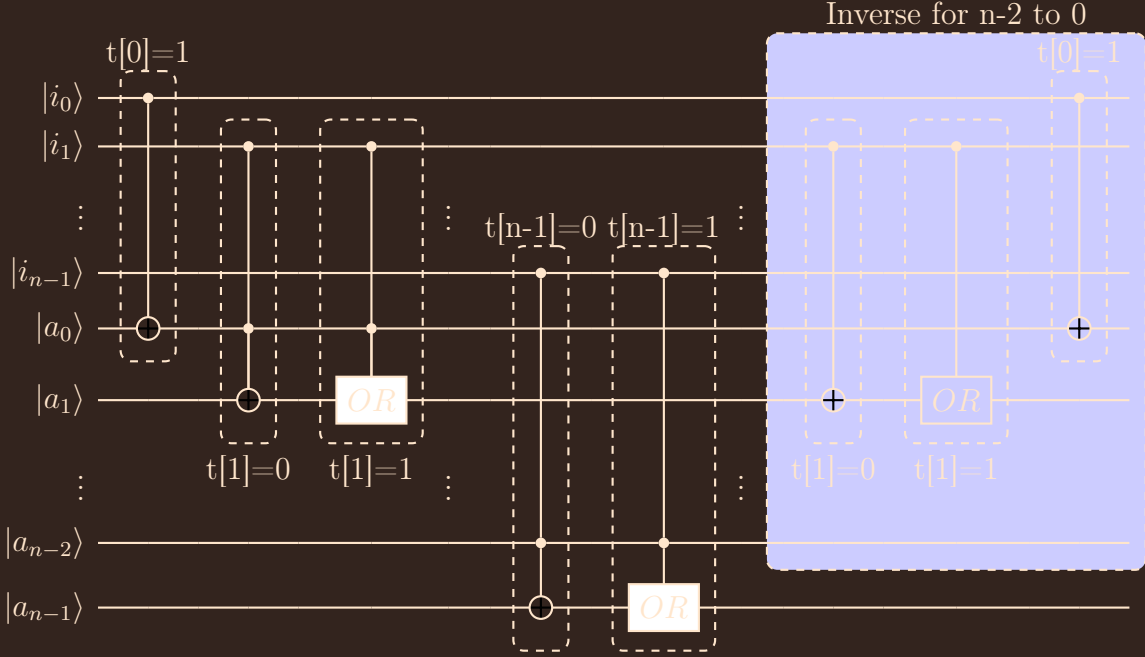


Figure 4: Quantum comparator for implementing a quantum version of max-function. Depending on strike price  $K$  and stock price stored in  $|i\rangle_n$  information if stock price is greater than strike price is stored in last qubit  $|a\rangle_{n-1}$ . State of  $|a\rangle_{n-1}$  is then kept for usage in later stage of circuit whereas rest of the circuit is inverted for next stock price.

### 3.2.6 Implementation of piece-wise linear functions

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

$$f(x) = \alpha x + \beta, \quad (35)$$

where  $f(x)$  maps  $x$  to the payoff by equation 24 and the domain of  $x$  consist of of two linear function. One for  $S_T < K$  and one for  $S_T \geq K$ . Furthermore the  $S_i$  do not occur as a real number but as an element of  $\{0, \dots, 2^n - 1\}$  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  $\hat{f}(x)$  which is in contrast to

$$f : [a, b] \rightarrow [c, d] \quad (36)$$

defined as

$$\hat{f} : \{0, \dots, 2^n - 1\} \rightarrow [0, 1]. \quad (37)$$

Note that in this case  $d = b - K$  thanks to the definition of the payoff. But how is the transformation from ?? to 37 actually done?

For piecewise function there are of course breakpoints needed within  $\{0, \dots, 2^n - 1\}$

$$b_p(x) = \frac{x - a}{b - a} \times 2^n - 1. \quad (38)$$

In case of the payoff only one breakpoint is given for  $S_T \geq K$ . The affine transformation from  $f$  to  $\hat{f}$  is then

$$\phi(x) = a + \frac{b - a}{2^n - 1} \cdot x \quad (39)$$

$$\hat{f}(x) = \frac{f(\phi(x)) - c}{d - c}, \quad (40)$$

which already have been introduced in Section 3.2.4.

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

$$\hat{\alpha} = \frac{\alpha}{2^n - 1} \cdot (b - a) \quad (41)$$

$$\hat{\beta} = \beta \quad (42)$$

$$\Rightarrow \frac{\alpha}{2^n - 1} \cdot (b - a) + \beta. \quad (43)$$

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

$$\theta_{\hat{\alpha}} = \frac{\pi}{2} f_c \frac{\alpha}{2^n - 1} \frac{b - a}{d - c} \quad (44)$$

$$\theta_{\hat{\beta}} = \frac{\pi}{4} (1 - f_c) + \frac{\pi}{2} \frac{\beta - c}{d - c} f_c, \quad (45)$$

where  $\pi/2$  is used to map from slope and offset to angles.  $f_c$  is here again the factor for Taylor approximation, introduced in ??.

Note here that the rotations using the angles above are realized on a Quantum Computer using  $R_Y$ -Gates and the inset of a slope, i.e. rotations around the slope-angle  $\theta_{\hat{\alpha}}$  is controlled by the last qubit of the quantum comparator as explained in the previous section.

Furthermore and again thanks to the fact that we are dealing with a vector in a complex Hilbert space the actual used angle for implementing offset and slope based on the given state in register  $|i\rangle_n$  is finally computed using the following equations

$$\theta_{\text{slope}}[b_p] = \theta_{\hat{\alpha}}^{b_p} - \sum_{i=0}^{N_{b_p}-1} \theta_{\hat{\alpha}}^i \quad (46)$$

$$\theta_{\text{offset}}[b_p] = \theta_{\hat{\beta}}^{b_p} - \theta_{\hat{\alpha}} \cdot b_p - \sum_{i=0}^{N_{b_p}-1} \theta_{\hat{\beta}}^i. \quad (47)$$

Hence the angles which have already been used for rotating the current state are subtracted from the final slope angle  $\theta_{\text{slope}}[b_p]$ . This is realized through the summation to the last breakpoint  $N_{b_p} - 1$ . The additional term in the offset  $-\theta_{\hat{\alpha}} \cdot b_p$  is coming from the fact that the payoff requires us to subtract the strike price from the current stock price, see Equation 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  $\mu$ ,  $\sigma_{\text{dist}}$  and lower and higher bounds to create the log-normal distribution. The result of the log-normal distribution can be seen in figure 5.

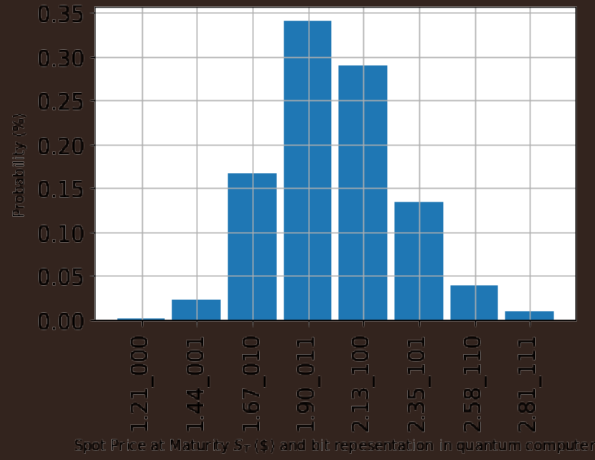


Figure 5: The distinct log-normal distribution (eq. 8), with  $\mu = (r - 0.5 \cdot \sigma^2) \cdot T + \log(S_0)$ ,  $\sigma_{\text{dist}} = \sigma \cdot \sqrt{T}$  and as lower  $l$  and highest  $h$  value we used the 3 standard deviation. As x value the classical value is shown and the corresponding qubit state which are representing the number in our quantum computer.

The equation of the European Payoff function is this

$$F(S(T)) = \max\{0, S(T) - K\} \quad (48)$$

and creates the result shown in figure 6.

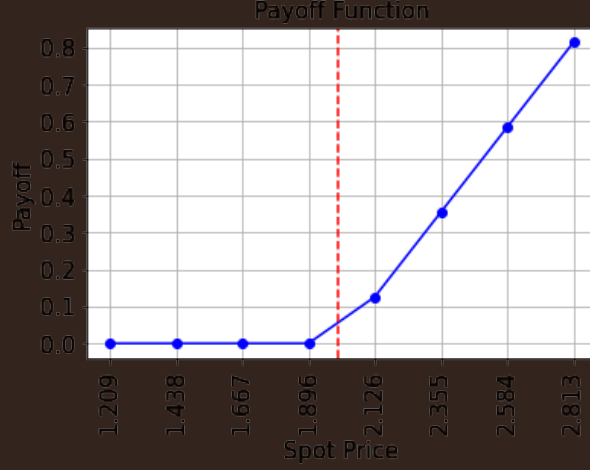


Figure 6: Here the value of the payoff function is shown. The breakpoints are set to  $[0, 2.126]$  or  $[|000\rangle, |100\rangle]$  thanks to ceil operation described in ??

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  $b_{\text{classic}} = [0, S_T]$

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

This values then result in the circuit displayed in figure 7 and a probability distribution shown in figure 8

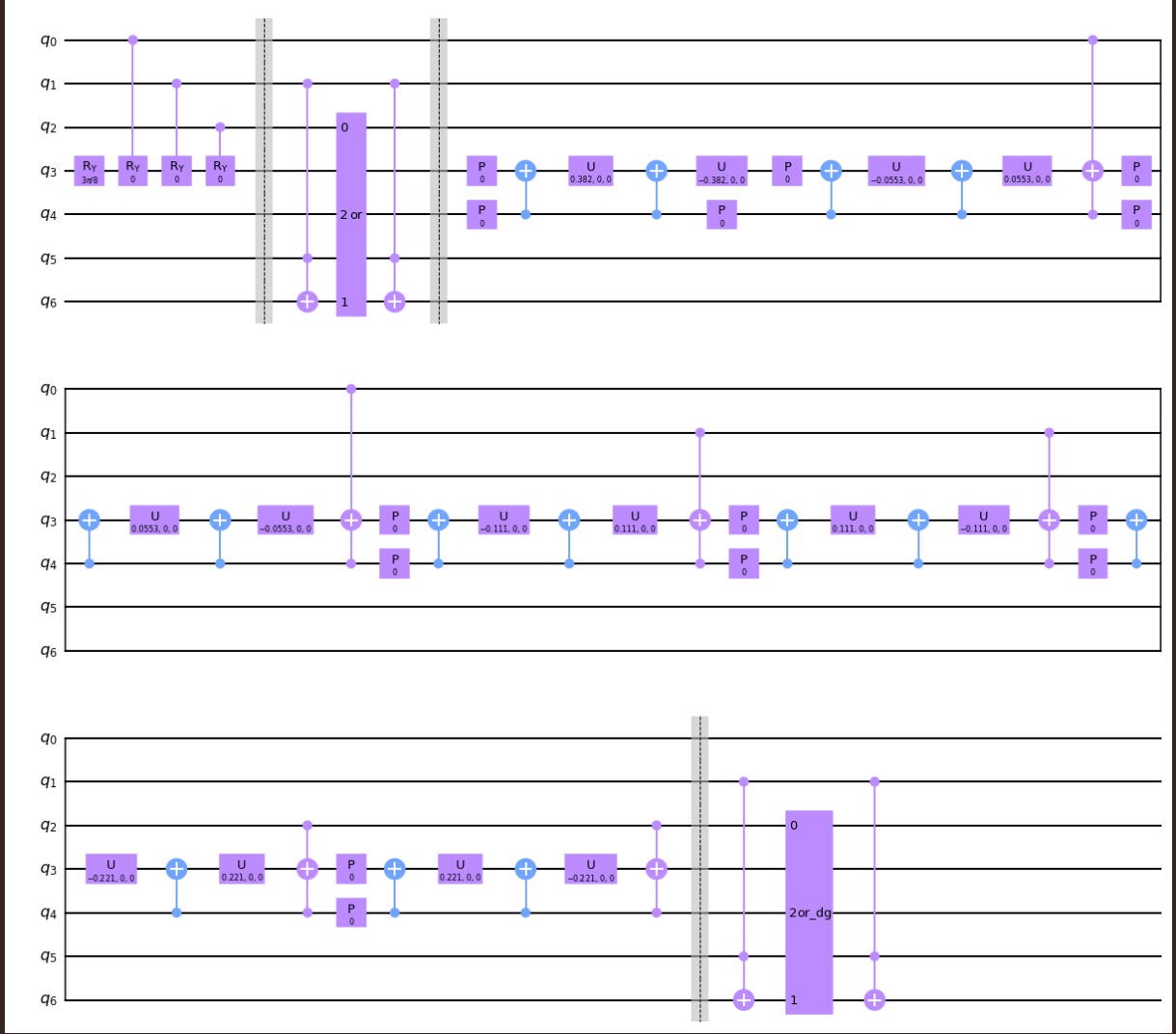


Figure 7: The output of the circuit for the payoff function. The image of the circuit was generated by qiskit.[1]. The circuit is divided in four sections. The first section is the  $Y$ -Rotation for the first breakpoint. The second section is then the greater or equal operation for the second breakpoint. 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.



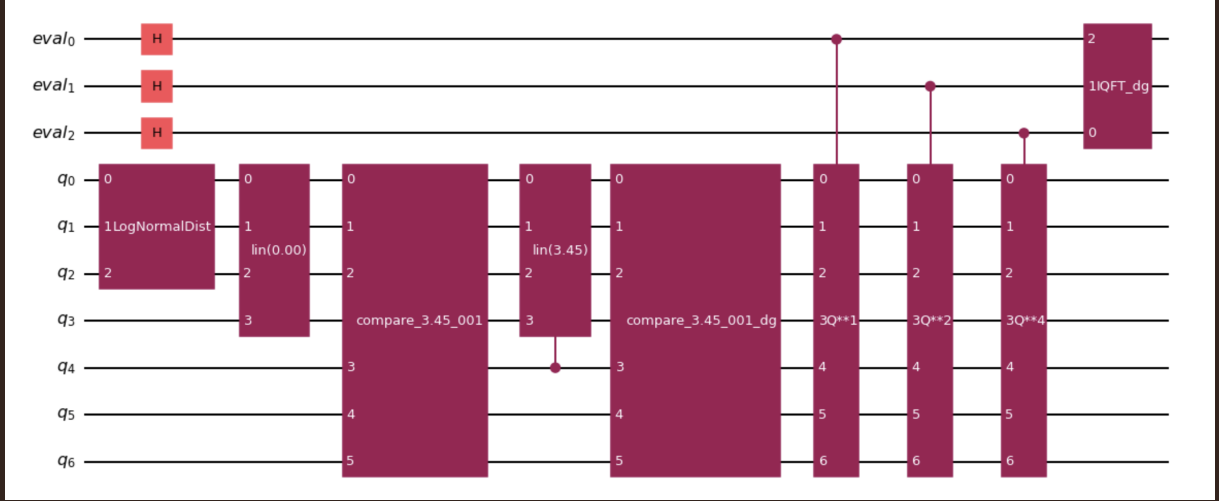


Figure 10: The combined circuit. Note that the objective qubit here is qubit 3. Evaluation qubits are chosen for pragmatical reasons. For results a register consisting of seven qubits have been used.

The combined Circuit is displayed in figure 10. 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:

$$\text{Exact Value:} \quad 0.1133 \quad (49)$$

$$\text{Estimated Value:} \quad 0.1061 \quad (50)$$

$$\text{95 Percent Confidence interval:} \quad [0.1157, 0.1209] \quad (51)$$

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. 18 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:

$$\text{MLE estimator value: } 0.1189. \quad (52)$$

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.



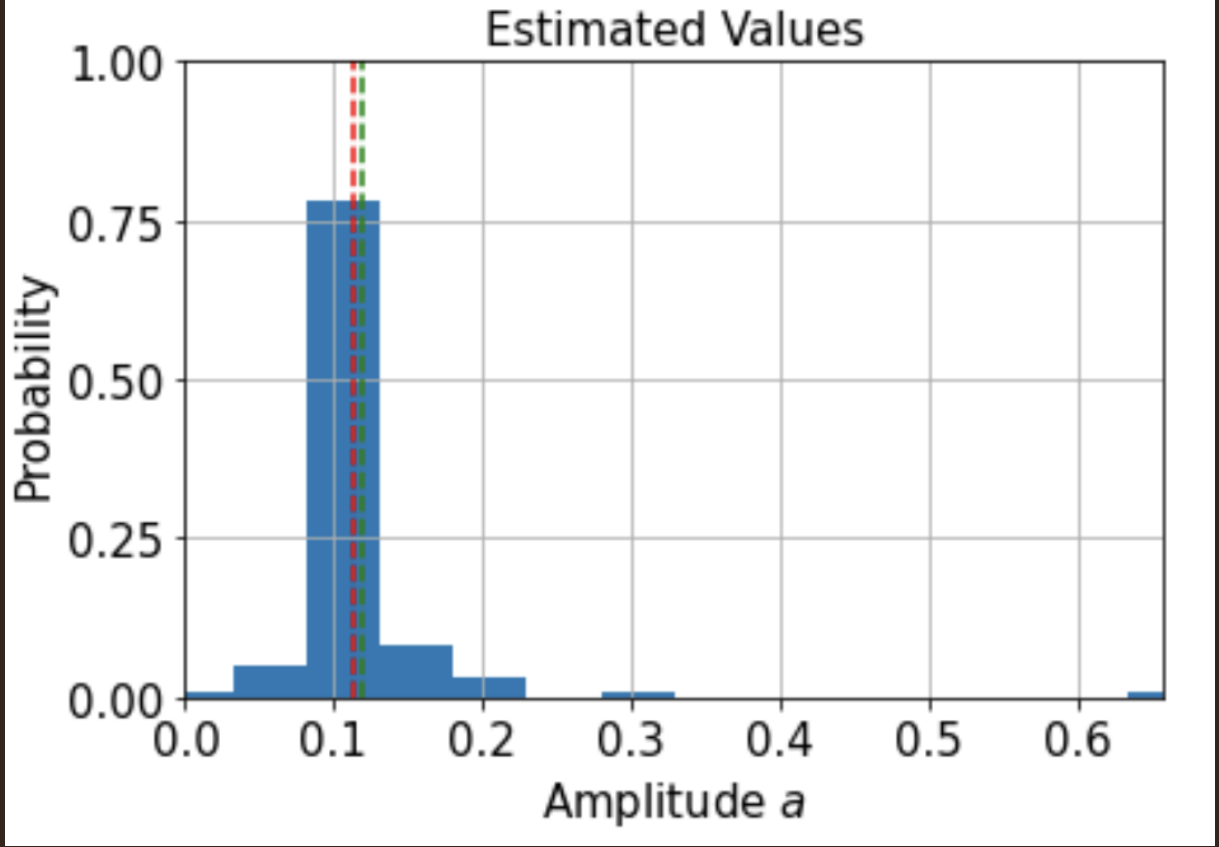


Figure 11: Results of Canonical AE. Red dotted line show the exact value. Green line is the MLE estimator value.

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

$$\text{Exact Value:} \quad 0.1133 \quad (53)$$

$$\text{Estimated Value:} \quad 0.1203 \quad (54)$$

$$\text{95 Percent Confidence interval:} \quad [0.1153, 0.1254] \quad (55)$$

Which is comparable to the values from Canonical Amplitude Estimation but comes with a lower cost, since Iterative Amplitude Estimation estimates the amplitude without 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 ones. 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/s41534-021-00379-1>.
- [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.