NExt ApplicationS of Quantum Computing

D5.4: Evaluation of quantum algorithms for pricing and computation of VaR

Document Properties

Contract Number	951821
Contractual Deadline	30-09-2022
Dissemination Level	Public
Nature	Report
Editors	Alberto Manzano, UDC Gonzalo Ferro, CESGA
Authors	Gonzalo Ferro, CESGA Alberto Manzano, UDC Andrés Gómez, CESGA Alvaro Leitao, UDC María R. Nogueiras, HSBC Carlos Vázquez, UDC
Reviewers	Vedran Dunjko, ULEI Emil Dimitrov, ICHEC
Date	23-09-2022
Keywords	Quantum Finance, Amplitude Amplification and Estimation, Quantum Accelerated Monte Carlo, Pricing, VaR
Status	Final
Release	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 951821

History of Changes

Release	Date	Author, Organisation	Description of Changes
0.1	02/09/2022	Gonzalo Ferro (CESGA), Alberto Manzano (UDC)	First full version
0.2	09/09/2022	María Nogueiras (HSBC), Carlos Vázquez (UDC), Andrés Gómez (CESGA), Alberto Manzano (UDC), Gonzalo Ferro (CESGA), Álvaro Leitao (UDC)	Executive Summary, Conclusions, and minor corrections
1.0	23/09/2022	Alberto Manzano (UDC), Gonzalo Ferro (CESGA)	Minor corrections after internal review

Table of Contents

1.Executive Summary

NEASQC Use Case 5 (UC5) works on the development and evaluation of quantum algorithms for financial applications. More specifically, the main line of research in UC5 focuses on Pricing and Value at Risk (VaR) problems. These two problems are computationally demanding tasks that are classically solved using Monte Carlo (MC) techniques. As Quantum Accelerated Monte Carlo (QAMC) techniques promise a quadratic speedup over Classical Monte Carlo (CMC) this roughly motivates why and how this field could benefit from the recent advances in Quantum Computing.

This report summarises the development of a new pricing algorithm as well as its experimental assessment. In the first part of the report there is a brief summary of the classical and quantum techniques used to tackle both financial challenges. In the second part of the report the new method is explained in detail and evaluated. This new method includes two well defined parts: a new encoding for the quantum oracle and a new Amplitude Estimation (AE) technique.

The new proposals are not yet applicable to current NISQ architectures although they represent a clear advance because:

- The new encoding algorithm allows pricing derivative products with negative payoffs. In particular, as illustrated in Section 3.1, it works in cases where other proposals fail as it is the case for a payoff of the form $V(x) = x - K$.
- The new AE algorithm, the so-called Real Quantum Amplitude Estimation (RQAE), allows pricing financial products with negative values (see Figure 9), which is an absolute novelty in the area. Current AE algorithms in the literature are concerned with the efficient estimation of the probability of finding a particular state, but they are not designed to be sensitive to the phases present in the underlying amplitudes. In contrast, RQAE is specifically tailored to be sensible to the sign of the amplitude.
- RQAE achieves a similar performance when compared with other previous well-known AE algorithms. In fact, in reference (Manzano et al., 2022), tighter bounds on the convergence of RQAE than any of the existing methods in the literature are proved. In this work, the experimental results show that although the performance is not the best one, it is on par with the cutting edge methods.

However, there are some less positive results as:

- The new encoding algorithm gives a worse performance compared to the standard encoding algorithm. It depends on a factor (N_{paths}) that can overshadow or even kill the speedup depending on its specific setup. Nevertheless, NEASQC researchers who participate in the Use Case are actively working in new encoding methods to solve this issue.

Two identified challenges have to be addressed in the near future. On the one hand, how to improve these algorithms to execute on current Quantum Processing Units (QPU), where deep circuits such as those required by the AE routines are not feasible without error correction techniques. On the other hand, previous works such as (Montanaro, 2015; Rebentrost et al., 2018) implicitly assume that the simulation of the Stochastic Differential Equation (SDE) to solve for these financial calculations can be directly done by translating the classical circuits into quantum circuits. However, this direct translation would require in practice a number of qubits too far from the current possibilities of the NISQ era.

2.Introduction

As discussed in the classical literature about quatitative finance, several strategies can be followed to solve the problem of pricing financial derivatives. In Figure 1 we depict schematically some of the most common ones. For an in depth review on the main concepts and strategies in derivatives pricing and VaR see (Gómez et al., 2022) and the references therein.

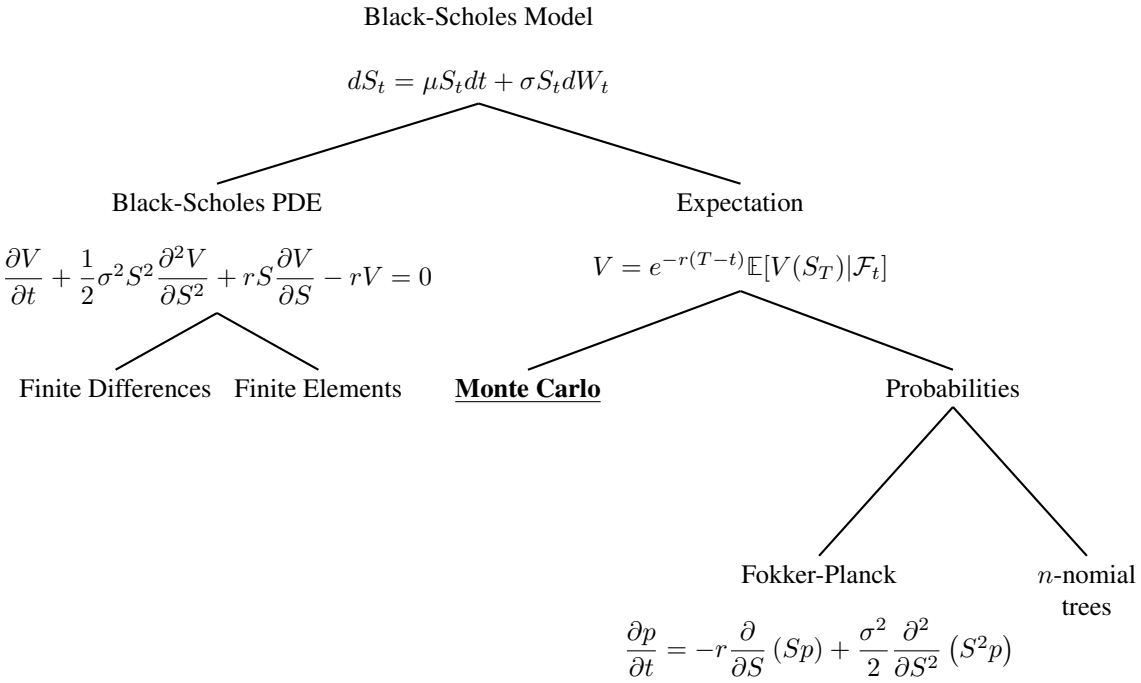


***Figure 1**: Some mathematical strategies for the pricing of financial derivatives.*

This document is particularly focused on pricing and VaR, via Monte Carlo (MC) techniques, because they share main kernel algorithms. Therefore, the forthcoming Section 2.1 briefly describes how classical MC techniques are used for pricing, while Section 2.2 outlines the way to obtain a quadratic speedup of this algorithm using Quantum Computing. Section 2.3 describes how the problem of computing VaR is transformed to that of pricing and the approximations to calculate it.

After this brief contextual review, Section 3 presents the authors contributions to this area, researched and developed during the execution of the Use Case 5 (UC5) of NEASQC European project: new algorithms for encoding data (Section 3.1) and for Amplitude Estimation (Section 3.2). The descriptions include their experimental evaluation as well.

Finally, the document closes with the main conclusions.

Additionally, two Appendices are included. The first one (Appendix A) documents the library with the Quantum Algorithms that have been implemented for Quantum Learning Machine (QLM). Appendix B describes the methodology for the experimental evaluation of the algorithms.

2.1. Classical Monte Carlo for derivatives pricing

The Classical Monte Carlo (CMC) method for derivatives pricing in finance is composed of two steps:

1. Simulation of an stochastic differential equation (SDE) satisfied by the underlying assets.
2. Use of the Monte Carlo integration techniques to compute an expectation.

2.1.1. Simulation of SDEs

For the simulation of a SDEs there exists several numerical methods such as the Euler-Maruyama method. For example, for the following Black-Scholes SDE (Black & Scholes, 1973):

$$dS_t = rS_t dt + \sigma S_t dW_t, \tag{2.1}$$

the application of the Euler-Maruyama scheme leads to the expression (Achdou & Pironneau, 2005):

$$S_{t+\Delta t} = rS_t \Delta t + \sigma S_t \sqrt{\Delta t} X. \tag{2.2}$$

Note that W in Equation (2.1) represents a Brownian motion, X is a standard normal random variable, i.e. with mean equal to 0 and variance equal to 1, and Δt is the time step that is considered in the numerical method. The time step is also referred as step of the time discretization. In this case, $\Delta t = (T-t)/N_T$, N_T being the number of time steps, T the maturity date and t the initial time. By using Equation (2.2), it is straightforward to produce samples of S_T, starting from a value of S_t at time t. More precisely, as follows:

- Start with an initial point S_t.
- Get a sample from the normal distribution.
- Compute $S_{t+\Delta t}$ from the random sample generated in the previous step.
- Repeat the process N_T times until maturity to obtain a sample of S_T.

2.1.2. Integration by Monte Carlo

Via the simulation of SDEs described in Section 2.1.1 it is possible to produce sample S_T^i of the random variable S_T, which can then be used to estimate the expectation:

$$\mathbb{E}[V(S_T)|\mathcal{F}_t] = \frac{1}{N_{\text{paths}}} \sum_{i=0}^{N_{\text{paths}}-1} V(S_T^i) + \epsilon_{MC} + \epsilon_{EM}. \tag{2.3}$$

Here N_{paths} is the number of samples (paths) S_T^i generated by the simulation of the SDE, V is the payoff function of the target derivatives contract, ϵ_{MC} is the error due to the Monte Carlo approximation of the expectation and ϵ_{EM} is the error due to the Euler-Maruyama scheme in the approximation of the samples. Due to Chebysev's inequality, the statistical error scales as (Glasserman, 2004):

$$\epsilon_{MC} \sim \frac{1}{\sqrt{N_{\text{paths}}}}. \tag{2.4}$$

The error due to the Euler-Maruyama scheme is (Kloeden & Platen, 2013):

$$\epsilon_{EM} \sim \Delta t = \frac{1}{N_T}. \tag{2.5}$$

Taking the definition of the total cost of the algorithm C as the number of calls to the Euler-Maruyama formula defined in Equation (2.2), it is straightforward to derive that the total cost is approximately $C \approx N_t N_{\text{paths}}$. Hence, the overall error of the algorithm ϵ scales with the cost as:

$$\epsilon \sim \frac{1}{\sqrt{C}}. \tag{2.6}$$

2.2. Quantum Accelerated Monte Carlo for derivatives pricing

The quantum computing community has proposed a quantum version of the MC techniques which can obtain quadratic speedups (see (Montanaro, 2015) for very general settings. We will refer to such techniques as Quantum Accelerated Monte Carlo (QAMC). In particular, this quadratic speedup can be applied to tackle financial problems such as derivatives pricing (see (Rebentrost et al., 2018; Stamatopoulos et al., 2020)) and the computation of Value at Risk (VaR) (see (Egger et al., 2020; Egger & Woerner, 2019)).

Nevertheless, QAMC can be a very misleading term outside the quantum computing community. Similarly to the CMC there are two main building blocks. On the one hand, there is the generation of an oracle and on the other hand the measuring step. These two blocks are discussed in Sections 2.2.1 and 2.2.2, respectively.

2.2.1. Standard encoding

The QAMC algorithm begins by creating a state in superposition where the probabilities of each path match those of the classical process discretized by using some numerical scheme such as the Euler-Maruyama formula. In order to build the algorithm $N_T + 1$ different registers are needed, one for each time step. The first N_T registers are composed of two registers of n_{qb} qubits each (see Figure 2a):

$$[|0\rangle\,|0\rangle]_0 \otimes [|0\rangle\,|0\rangle]_1 \otimes ... \otimes |0\rangle_{N_T}\,, \tag{2.7}$$

where $[|0\rangle\,|0\rangle]_i = \left[|0\rangle^{\otimes n_{qb}}\,|0\rangle^{\otimes n_{qb}}\right]_i$ and $|0\rangle_{N_T} = |0\rangle^{\otimes n_{qb}}$.

Each of the individual registers $|0\rangle^{\otimes n_{qb}}$ will be used to represent a decimal number. For simplicity, it can be understood as a single precision register.

The algorithm now requires the definition of a unitary operator U_p which performs the following transformation:

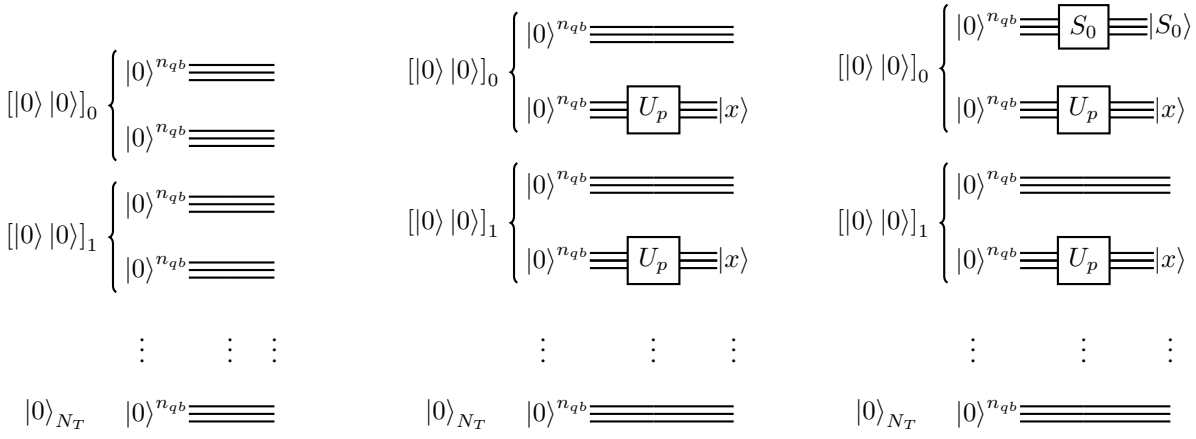
$$[U_p\,|0\rangle\,|0\rangle]_i = \left[\sum_{i=0}^{J-1}\sqrt{p(x_j)}\,|x_j\rangle\,|0\rangle\right]_i\,, \tag{2.8}$$

where x is a set of J floating point numbers that can be represented by the n_{qb} qubits from the individual registers and $p(x)$ is a discrete probability distribution defined in the set $x = \{x_0, x_1, ..., x_J\}$. Note that, in general, J doesn't need to be equal to $2^{n_{qb}} = N_{qb}$. The efficiency of this subroutine is crucial for the overall efficiency of the algorithm. In the best case, this efficiency can be achieved using $O(\log_2(J))$ gates (see (Grover & Rudolph, 2002)). In the worst case, it can be achieved in $O(J\log_2(J))$ combining the results in (Grover & Rudolph, 2002) and (Shende et al., 2006).

The first step requires applying the operator U_p to one of the members of all pairs $[|0\rangle\,|0\rangle]_i$, thus obtaining the state:

$$[U_p\,|0\rangle\,|0\rangle]_0 \otimes [U_p\,|0\rangle\,|0\rangle]_1 \otimes ... \otimes |0\rangle_{N_T} = \left[\sum_{j=0}^{J-1}\sqrt{p(x_j)}\,|x_j\rangle\,|0\rangle\right]_0 \otimes \left[\sum_{j=0}^{J-1}\sqrt{p(x_j)}\,|x_j\rangle\,|0\rangle\right]_1 \otimes ... \otimes |0\rangle_{N_T}\,. \tag{2.9}$$

In this configuration, the amplitudes encode (with the square roots) the probabilities for all the different combinations of x in the different steps. In order to continue, the first register also has to be initialised to S_0. More precisely, the first state $|0\rangle$ has to be transformed to the state $|S_0\rangle$. Figure 2 depicts schematically this process.



(a) Structure of registers. *(b) Loading probability.* *(c) Loading initial value.*

***Figure 2**: Circuit initialisation.*

Once the circuit is correctly initialised, an evolution operator must be applied. This evolution operator acts upon three individual registers as follows:

$$U_{\Delta t}\left[|x_j\rangle\,|S_k\rangle\right]_i\left[|0\rangle\right]_{i+1} \longrightarrow \left[|x_j\rangle\,|S_k\rangle\right]_i\left[|S(S_k,x_j)\rangle\right]_{i+1},$$

where the update rule is given, for instance, by Equation (2.2). Figure 3 depicts schematically this process.

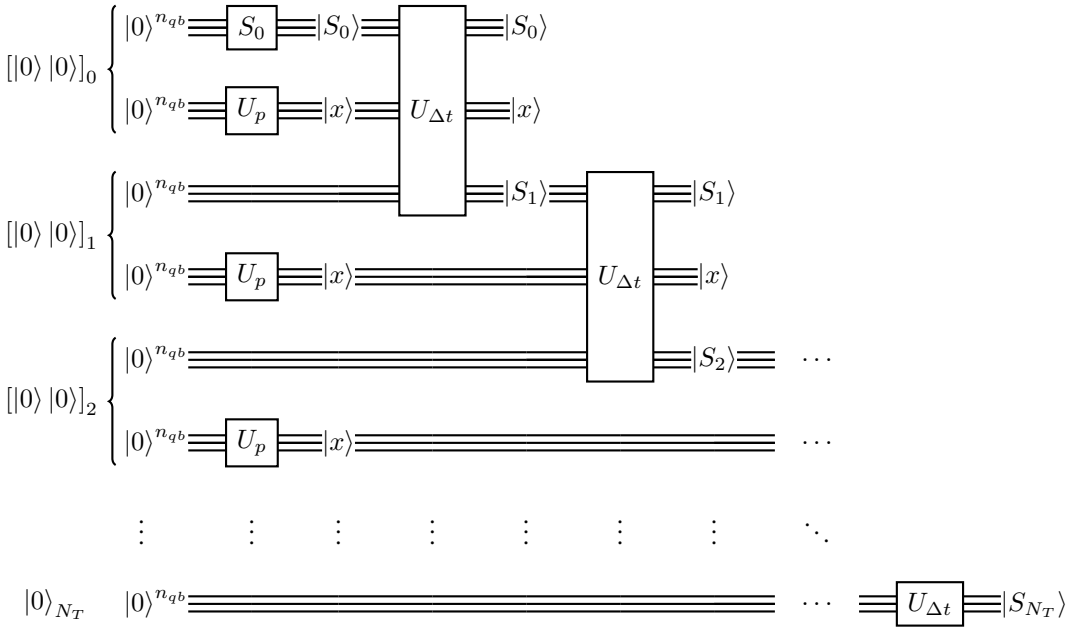


***Figure 3**: Sketch description of the construction of the oracle defined in Equation (2.10).*

So far, a circuit has been built which samples paths with the same probability as the classical circuit would. Moreover, the computational cost of one execution of the circuit is the same as in one execution of the classical circuit. In other words, the number of gates needed to sample one path from the classical and the quantum circuit is "the same", as the classical circuit can always be translated to a quantum one using Toffoli gates (see (Nielsen & Chuang, 2011)). However note that classical and quantum gates are not directly comparable.

2.2.2. Amplitude estimation

As discussed in the previous section, up to this point the quantum and the classical circuit have the same complexity. Nevertheless, when the error correction is taken into consideration the current quantum gates are much slower than the analogous classical ones. Now, the mechanism that produces an speedup is briefly described.

The current quantum state is:

$$|S\rangle = U_S\,|0\rangle = \sum_{i=0}^{N_{\text{paths}}-1} \sqrt{p(S_i)}\,|S_i\rangle\,, \tag{2.10}$$

where N_{paths} are the number of possible paths defined by the given discretization. The next step is to define the operator U_V such that pushes the payoff into the amplitude. In order to do that an additional single qubit register is needed:

$$|V\rangle = U_V\,|S\rangle = \frac{1}{\|\sqrt{V(S)}\|_\infty}\sum_{i=0}^{N_{\text{paths}}-1} \sqrt{p(S_i)V(S_i)}\,|S_i\rangle\,|0\rangle + \sqrt{p(S_i)\,(1-V(S_i))}\,|S_i\rangle\,|1\rangle\,. \tag{2.11}$$

For simplicity the term $\|\sqrt{V(S)}\|_\infty$ is considered to be equal to one. In other words, it is assumed to be properly normalised. Moreover, it tacitly assumed that the operator U_V can be efficiently implemented.

Figure 4 depicts schematically the overall process.

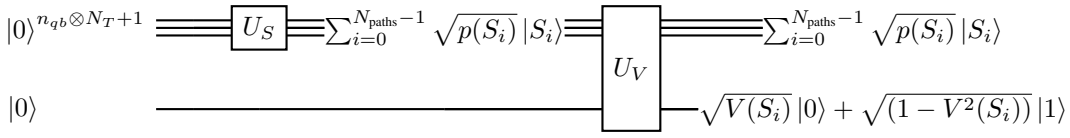


***Figure 4**: Scheme of the generation of the oracle. The gate U_S corresponds to Equation (2.10). The gate U_V corresponds to equation (2.11).*

Finally, the expectation of the payoff can be approximated by estimating the probability of obtaining $|0\rangle$ in the last register:

$$\mathbb{E}[V(S_T)|\mathcal{F}_t] = \int_{\mathbb{R}} p(t,S)V(S)dS = \sum_{i=0}^{N_{\text{paths}}-1} p(S_i)V(S_i) + \epsilon_{\text{Riemann}} + \epsilon_{\text{MC}} \approx p_{|0\rangle} + \epsilon_{\text{Riemann}} + \epsilon_{\text{MC}}, \tag{2.12}$$

where $\epsilon_{\text{Riemann}}$ is the error due to approximating the integral using a Gaussian quadrature (see (Quarteroni et al., 2006)), ϵ_{MC} is the sampling error and the expression

$$p_{|0\rangle} = \sum_{i=0}^{N_{\text{paths}}-1} |p(S_i)V(S_i)|$$

is proportional to the exact probability of obtaining the state $|0\rangle$ in the last register.

Assuming that the payoff depends only on the price of the underlying asset at maturity, the error coming from the truncation of the integral scales as:

$$\epsilon_{\text{Riemann}} \sim \frac{1}{2^{n_{qb}}}. \tag{2.13}$$

Using amplitude estimation (**AE**) techniques, the error coming from the estimation (by sampling the state $|0\rangle$) of the Riemann sum is:

$$\epsilon_{\text{MC}} \sim \frac{1}{N_{\text{oracle}}}, \tag{2.14}$$

with N_{oracle} being the number of calls to the oracle defined by Equation (2.11). Recall that this oracle is strictly the same as in the classical algorithm described in Section 2.1.1 except for the application of operator U_V.

Table 1 shows the computational cost, measured in the number of evaluations of the Euler-Maruyama formula, of each of the CMC and the QAMC. It can be easily seen that the QAMC performs quadratically better than the CMC.

	Error
CMC d-dimensions	$O(1/\sqrt{C})$
QAMC d-dimensions	$O(1/C)$

***Table 1**: Comparison of order of the errors for the CMC and the QAMC in d dimensions. The letter C denotes the number of evaluations of the Euler-Maruyama formula.*

In Figure 5 the results of the QAMC algorithm for different payoffs and **AE** algorithms are shown. If tweaked properly, the best performance is obtained with the Maximum Likelihodd Amplitude Estimation (MLAE) (Suzuki et al., 2020). However, if no fine tuning is performed, the best one is the Iterative Quantum Amplitude Estimation (IQAE)(Grinko et al., 2021) as it is more stable. The standard amplitude estimation algorithm (here denoted by CQPEAE) (Brassard et al., 2002) and the version where the classical quantum phase estimation is replaced by an iterative phase estimation (here denoted by IQPEAE) (Dobsicek et al., 2007; Kitaev, 1995) have similar results and are stable but, in general, they have worse performance. A detailed explanation of the experiments performed for getting the results presented in Figure 5 is provided in Appendix B (see Sections B.1 and B.2).

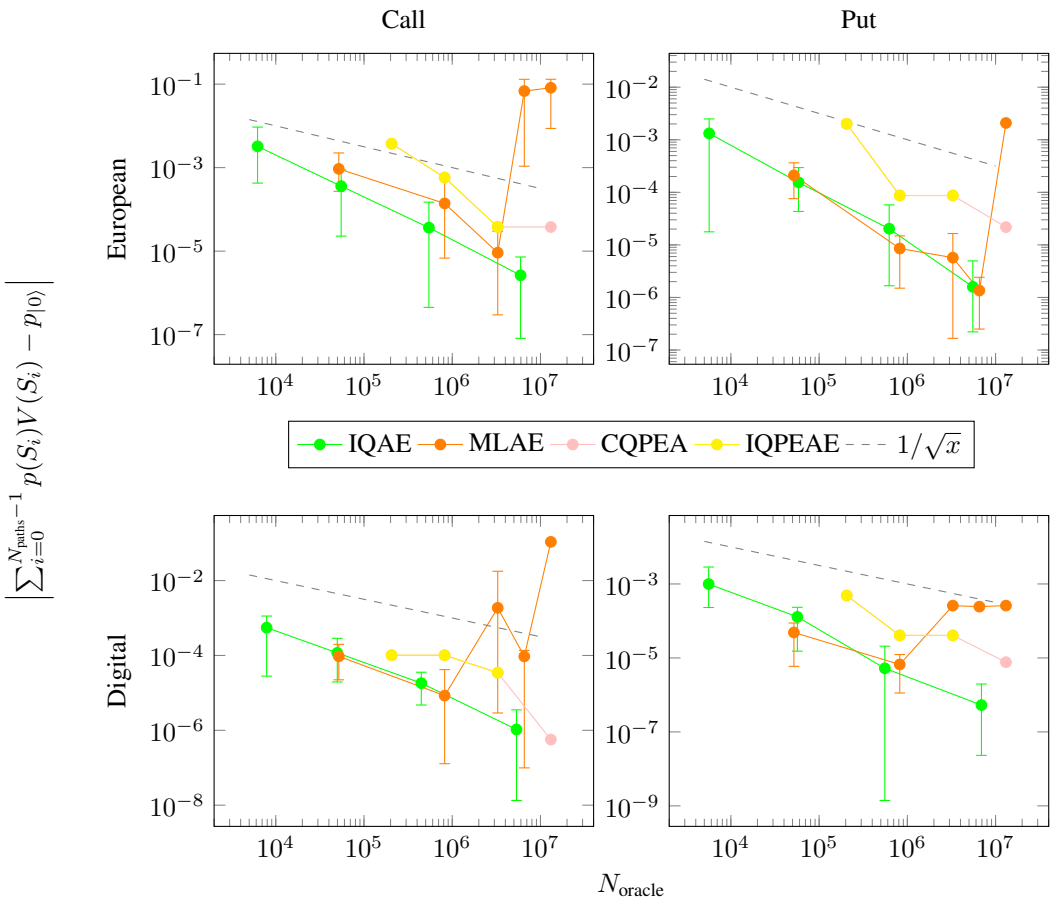


***Figure 5**: Absolute error between the **QAMC** algorithm and the Riemann sum for different number of calls to the oracle. Each of the panels corresponds to a different payoff (see **B.1** for details) and each line corresponds to a different **AE** algorithm . The experiments have been performed using the standard encoding. Detailed information about the experiments are provided in Tables **3**, **4**, **5** and **6** from Appendix **B**.*

2.3. VaR/Expected shortfall

Very similar techniques as the ones discussed in Section 2.2 can be extended to compute risk measures such as VaR. $\text{Var}_\alpha(X)$ of a random variable $X : S \to \mathbb{R}$ is defined as the smallest value x such that:

$$P[X \leq x] \geq (1 - \alpha), \tag{2.15}$$

where α is the confidence level.

In finance, typically the space S is a set of assets and X represents the profit. To simplify things, the case where there is a single asset and $X = S_T$ will be considered.

To leverage the power of a quantum computer the problem of computing VaR is transformed to that of pricing. The first part of the algorithm is the generation of an appropriate oracle. That starts by generating a state in superposition encoding the probabilities of each possible path:

$$|S\rangle = U_S\,|0\rangle = \sum_{i=0}^{N_{\text{paths}}-1} \sqrt{p(S_i)}\,|S_i\rangle\,. \tag{2.16}$$

Next, instead of pushing a payoff function into the amplitude, a step function is pushed.

$$1_{\overline{x}}(S_i) = \begin{cases} 1 & \text{if} \quad S_i(T) \leq \overline{x}, \\ 0 & \text{if} \quad S_i(T) > \overline{x}. \end{cases} \tag{2.17}$$

Applying the step function to the state $|S\rangle$, its effect will be the following. If U_S is applied followed by U_V:

$$U_V U_S \left|0\right\rangle \left|0\right\rangle = \sum_{S_i(T)\leq\overline{x}} \sqrt{p(S_i)} \left|S_i\right\rangle \left|0\right\rangle + \sum_{S_i(T)>\overline{x}} \sqrt{p(S_i)} \left|S_i\right\rangle \left|0\right\rangle . \tag{2.18}$$

Hence, the probability of measuring $|0\rangle$ is:

$$p_{|0\rangle} = \sum_{S_i(T)\leq\overline{x}} p(S_i). \tag{2.19}$$

The second part of the algorithm uses a search algorithm over $\overline{x}$ to find the $\text{VaR}_\alpha(X)$. This routine boils down to efficiently estimating the probability of measuring $|0\rangle$. Typically this is done by means of AE algorithms.

3.Original contributions

So far, in Section 2, the application of QAMC in pricing and VaR computation has been explained. Theoretically, there is a quantum advantage in the complexity scaling, although it is not achievable with the current hardware (Chakrabarti et al., 2021). This is due to three factors:

1. The implementation of the oracle U_S as explained in Section 2.2.1 requires an excessively large number of qubits.
2. The depths required by the current Grover-like routines are not feasible under the current decoherence times.
3. The total number of gates when combining the implementation of oracle U_S with a Grover-like algorithm requires a gate error beyond the capabilities of the current technology.

In the next sections, some of the problems present in the standard pricing algorithm are presented and solutions to them are proposed.

3.1. New encoding protocol

As it can be seen in Section 2.2.2, by sampling from the quantum circuit an estimation of:

$$\sum_{i=0}^{N_{\text{paths}}-1} |p(S_i)V(S_i)|\,, \tag{3.1}$$

can be obtained. Note that the payoff is assumed to be normalised so that $\|\sqrt{V(S)}\|_\infty = 1$.

Nevertheless, it is important to note that, for derivatives with payoffs that can become negative this method will not yield to correct prices approximations. In order to illustrate this, suppose that there is a payoff of the form:

$$V_T(S;K) = S_T - K, \tag{3.2}$$

with T being the maturity of the contract, S_T the price of the underlying at maturity and K the strike price of the contract (see (Gómez et al., 2022) for details).

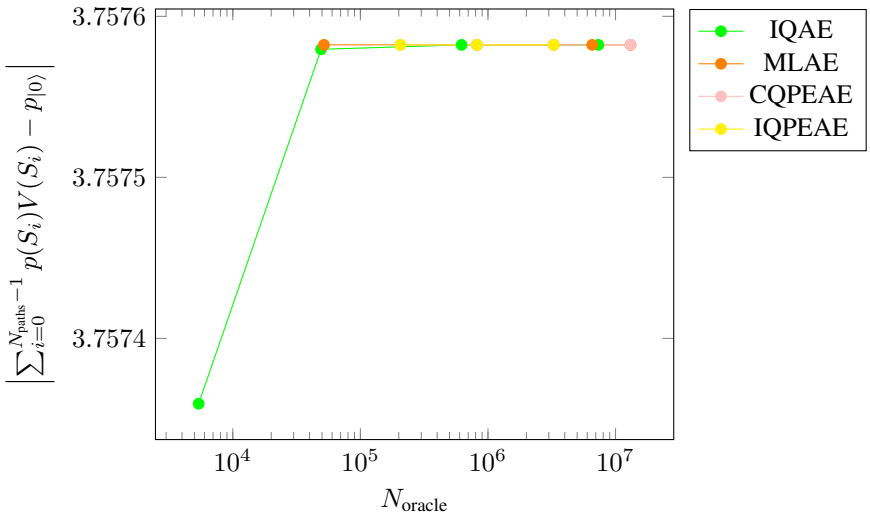


***Figure 6**: Absolute error between the QAMC algorithm and the Riemann sum of an option with a payoff $(S_T - K)$ with $K = S_0$ for different number of calls to the oracle. Each line corresponds to a different amplitude estimation algorithm. The experiments have been performed using the standard encoding. Detailed information about the experiments are provided in Tables 3, 4, 5 and 6 from Appendix B*

In Figure 6, when this standard encoding is used, it is easily seen that none of the algorithms can converge to the real value because of the presence of the absolute value. In order to address this issue, a new encoding protocol has been developed. Detailed information about the experiments for getting the results in Figure 6 can be found in subsections B.1 and B.2 included in Appendix B.

The new encoding algorithm requires a register with one additional qubit than the standard one. It begins by applying a Walsh-Hadamard transform on the first register:

$$|H\rangle = (W\mathbb{1})\,|0\rangle\,|0\rangle\,|0\rangle = (W\mathbb{1})\,|0\rangle = \frac{1}{\sqrt{N_{\text{paths}}}}\sum_{i=0}^{N_{\text{paths}}-1}|i\rangle\,|0\rangle\,|0\rangle\,, \tag{3.3}$$

where $\mathbb{1}$ denotes the identity matrix, $W = H^{\otimes n_{qb}}$ and for simplicity $|0\rangle\,|0\rangle\,|0\rangle = |0\rangle$.
Next, the encoding protocol requires a redefinition of gates U_S and U_V:

$$U_S\,|i\rangle\,|0\rangle\,|0\rangle = p(S_i)\,|i\rangle\,|0\rangle\,|0\rangle + \sqrt{1-p^2(S_i)}\,|i\rangle\,|1\rangle\,|0\rangle\,, \tag{3.4}$$

$$U_V\,|i\rangle\,|0\rangle\,|0\rangle = V(S_i)\,|i\rangle\,|0\rangle\,|0\rangle + \sqrt{1-V^2(S_i)}\,|i\rangle\,|0\rangle\,|1\rangle\,. \tag{3.5}$$

Note what is of interest now is to load the payoff and probability density itself instead of loading the square roots of the payoff and the probability density, as the standard protocol requires. When the operators U_S and U_V in Equation (3.3) are applied, the result is:

$$\begin{aligned}(U_V U_S)\,(W\mathbb{1})\,|0\rangle = \frac{1}{\sqrt{N_{\text{paths}}}}\sum_{i=0}^{N_{\text{paths}}-1}|i\rangle\Big[&p(S_i)V(S_i)\,|0\rangle\,|0\rangle \\ &+\sqrt{1-p^2(S_i)}V(S_i)\,|1\rangle\,|0\rangle \\ &+p(S_i)\sqrt{1-V^2(S_i)}\,|0\rangle\,|1\rangle \\ &+\sqrt{(1-p^2(S_i))(1-V^2(S_i))}\,|1\rangle\,|1\rangle\Big].\end{aligned} \tag{3.6}$$

Finally, a Walsh-Hadamard transform has to be applied again on the first register, to get:

$$(W\mathbb{1})\,(U_V U_S)\,(W\mathbb{1})\,|0\rangle = \left[\frac{1}{N_{\text{paths}}}\sum_{i=0}^{N_{\text{paths}}-1}p(S_i)V(S_i)\right]|0\rangle\,|0\rangle\,|0\rangle + \ldots \tag{3.7}$$

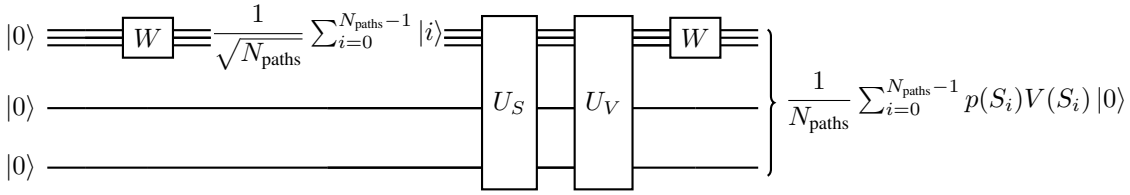


***Figure 7**: Scheme of the first algorithm.*

If projecting this state onto state $|0\rangle$, the result is:

$$p_{|0\rangle} = |\langle 0|\,(W\mathbb{1})\,(U_V U_S)\,(W\mathbb{1})\,|0\rangle|^2 = \left|\frac{1}{N_{\text{paths}}}\sum_{i=0}^{N_{\text{paths}}-1}p(S_i)V(S_i)\right|^2. \tag{3.8}$$

Hence, an estimation of the Riemann sum is obtained as:

$$\left|\sum_{i=0}^{N_{\text{paths}}-1}p(S_i)V(S_i)\right| = N_{\text{paths}}\sqrt{p_{|0\rangle}}. \tag{3.9}$$

Again, in order to achieve a speedup, an amplitude estimation algorithm is needed. As there is a very large constant factor involved, namely N_{paths}, it will be very difficult to obtain an actual speedup in practice. However, as can be seen in Figure 8, when this new encoding algorithm is used, the result converges to the Riemann sum even when the payoff takes negative values, which is a novelty when compared with other proposed methods, that cannot work with these negative values.

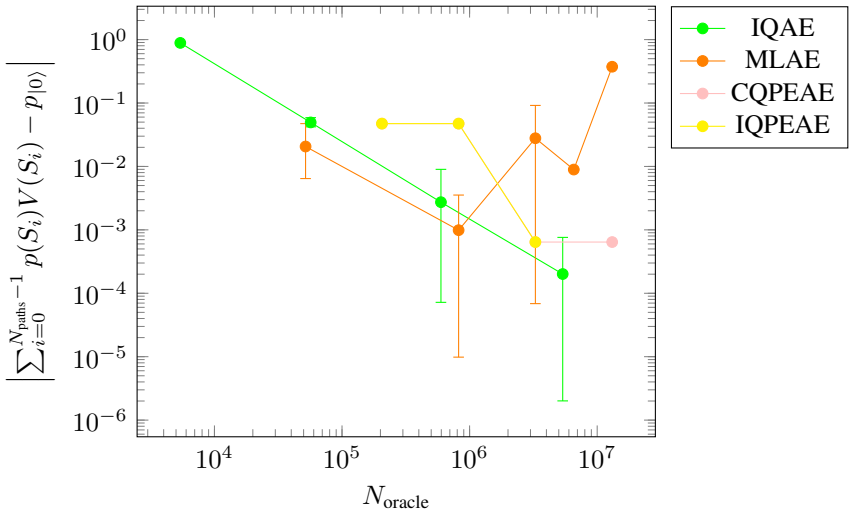


***Figure 8**: Absolute error between the QAMC algorithm and the Riemann sum of an option with a payoff $(S_T - K)$ with $K = S_0$ for different number of calls to the oracle. Each line corresponds to a different amplitude estimation algorithm. The experiments have been performed using the new encoding. Detailed information about the experiments are provided in Tables 3, 4, 5 and 7 from Appendix B.*

Detailed information about the experiments done for getting results for Figure 8 are provided in sections B.1 and B.3 included in Appendix B.

3.2. New amplitude estimation algorithm

At the end of the previous section it was briefly mentioned that the Riemann sum can be estimated through the probability of measuring state $|0\rangle$:

$$\sqrt{p_{|0\rangle}} \propto \left| \sum_{i=0}^{N_{\text{paths}}-1} p(S_i)V(S_i) \right|.$$

Now, this partially solves the initial problem. Instead of obtaining the sum of absolute values, something proportional to the absolute value of the sum is returned. Hence, in a situation where the sign of the expectation is of interest, an additional mechanism to overcome this issue is needed. In fact, this is usually the case in financial applications, where the sign is the difference between a profit and a loss.

For this case, a new algorithm called Real Quantum Amplitude Estimation (RQAE) has been developed. Its main qualitative feature is that it is able to recover the amplitude of the quantum state plus some information on the quantum phase. Additionally, it has a free parameter q called amplification ratio which allows the user to control the overall depth of the circuit. Bigger values of q yields shallower circuits but slower convergence while smaller values of q yields deeper circuits but faster convergence. In Figure 9, an example where the price of the derivative becomes negative is shown. As it can be seen, RQAE is the only algorithm that converges to the exact solution. Moreover, RQAE is competitive with the most advanced algorithms in the literature. In fact, it has the tightest convergence bounds. Nonetheless, Figure 10 shows that it requires a greater number of calls to the oracle in practice than some of the most advanced ones.

A detailed explanation of the experiments performed for getting the results presented in Figures 9 and 10 is provided in Appendix B (see subsections B.1 and B.3).

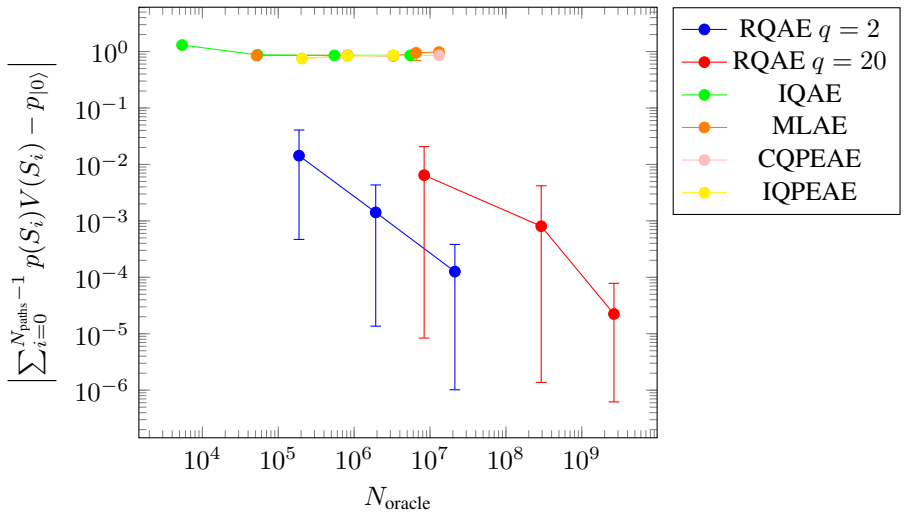


***Figure 9**: Absolute error between the QAMC algorithm and the Riemann sum of an option with a payoff $(S_T - K)$ with $K = 1.5S_0$ for different number of calls to the oracle. Each line corresponds to a different amplitude estimation algorithm. The experiments have been performed using the new encoding. Detailed information about the experiments are provided in Tables 3, 4, 5 and 7 from Appendix B.*

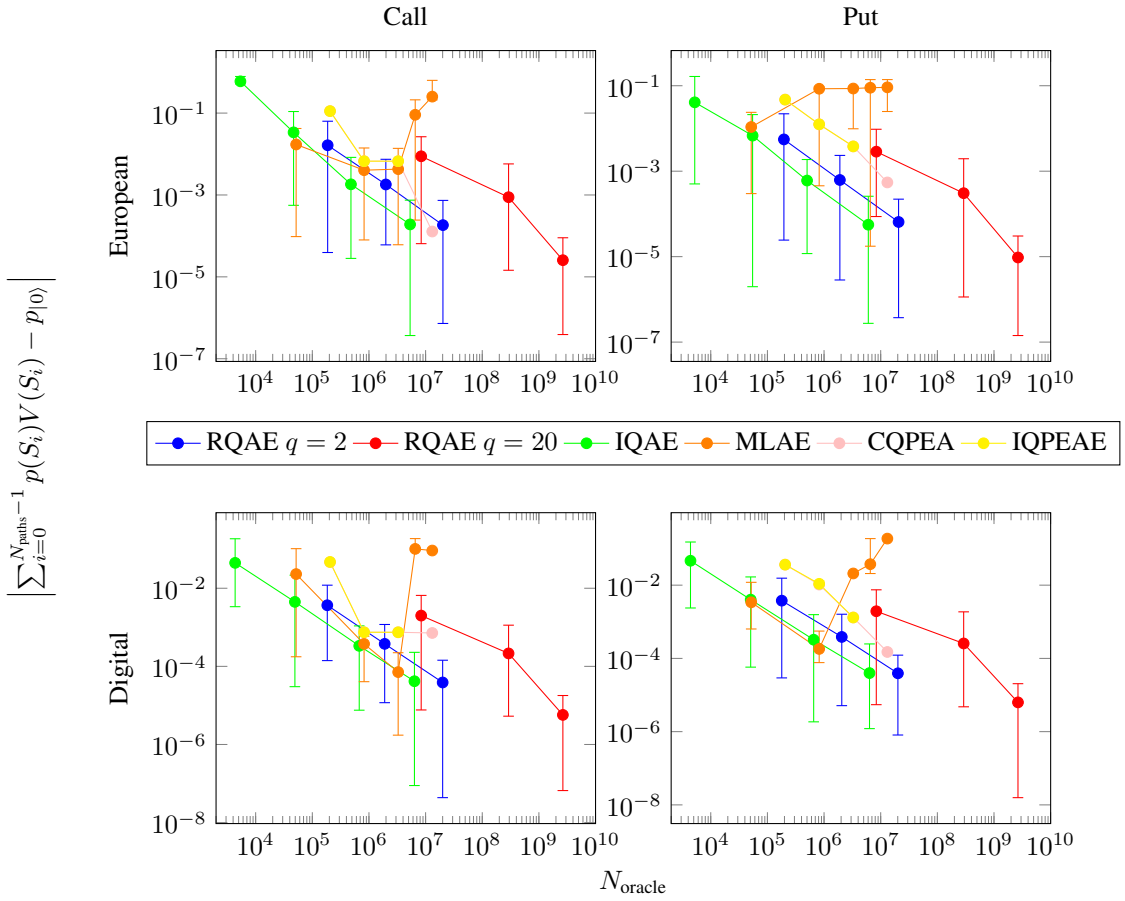


***Figure 10**: Absolute error between the QAMC algorithm and the Riemann sum for different number of calls to the oracle. Each of the panels corresponds to a different payoff and each line corresponds to a different amplitude estimation algorithm. Detailed information about the experiments are provided in Tables 3, 4, 5 and 7 from Appendix B.*

4.Conclusions

Pricing and VaR estimation are computationally demanding tasks that can benefit from the advances in Quantum Computing. In theory, as QAMC gets a quadratic speedup over CMC, it would be possible to directly apply this result to our target goals. However, there are still many issues to solve in practice, more if we take into consideration the current hardware constraints. This work presented two novel proposals and their evaluation using simulators. For making such an evaluation, an exhaustive benchmarking of the existing methods was performed with rigorous experiments to validate the actual performance of the different AE algorithms. From the obtained results presented in previous sections and from this benchmarking, some conclusions can be extracted.

The positive outcomes are:

- A new encoding algorithm allows pricing derivative products with negative payoffs. In particular, as illustrated in Section 3.1, it works when other older proposals fail for a payoff of the form $V(x) = x - K$.
- The new AE algorithm RQAE allows pricing financial products with negative values. The AE algorithms in the literature are concerned with the efficient estimation of the probability of finding a particular state, but they are not designed to be sensible to the phases present in the underlying amplitudes. In contrast, RQAE is specifically tailored to be sensible to the sign of the amplitude.
- RQAE achieves a similar performance when compared with other previous well-known AE algorithms. In the reference (Manzano et al., 2022), tighter bounds on the convergence of RQAE than any of the existing methods in the literature are proven. Here, experimental results show that although the performance is not the best one, it is on par with the cutting edge methods.
- A new library of quantum computational methods in finance for QLM has been made available. This library includes implementations of both encoding protocols and all the AE used algorithms. Moreover, it provides many user-friendly notebooks showing how the library can be used. A more detailed description of the available software can be found in Appendix A.

However, there are some less positive outcomes such as:

- The new encoding algorithm gives a worse performance compared to the standard encoding algorithm. This is due to a factor N_{paths} appearing in Equation (3.9). Depending on the specific setup this factor can overshadow or even kill the speedup. Nevertheless, NEASQC researchers who participate in the Use Case are actively working in new encoding methods to solve this caveat.

Even though some progress has been made with respect to pricing financial derivatives and VaR calculation, the delivered quantum algorithm is not competitive yet with respect to classical algorithms when executed on NISQ architectures. In this regard, the challenge to address hereon is twofold. On the one hand, without error correction techniques, the execution of deep circuits such as those required by the AE routines is not feasible. On the other hand, in works such as (Montanaro, 2015; Rebentrost et al., 2018) they implicitly assume that the simulation of the SDE can be done by directly translating the classical circuit into a quantum one. However, this direct translation would require in practice a number of qubits too far from the current possibilities of the NISQ era.

These challenges require the Use Case 5 to mainly focus its efforts on techniques which efficiently solve the SDE associated with the pricing problem. The goal would be to reduce both the depth and the number of qubits required to implement the oracle, as it would tackle both problems at the same time.

List of Acronyms

Term	Definition
AE	Amplitude Estimation
CMC	Classical Monte Carlo
CQPEAE	Amplitude Estimation based on Classical Quantum Phase Estimation
FT3	FinisTerrae III
IQPEAE	Amplitude Estimation based on Iterative Quantum Phase Estimation
IQAE	Iterative Quantum Amplitude Estimation
MC	Monte Carlo
MLAE	Maximum Likelihood Amplitude Estimation
NEASQC	NExt ApplicationS of Quantum Computing
NISQ	Noisy Intermediate-Scale Quantum
PE	Phase Estimation
QAMC	Quantum Accelerated Monte Carlo
QFT	Quantum Fourier Transformation
QLM	Quantum Learning Machine
QPU	Quantum Processing Unit
QQuantLib	Quantum Quantitative Finance Library
RQAE	Real Quantum Amplitude Estimation
SDE	Stochastic Differential Equation
UC	Use Case
VaR	Value at Risk

***Table 2**: Acronyms and Abbreviations*

List of Figures

List of Tables

Bibliography

Achdou, Y., & Pironneau, O. (2005). *Computational methods for option pricing (frontiers in applied mathematics) (frontiers in applied mathematics 30)*. Society for Industrial; Applied Mathematics.

Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, *81*(3), 637–654.

Brassard, G., Høyer, P., Mosca, M., & Tapp, A. (2002). Quantum amplitude amplification and estimation. https://doi.org/10.1090/conm/305/05215

Chakrabarti, S., Krishnakumar, R., Mazzola, G., Stamatopoulos, N., Woerner, S., & Zeng, W. J. (2021). A threshold for quantum advantage in derivative pricing. *Quantum*, *5*, 463. https://doi.org/10.22331/q-2021-06-01-463

Dobsicek, M., Johansson, G., Shumeiko, V., & Wendin, G. (2007). Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark. *Physical Review A*, *76*(3). https://doi.org/10.1103/physreva.76.030306

Egger, D. J., Gutiérrez, R. G., Mestre, J. C., & Woerner, S. (2020). Credit risk analysis using quantum computers. *IEEE Transactions on Computers*.

Egger, D. J., & Woerner, S. (2019). Quantum risk analysis. *Quantum Information*, *5*(1).

Glasserman, P. (2004). *Monte Carlo methods in financial engineering*. Springer.

Gómez, A., Leitao Rodriguez, A., Manzano, A., Nogueiras, M., Ordóñez, G., & Vázquez, C. (2022). A survey on quantum computational finance for derivatives pricing and var. *Archives of Computational Methods in Engineering*. https://doi.org/10.1007/s11831-022-09732-9

Grinko, D., Gacon, J., Zoufal, C., & Woerner, S. (2021). Iterative quantum amplitude estimation. *npj Quantum Information*, *7*(1). https://doi.org/10.1038/s41534-021-00379-1

Grover, L., & Rudolph, T. (2002). Creating superpositions that correspond to efficiently integrable probability distributions. https://doi.org/10.48550/ARXIV.QUANT-PH/0208112

Kitaev, A. Y. (1995). Quantum measurements and the abelian stabilizer problem. https://doi.org/10.48550/ARXIV.QUANT-PH/9511026

Kloeden, P. E., & Platen, E. (2013). *Numerical solution of stochastic differential equations* (Vol. 23). Springer Science & Business Media.

Manzano, A., Musso, D., & Leitao, Á. (2022). Real quantum amplitude estimation. https://doi.org/10.48550/ARXIV.2204.13641

Montanaro, A. (2015). Quantum speedup of Monte Carlo methods. https://doi.org/http://doi.org/10.1098/rspa.2015.0301

Nielsen, M. A., & Chuang, I. L. (2011). *Quantum computation and quantum information: 10th anniversary edition*. Cambridge University Press.

Quarteroni, A., Sacco, R., & Saleri, F. (2006). *Numerical mathematics (texts in applied mathematics)*. Springer-Verlag.

Rebentrost, P., Gupt, B., & Bromley, T. R. (2018). Quantum computational finance: Monte Carlo pricing of financial derivatives. *Physical Review A*, *98*(2).

Shende, V., Bullock, S., & Markov, I. (2006). Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *25*(6), 1000–1010. https://doi.org/10.1109/tcad.2005.855930

Stamatopoulos, N., Egger, D. J., Sun, Y., Zoufal, C., Iten, R., Shen, N., & Woerner, S. (2020). Option pricing using quantum computers. *Quantum*, *4*, 291.

Suzuki, Y., Uno, S., Raymond, R., Tanaka, T., Onodera, T., & Yamamoto, N. (2020). Amplitude estimation without phase estimation. *Quantum Information Processing*, *19*(2). https://doi.org/10.1007/s11128-019-2565-2

A.Financial Application library

In this appendix the Financial Application software repository is described. The repo can be downloaded at the NEASQC github: https://github.com/NEASQC/FinancialApplications.

Main idea of this repo is the development of a software library called *Quantum Quantitative Finance Library* (**QQuantLib** from now) for assembling different quantum algorithms and techniques used in the financial industry. **QQuantLib** library was developed using Python 3.9.9 and is based on the Atos Quantum Learning Machine library (**QLM**) 1.5.1.

The Financial Applications software repository has the following folder structure:

- QQuantLib. This folder contains the code of the **QQuantLib** library. The library is a typical Python library (so *import QQuantLib* gives the user the complete access to the library, its packages and modules).
- misc. This folder contains a *notebooks* folder where several jupyter notebooks were created. This notebooks were developed as a tutorial for the use of the different packages and modules of the **QQuantLib** library.
- doc. This folder contains all mandatory files for creating a web documentation which can be find at https://neasqc.github.io/FinancialApplications/.
- tests. This folder contains several Python scripts for testing purposes.
- benchmark. This folder contains several Python scripts and Jupyter notebooks for automatization of price estimation problems and benchmarking purpouses.
- binder. This folder contains mandatory files for deploying and test the complete software repository on binder environment.

The main folder of the repo is the QQuantLib where the **QQuantLib** library is developed. **QQuantLib** is organised in the following packages:

- Data Loading
- Amplitude Amplification
- Phase Estimation
- Amplitude Estimation
- Utils

In the following sections these packages are described.

A.1. Data Loading package

This package contains modules related with the loading of the data into the quantum circuits. The main module is the *data_loading* (import **QQuantLib.DL.data_loading**) where different functions for loading data into quantum circuits were developed. For the user, the more useful functions are:

- *load_probability*. It creates a *QLM AbstractGate* for loading a discretized probability density $p(x)$ (given as a numpy array) into the amplitudes of a register of a quantum circuit (Grover & Rudolph, 2002). Equation A.1 provides a mathematical representation of the desired operator.

$$U_S|0\rangle^n = \sum_{i=0}^{2^n-1} \sqrt{p(x_i)}|i\rangle^n. \tag{A.1}$$

- *load_array*. It creates a *QLM AbstractGate* for loading an input numpy array $f(x)$ following A.2.

$$U_V|x\rangle^n|0\rangle = |x\rangle^n \left(\sqrt{f(x)}|0\rangle + \sqrt{1-f(x)}|1\rangle\right). \tag{A.2}$$

Both functions load the data into the amplitudes of quantum state using controlled rotations by state $|x\rangle$:

$$|x\rangle|0\rangle \longrightarrow (\mathbb{1} \otimes R_y(\theta_x))\,|x\rangle|0\rangle = |x\rangle\,(\cos(\theta_x)|0\rangle + \sin(\theta_x)|1\rangle)\,. \tag{A.3}$$

Direct implementation of A.3 usually results in deep circuits. A more efficient circuit implementation of this kind of operators are the Quantum Multiplexors (Shende et al., 2006). The user can select between these two different implementations by providing the *method* argument to the before functions:

- brute_force: for direct implementation
- multiplexor: for multiplexor implementation

A more detailed explanation on the use of the package and the modules is provided in the jupyter notebook: 01_Data_Loading_Module_Use.ipynb

A.2. Amplitude Amplification package

This package deals with the construction of amplitude amplification (Grover-like) operators (Brassard et al., 2002) given an input oracle. The main module is *amplitude_amplification* (import **QQuantLib.AA.amplitude_amplification**) where mandatory functions for creating this Grover-like operators are located. For the user, the most useful function is the *grover* one: given an input oracle (A.4) as a *QLM Routine*

$$\mathcal{O}|0\rangle = |\Psi\rangle = \sin(\theta)|\Psi_0\rangle + \cos(\theta)|\Psi_1\rangle, \tag{A.4}$$

where $|\Psi_0\rangle$ and $|\Psi_1\rangle$ are orthogonal states, the *grover* creates a *QLM AbstractGate* implementation of the corresponding Grover-like operator (A.5)

$$\hat{Q} = \hat{U}_{|\Psi\rangle}\hat{U}_{|\Psi_0\rangle}, \tag{A.5}$$

where:

$$\hat{U}_{|\Psi_0\rangle} = \hat{I} - 2|\Psi_0\rangle\langle\Psi_0|, \tag{A.6}$$

$$\hat{U}_{|\Psi\rangle} = \hat{I} - 2|\Psi\rangle\langle\Psi|. \tag{A.7}$$

In order to use the *grover* function, the user should provide:

- oracle: *QLM Routine* with the oracle implementation.
- target: state which the user want to amplify as a Python list of ints.
- index: Python list with the index of the qubits where the Grover-like operator is applied.

For constructing the $\hat{U}_{|\Psi\rangle}$ operator (A.7), the use of a multi-controlled Z gate is mandatory. There are a couple of functions that allow the user to implement this gate in 2 ways, selected by using the Boolean argument *mcz_qlm*:

- True: direct *QLM* construction for multi-controlled Z gate is used.
- False: multiplexor implementation of multi-controlled Z gate is used.

A more detailed explanation on use of the package and the modules is provided in the Jupyter notebook: 02_Amplitude_Amplification_Operators.ipynb

A.3. Phase Estimation package

This package contains modules for pure Phase Estimation algorithms (**PE**). More precisely, given an initial state $|\Psi\rangle$ and an unitary operator $\mathcal{U}$ such that:

$$\mathcal{U}\,|\Psi\rangle = e^{2\pi i\lambda}\,|\Psi\rangle\,. \tag{A.8}$$

PE algorithms allow to estimate λ. In general, **PE** algorithms need to use the Quantum Fourier Transformation (**QFT**) as a subroutine.

PE package from **QQuantLib** has the following main modules:

- classical_qpe
- iterative_quantum_pe

Following sub sections summarise these modules.

A.3.1. classical_qpe module

This module (import **QQuantLib.PE.classical_pe**) implements classical **PE** based on **QFT** (Brassard et al., 2002). The algorithm was coded as a Python class, called *CQPE*.

In order to instantiate the class, the user should provide a Python dictionary where the following keys are mandatory:

- initial_state: *QLM Routine* with the implementation of $|\Psi\rangle$
- unitary_operator: *QLM Routine* with the implementation of the unitary operator $\mathcal{U}$

Other optional keys allow the user to configure the algorithm:

- auxiliar_qbits_number: number of qubits used for phase estimation
- shots: number of shots.

The Jupyter notebook in 04_Classical_Phase_Estimation_Class.ipynb provides more explanations and using examples of the module.

A.3.2. iterative_quantum_pe

This module (import **QQuantLib.PE.iterative_quantum_pe**) implements **PE** based on an iterative implementation of **QFT** where only one additional qbit is needed for the estimation of λ (Dobsicek et al., 2007; Kitaev, 1995). The algorithm was coded as a Python class, called *IQPE*.

In order to instantiate the class, the user should provide a Python dictionary where the following keys are mandatory:

- initial_state: *QLM Routine* with the implementation of $|\Psi\rangle$
- unitary_operator: *QLM Routine* with the implementation of the unitary operator $\mathcal{U}$

Other optional keys allow the user to configure the algorithm:

- cbits_number: number of classical bits used for phase estimation
- shots: number of shots.

The Jupyter notebook in 05_Iterative_Quantum_Phase_Estimation_Class.ipynb provides more explanations and using examples of the module.

A.4. Amplitude Estimation package

Given an oracle operator (A.4), the **AE** problem consists in getting an estimation of $\sin^2(\theta)$ by making measurements of the state $|\Psi_0\rangle$. For improving this estimation, the corresponding Grover-like operator (A.5) can be used. The effect of this operator is the following:

$$\mathcal{Q}^k|\Psi\rangle = \sin((2k+1)\theta)|\Psi_0\rangle + \cos((2k+1)\theta)|\Psi_1\rangle. \tag{A.9}$$

When the k is selected in such a way that $(2k+1)\theta \sim \frac{\pi}{2}$ then the probability of measuring $|\Psi_0\rangle$is near 1. The main problem is that this optimal k depends on θ. **AE** algorithms are systematic strategies for selecting this k aiming to obtain the best $\sin^2(\theta)$ estimation.

Several **AE** algorithms were implemented in the following modules of this package:

- maximum_likelihood_ae: implements **MLAE** algorithm (Suzuki et al., 2020).
- iterative_quantum_ae: implements **IQAE** algorithm (Grinko et al., 2021).
- ae_classical_qpe: implements **AE** using classical quantum phase estimation based on QFT A.3.1.
- ae_iterative_quantum_pe: implements **AE** using classical quantum phase estimation based on an iterative version of QFT A.3.2.
- real_quantum_ae: implements the **RQAE** algorithm (Manzano et al., 2022).

All the **AE** algorithms were implemented as Python classes that follow the same working scheme. For instantiating any of theses classes three common mandatory arguments, that are related with the oracle and the creation of the correspondent Grover-like operator, should be provided:

- oracle: *QLM Routine* with the oracle implementation.
- target: state that the user want to amplify as a Python list of ints.
- index: Python list with the index of the qubits where the Grover-like operator is applied.

Additionally, a Python dictionary can be provided for configuring the algorithm.

Each class has a *run* method that executes the **AE** algorithm and returns the desired $\sin^2(\theta)$.

In the following subsections a brief summary of the algorithms and some tips about their corresponding classes will be provided.

A.4.1. maximum_likelihood_ae

In the **MLAE** algorithm (Suzuki et al., 2020) a schedule for the Grover-like operator is defined. This schedule consist of a list of two pairs of integers (m_j, n_j) for $j \in \mathcal{N}$. For each pair (m_j, n_j) the quantum circuit for (A.9) with $k = m_j$ is created and n_j measurements will be done. The number of occurrences of the state $|\psi_0\rangle$, h_j, is recorded. So for each possible j a list of three values, (m_j, n_j, h_j), is generated and the **MLAE** algorithm can compute the associated cost function:

$$C(\theta) = -\log\left(\prod_{j=0}^{J} l_k(\theta/h_j, n_j, m_j)\right), \tag{A.10}$$

where $l_k(\theta/h_j, n_j, m_j)$ is the associated likelihood:

$$l_k(\theta/h_j, n_j, m_j) = \sin^2\left((2m_j+1)\theta\right)^{h_j} \cos^2\left((2m_j+1)\theta\right)^{n_j-h_j}. \tag{A.11}$$

Then, the **MLAE** algorithm finds the angle θ^* that minimises the cost function:

$$\theta^* = \arg\min_{\theta} C(\theta). \tag{A.12}$$

Once this θ^* is obtained then the desired $\sin^2(\theta^*)$ can be computed.

The **MLAE** algorithm was implemented in the **QQuantLib** library as a Python class called *MLAE* (from **QQuantLib.AE.maximum_likelihood_ae** import *MLAE*). For configuring the **MLAE** algorithm, a Python input dictionary can be provided. The more useful keys for the user are the following:

- schedule: Python list of list where the user can provided the schedule for the method (*example*: [[0, 1, 2, 3], [100, 200, 100, 50]]).
- optimizer: optimizer used to minimize the cost function (A.12). It should be provided as a Python **lambda function**. If not provided, the *brute* optimizer from *scipy.optimize* will be used.

The Jupyter notebook in 03_Maximum_Likelihood_Amplitude_Estimation_Class.ipynb provides a complete explanation of the **MLAE** algorithm and the *MLAE* class.

A.4.2. iterative_quantum_ae

IQAE algorithm (Grinko et al., 2021) is an iterative one where two inputs should be provided: an error ϵ and a confidence level α. The return will be two angles (θ_l, θ_u) such that the θ angle of the **AE** problem satisfies that:

$$P\left[\theta \in [\theta_l, \theta_u]\right] \geq 1 - \alpha, \tag{A.13}$$

and

$$\frac{\theta_u - \theta_l}{2} \leq \epsilon. \tag{A.14}$$

The *run* method of the *IQAE* class will return $[\sin^2 \theta_l, \sin^2 \theta_u]$.

For each step of the **IQAE**, k applications of the Grover-like operator are selected in a smart way (and depending of previous steps) trying to reduce the (θ_l, θ_u) interval until the previous conditions are satisfied.

The **IQAE** algorithm was implemented in the **QQuantLib** library as a Python class called *IQAE* (from **QQuantLib.AE.iterative_quantum_ae** import *IQAE*). For configuring the **IQAE** algorithm, a Python input dictionary can be provided. The more useful keys for the user are the following:

- epsilon (ϵ): error allowed for the algorithm
- alpha (α): confident interval for the algorithm.
- shots: number of shots for each iteration.

The Jupyter notebook in 06_Iterative_Quantum_Amplitude_Estimation_class.ipynb provides a complete explanation of the **IQAE** algorithm and of the *IQAE* class.

A.4.3. ae_classical_qpe

In this module the classical **PE** algorithm based on **QFT** (see A.3.1) is adapted for solving **AE** problems in a transparent view for the user. A Python class called *CQPEAE* was created (from **QQuantLib.AE.ae_classical_qpe** import *CQPEAE*) based on the class *CQPE* from **QQuantLib.PE.classical_pe**.

For configuring the algorithm, a Python input dictionary can be provided. The more useful keys for the user are the following:

- auxiliar_qbits_number: number of qubits for doing phase estimation.
- shots: number of shots for each iteration.

The Jupyter notebook in 04_Classical_Phase_Estimation_Class.ipynb provides a complete explanation of the algorithm and the *CQPEAE* class.

A.4.4. ae_iterative_quantum_pe

In this module the classical **PE** algorithm based on iterative **QFT** (see A.3.2) is adapted for solving **AE** problems in a transparent view for the user. A Python class called *IQPEAE* was created (from **QQuantLib.AE.ae_iterative_quantum_pe** *IQPEAE*) based on the class *IQPE* from **QQuantLib.PE.iterative_quantum_pe**.

For configuring the algorithm, a Python input dictionary can be provided. The more useful keys for the user are the following:

- cbits_number: number of classical bits used for phase estimation
- shots: number of shots for each iteration.

The Jupyter notebook in 05_Iterative_Quantum_Phase_Estimation_Class.ipynb provides a complete explanation of the algorithm and the *IQPEAE* class.

A.4.5. real_quantum_ae

As explained, typical **AE** try to estimate $\sin^2(\theta)$ by measuring the probability of the state $|\psi_0\rangle$ of (A.9). This estimation always will be in the interval $[0, 1]$, even when the amplitude of $|\psi_0\rangle$ was negative. The Real Quantum Amplitude Estimation (**RQAE**) algorithm (Manzano et al., 2022) aims to estimate the amplitude, so the $\sin(\theta)$ return of the algorithm will be in the domain $[-1, 1]$.

For the **RQAE** two inputs should be provided: an error ϵ and a confidence level γ. The return will be two values (a_l, a_u) such that the desired $\sin(\theta)$ will satisfy

$$P\big[\sin(\theta) \in [a_l, a_u]\big] \geq 1 - \gamma, \tag{A.15}$$

and

$$\frac{a_u - a_l}{2} \leq \epsilon. \tag{A.16}$$

RQAE is an iterative algorithm where, in each step, k applications of the Grover-like operator are selected in an smart way (and depending of previous steps) trying to reduce the (a_l, a_u) interval until the previous conditions are satisfied.

For configuring the algorithm a Python input dictionary can be provided. The more useful keys for the user are the following:

- epsilon (ϵ): error allowed for the algorithm
- gamma (γ): confidence interval for the algorithm.

The Jupyter notebook in 07_Real_Quantum_Amplitude_Estimation_class.ipynb provides a complete explanation of the algorithm and the *IQPEAE* class.

A.5. Utils package

In this package several auxiliary modules needed by the different packages of the **QQuantLib** library were developed:

- data_extracting: this module implements functions for creating QLM Programs from AbstractGates or QRoutines, creating their correspondent quantum circuits and jobs, simulating them and post-processing the obtained results.
- qlm_solver: module for calling the QLM solver.
- utils: module with different auxiliary functions used by the other packages of the library.
- classical_finance: module with several functions from classical quantitative finance.

Usually, the library user does not have to interact with these modules.

A.6. Price estimation and applications to finance

The price estimation of an option, given an underlying S, with strike K at maturity T, is the computation of the fair price of the option at a time $t < T$. Let $V(S)$ be the payoff of the option for a value S of the underlying and $p(T, S)$ be the probability distribution of the underlying value at option expiration time T. The classical price estimation of the option at time $t < T$ is computed by following A.17:

$$V(t, S) = e^{-r(T-t)}\mathbb{E}\left[V|\mathcal{F}_t\right], \tag{A.17}$$

where r is the *risk free rate* and $\mathbb{E}\left[V|\mathcal{F}_t\right]$ will be the expectation of the payoff $V(S)$ under probability distribution $p(T, S)$:

$$\mathbb{E}\left[V|\mathcal{F}_t\right] = \int_{\mathbb{R}} p(T, S)V(S)dS. \tag{A.18}$$

This integral can be approximated as a Riemman sum:

$$\mathbb{E}\left[V|\mathcal{F}_t\right] \sim \sum_i p(x_i)V(x_i). \tag{A.19}$$

QQuantLib allows the user to compute A.19 by using **AE** algorithms. The user needs to create an oracle operator where the desired Riemman sum (A.19) is codified in the amplitude of the $|\psi_0\rangle$, see equation (A.4). For this oracle creation, the two encoding protocols explained in Section 2.2.1 and in Section 3.1 can be used:

1. *Standard encoding*: this encoding is explained in Section 2.2.1 and summarized in equation (2.11). The *load_probability* function (see A.1) will be used for loading the desired probability density and the *load_array* function (see A.2) will be used for loading the payoff. The implementation of this encoding is explained with great detail in the 08_ApplicationTo_Finance_01_StandardApproach.ipynb Jupyter notebook. With this encoding protocol, the encoded value in the amplitude of the state to amplify will be: $\sin\theta = \sum_{i=0}^{2^n-1}\left|\sqrt{p(x_i)V(x_i)}\right|^2$.
2. *New encoding Protocol*: this encoding is explained in Section 3.1 and summarised in equation (3.7). In this case the *load_array* function (see A.2) will be used for loading probability distribution and payoff. This protocol is explained in Jupyter notebook: 10_ApplicationTo_Finance_03_StandardApproachProblems.ipynb. With this encoding protocol, the encoded value in the amplitude of the state to amplify will be: $\sin\theta = \frac{1}{2^n}\sum_{i=0}^{2^n-1} p(x_i)V(x_i)$.

A more detailed tutorial for computing price estimation using typical probability densities associated to financial models (like **Black-Scholes** model) or different options (like European Put or Call options, Digital Call or Put options or Futures) using **QQuantLib** can be found in the following jupyter notebooks:

- In 09_ApplicationTo_Finance_02_Call_Option_BlackScholes.ipynb the *standard encoding* protocol is used.
- 11_ApplicationTo_Finance_04_NewDataLoading.ipynb the *new encoding* protocol is used.

A.7. Benchmark folder

In the benchmark folder of the **Financial Applications** software repository, some scripts and notebooks were developed for the experimental tests presented in this deliverable. The user of the library can find useful the *finance_benchmark* module. This module develops a Python class called **PriceEstimation**. This class creates a *price estimation* problem and uses the **QQuantLib** library to solve it in a transparent way for the user. The user can configure the problem to be solved and the *amplitude estimation* algorithm to be used by providing an input Python dictionary.

Some useful keys for the input dictionary are:

- **ae_type**: string for setting the type of **AE algorithm to use** (*MLAE, CQPEAE, IQPEAE, IQAE, RQAE*)
- **probability_loading**. Boolean value: *True* for *standard encoding* and False for *new encoding*
- **pay_off_type**: string for setting the type of option: *European_Call_Option, European_Put_Option, Digital_Call_Option, Digital_Put_Option, Futures*

Additionally, different keys for **AE** algorithm, probability density or option configuration can be provided. In Jupyter notebook 01_Benchamark_PriceEstimation.ipynb an explanation of the class and the different configuration keys can be found.

The **QQuantLib** can be used for solving VaR computation problems by means of **AE** techniques. Jupyter notebook 03_Benchmark_VaR.ipynb from benchmark folder explains how to do this kind of computations. For automating these VaR computations, a module called *var_benchmark* was created, that includes several classes for doing these computations in a transparent way for the user. The Jupyter notebook 04_Benchmark_VaR.ipynb explains how these classes can be used.

B.Summary of experiments

This appendix describes the experiments performed for the evaluation of the different **AE** algorithms presented in this deliverable (see Figure 5 from Section 2.2.2 and Figures 6, 8, 9 and 10 from Section 3). The proposed experiments consist in the simulation of a price estimation problem for different financial derivatives (see Section B.1) using **QAMC** and **AE** techniques. For encoding the price estimation problem into the quantum circuit, the *standard* and the *new encoding* algorithms, respectively described in Subsection B.2 and Section B.3, were used.

All the experiments were done with the **QQuantLib** Python library and the software package *Financial Applications* described in Appendix A.

B.1. Price Estimation problems

The proposed experiments correspond to a price estimation problem for different typical derivative options, the solution of which is defined by equation (2.12). Table 3 shows the different options, their correspondent strikes (K) and returns used.

Option	K/S_0	Return
European_Call_Option	0.5	$V(S) = \max(0, S(t) - K)$
European_Put_Option	1.5	$V(S) = \max(0, K - S(t))$
Digital_Call_Option	0.5	$V(S) = 1$ if $S(t) \geq K$. 0 *otherwise*
Digital_Put_Option	1.5	$V(S) = 1$ if $S(t) \leq K$. 0 *otherwise*
Futures	0.5	$V(S) = S(t) - K$
Futures	1.0	$V(S) = S(t) - K$
Futures	1.5	$V(S) = S(t) - K$

***Table 3**: List of the different options simulated, their corresponding strikes and returns. Maturity T and initial value of the underlying, S_0, are provided in Table 4.*

For the options of Table 3, an underlying that follows *Black-Scholes* model was chosen. Table 4 shows the different parameters of the underlying used in the evaluation.

Probability type	S0	maturity	volatility	risk free rate
Black-Scholes	1.0	1.0	0.5	0.05

***Table 4**: Financial model and parameters for the underlying used in the evaluation.*

For computing expected value appearing in equation (2.12), the domain of the underlying was discretized following Table 5.

x_0	x_f	$n_q bits$	intervals
0.01	5.0	5	32

***Table 5**: Settings for the domain discretization used for the simulations.*

B.2. Standard Encoding Evaluation.

The *standard encoding* procedure explained in Section 2.2.1 and summarised in equation (2.11) was used for loading the data for the proposed price estimation problems into the quantum circuit. The following **AE** techniques were used for solving the **QAMC** simulation:

- Classical Quantum Phase Estimation, **CQPEAE**, see (Brassard et al., 2002).
- Iterative Quantum Phase Estimation, **IQPEAE**, see (Dobsicek et al., 2007).
- Iterative Quantum Amplitude Estimation, **IQAE**, see (Grinko et al., 2021).
- Maximum Likelihood Amplitude Estimation, **MLAE**, see (Suzuki et al., 2020).

In A.4 a brief summary of these techniques is provided. In Table 6 the parameters for the different tested **AE** techniques are presented.

Parameters	CQPEAE	IQPEAE	IQAE	MLAE
auxiliar qubits number	10, 12, 14, 16	N/A	N/A	N/A
classical bits number	N/A	10, 12, 14	N/A	N/A
epsilon (ϵ)	N/A	N/A	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$	NA
alpha (α)	N/A	N/A	0.05	N/A
number of different schedules	N/A	N/A	N/A	5
ns	N/A	N/A	N/A	10000
delta	N/A	N/A	N/A	10^{-7}
shots	100	100	100	NA
QPU	QLM	QLM	FT3	QLM
repetitions	10	5	10	10

***Table 6**: AE algorithms and their corresponding parameters for the standard encoding protocol.*

So the different price estimation problems defined in Tables 3, 4 and 5 were simulated using the different **AE** techniques (and the different parameters) of the Table 6. The parameter **QPU** from Table 6 references to the type of hardware platform where the simulations were done:

- **QLM**: the simulations were done in the Atos *Quantum Learning Machine*: this is a hardware platform for quantum circuit simulator up to 30 qubits: with 96 cores and 1.5 TB of RAM.
- **FT3**: simulations were done in a computation node of the FinisTerrae III: Intel Xeon Ice Lake 835Y with 256 GB of RAM and 64 cores[1].

The parameter *repetitions* in Table 6 was the number of simulations done for each possible combination of the different options (Table 3), **AE** techniques, and parameters (Table 6). As can be seen 10 repetitions were done for each **AE** technique, except for the **IQPEAE** where only 5 repetitions were done. This is because this technique needs a longer run time simulation compared with the other ones.

B.3. New encoding evaluation

The *new encoding* protocol presented in Section 3.1 and summarised in equation (3.7) was used to load the data for the proposed price estimation problems into the quantum circuit. The price estimation problems defined in Tables 3, 4 and 5 were solved using **QAMC** and the **AE** techniques enumerated in Section B.2. Additionally, the new original Real Quantum Amplitude Estimation (**RQAE**) algorithm, see (Manzano et al., 2022), was used (brief description of the algorithm is provided in A.4). Table 7 shows the different **AE** algorithms and the parameters used for the evaluation of the *new encoding* protocol.

Parameters	CQPEAE	IQPEAE	IQAE	MLAE	RQAE
auxiliar qubits number	10, 12, 14, 16	N/A	N/A	N/A	N/A
classical bits number	N/A	10, 12, 14	N/A	N/A	N/A
epsilon (ϵ)	N/A	N/A	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$	N/A	$10^{-3}, 10^{-4}, 10^{-5}$
alpha (α)	N/A	N/A	0.05	N/A	N/A
gamma (γ)	N/A	N/A	N/A	N/A	0.05
q	N/A	N/A	N/A	N/A	1.2, 1.5, 2, 5, 10, 20
number of different schedules	N/A	N/A	N/A	5	N/A
ns	N/A	N/A	N/A	10000	N/A
delta	N/A	N/A	N/A	10^{-7}	N/A
shots	100	100	100	N/A	N/A
repetitions	10	5	100	10	100
QPU	QLM	QLM	FT3	QLM	QLM + FT3

***Table 7**: AE algorithms and their correspondent parameters used for the simulations with the new encoding protocol.*

[1]The number of cores and the amount of the RAM used depended on the simulations to be executed.

See Section B.2 for explanation of **QPU** and **repetitions** parameters. For the evaluation of the new loading protocol, a detailed comparison of the results of the **IQAE** vs the **RQAE** algorithms was desired, so more repetitions for this two algorithms were used. As explained before, due to the long simulation times, lower number of repetitions were used for the **IQPEAE** algorithm.