Quantum Pricing with a Smile: Implementation of Local Volatility Model on Quantum Computer

Kazuya Kaneko,* Koichi Miyamoto,† Naoyuki Takeda,‡ and Kazuyoshi Yoshino§
Mizuho-DL Financial Technology Co., Ltd.
2-4-1 Kojimachi, Chiyoda-ku, Tokyo, 102-0083, Japan

Applications of the quantum algorithm for Monte Carlo simulation to pricing of financial derivatives have been discussed in previous papers. However, up to now, the pricing model discussed in such papers is Black-Scholes model, which is important but simple. Therefore, it is motivating to consider how to implement more complex models used in practice in financial institutions. In this paper, we then consider the local volatility (LV) model, in which the volatility of the underlying asset price depends on the price and time. We present two types of implementation. One is the *register-per-RN* way, which is adopted in most of previous papers. In this way, each of random numbers (RNs) required to generate a path of the asset price is generated on a separated register, so the required qubit number increases in proportion to the number of RNs. The other is the *PRN-on-a-register* way, which is proposed in the author's previous work. In this way, a sequence of pseudo-random numbers (PRNs) generated on a register is used to generate paths of the asset price, so the required qubit number is reduced with a trade-off against circuit depth. We present circuit diagrams for these two implementations in detail and estimate required resources: qubit number and T-count.

I. INTRODUCTION

With recent advances of quantum computing technologies, researchers are beginning considering how to utilize them in industries. One major target is finance (see [2] for a review). Since financial institutions are performing enormous tasks of numerical calculation in their daily works, speed-up of such tasks by quantum computer can bring significant benefits to them. One of such tasks is pricing of financial derivatives[1]. Financial derivatives, or simply derivatives, are contracts in which payoffs are determined in reference to the prices of underlying assets at some fixed times. Large banks typically have a huge number of derivatives written on various types of assets such as stock price, foreign exchange rate, interest rate, commodity and so on. Therefore, pricing of derivatives is an important issue for them.

In derivative pricing, we represent random movements of underlying asset prices using stochastic processes and calculate a derivative price as a expected value of the sum of payoffs discounted by the risk-free interest rate under some specific probability measure. In order to calculate the expected value, Monte Carlo (MC) simulation is often used. There are quantum algorithms for MC simulation[5, 6], which bring quadratic speed-up compared with that on classical computers and there already exists some works which discuss application of such quantum algorithms to derivative pricing[7–9]. However, in order to bring benefits to practice in finance, previous works have some room to be extended. That is, previous works consider the Black-Scholes (BS) model[10, 11]. Although the BS model is the pioneering model for derivative pricing and still used in many situations in today's financial firms, it is insufficient to consider only the BS model as an application target of MC for practical business for some reasons. First, for various types of derivatives, market prices of derivatives are inconsistent with the BS model. This phenomenon is called *volatility smile*, which we will explain in Section II. In order to precisely price derivatives taking into account volatility smiles, financial firms often use models which have more degree of freedom than the BS models. Second, the BS model is so simple that analytic formulae are available for the price of some types of derivatives in the model. In such cases, MC simulation is not necessary. Actually, banks use MC simulation mainly for complex models which can take into account volatility smiles. Although it is the natural first step to consider MC in the BS model, the above points motivate us to consider how to apply quantum algorithms for MC to the advanced models.

In this paper, we will focus on one of the advanced models, that is, the *local volatility (LV)* model[12]. The LV model, which we will describe later, is the model in which a volatility of an asset price depends on the price itself and time, so the BS model is included in this category as a special case. With degrees of freedom to adjust the function form of volatility, the LV model can make derivative prices consistent with volatility smiles. So this model is widely used to price derivatives, especially *exotic derivatives*, which have complex transaction terms such as early redemption, in many banks.

In order to price a derivative by MC simulation, we generate *paths*, that is, random trajectories of time evolution of asset prices, then calculate the expectation value of the sum of discounted payoffs which arise in each path. Since we cannot generate continuous paths on computers, we usually consider evolutions on a discretized time grid, using a random number (RN) for each time step, which represents the stochastic evolution in the step. In this paper, we focus on how to implement such a time evolution in the LV model on quantum computers.

We can consider two ways to implement the time evolution. In this paper, we call them the *register-per-RN* way and the *PRN-on-a-register* way. The difference between them is how to generate RNs required to generate a path. The register-per-RN is adopted in previous papers[7–9]. In this way, following

* kazuya-kaneko@fintec.co.jp
† koichi-miyamoto@fintec.co.jp
‡ naoyuki-takeda@fintec.co.jp
§ kazuyoshi-yoshino@fintec.co.jp
[1] As textbooks of financial derivatives and pricing of them, we refer to [3, 4]

the procedure described in, e.g., [13], one creates a superposition of bit strings which correspond to binary representations of possible values of a RN, where the probability amplitude of a each bit string is the square root of the possibility that the RN take the corresponding value. The point is that one register is used for one RN, so the required qubit number is proportional to the number of RNs used to generate a path. This can be problematic in terms of qubit number when many RNs are required. The number of RNs is equal to that of time steps times that of underlying assets[2]. The number of time steps can be large for derivatives with long maturity. Maturity can be as long as 30 years, so if we take time grid points monthly, the total number of time steps is 360. The number of underlying assets can be large too, say $O(10)$. Assuming that the number of asset is $O(10)$ and that of time steps is $O(100)$, the total number of RNs becomes $O(10^3 - 10^4)$. If we use a register with $O(10)$ qubits for each RN, the total qubit number can be $O(10^5)$ easily. The current state-of-art quantum computers have only $O(10)$ qubits[14]. Even if we obtain large-scale fault-tolerant machines in the future, the large qubit overhead might be required to make a logical bit (see [15] as a review and references therein). Therefore, calculations which require the large number of qubits as above might be prohibitive even in the future. This situation is similar to credit portfolio risk measurement, which is described in [16].

We are then motivated to consider the PRN-on-a-register way, which is proposed in [16]. In this way, one does not create RNs on different register, but generates a sequence of pseudo-random number (PRN) on a register. At each time step, the PRN sequence is progressed and its value is used to evolve the asset price. Therefore, the required qubit number does not depend on the number of RNs and is largely reduced. The drawback is the circuit depth. Here, we define the circuit depth as the number of layers consisting of gates on distinct qubits that can be performed simultaneously, as T-depth used in [17, 18]. Since calculations to update the PRN is sequentially performed on a register, the circuit depth is now proportional to the number of RNs. Since in the fault-tolerant computation some kinds of gates, for example T-gates in the Clifford+T gate set, can take long time to be run[19, 20], the sequential run of such gates might be also prohibitive in terms of calculation time[21]. At any rate, in the current stage, where it is difficult to foresee the spec of future quantum computers, we believe that it is meaningful to consider the implementation which saves qubits but consumes depth as a limit.

When it comes to the LV model, the PRN-on-a-register way becomes more motivating, since its disadvantage on the circuit depth compared with the register-per-RN way is alleviated. In the LV model, the volatility varies over time steps depending on the asset price, so the calculation for the time evolution is necessarily stepwise[3]. Therefore, the PRN-on-a-register way and the register-per-RN way are equivalent with respect to this point, that is, the circuit depth is proportional to the time step number in both ways. This is different from the situation in credit portfolio risk management [21], where, in the register-per-RN way, a register is assigned to each random number which determine whether each obligor defaults or not and parallel processing on different registers reduces circuit depth.

In this paper, we design the quantum circuits in the above two way in the level of elementary arithmetic[4]. In doing so, we follow the policies of the two ways to the extent possible. That is, not only with respect to RNs but also in other aspects, we try to reduce qubits accepting some additional procedures in the PRN-on-a-register way, and vice versa in the register-per-RN way. For example, in the PRN-on-a-register way, we have to intermediately output the information of the volatility used to evolve the asset price at each time step and clear it by the next step. Otherwise, we need a register to hold the information per step and the required qubit number becomes proportional to the number of time steps. It is nontrivial to implement such a procedure and we will present how to do this later. Note that such clearing procedure is unnecessary in the register-per-RN way.

We then estimate the resources to implement the proposed circuits. We focus on two metrics: qubit number and T-count. As mentioned above, we see that the qubit number in the PRN-on-a-register way is independent from the time step number and much less than the register-per-RN way. The T-count is proportional to the time step number in the both ways. We see that in some specific setting the both ways yield the T-counts of same order of magnitude, except that in the PRN-on-a-register way is larger by some $O(1)$ factor.

The rest of this paper is organized as follows. Section II and III are preliminary sections, the former and the latter briefly explain the LV model and the quantum algorithm for MC simulation, respectively. In section IV, we present the circuit diagram in the two way. In section V, we estimate qubit number and T-count of the proposed circuits. Section VI gives a summary.

II. LV MODEL

In this paper, we consider only the single-asset case, since it is straightforward to extend the discussion in this paper to the multi-asset case.

A. pricing of derivatives

Consider a party A involved in a derivative contract written on some asset. We let S_t denote a stochastic process which represents the asset price at time t, which is set as $t = 0$ at the present. We assume that the payoffs arise at the multiple

[2] In this paper, we consider arbitrage-free and complete markets, standard assumptions for derivative pricing, so the number of stochastic factors is equal to that of assets. For details, see [4].

[3] In the multi-asset case, parallel computing over assets is possible in the register-per-RN way.

[4] Actually, we here present only the overviews of the circuits and refer to [1] for the full details.

times $t_i^{\text{pay}}, i = 1, 2, ...$ and the i-th payoff is given by $f_i^{\text{pay}}\left(S_{t_i^{\text{pay}}}\right)$, where f_i^{pay} is some function which maps $\mathbb{R}$ to $\mathbb{R}$. The positive payoff means that A receives a money from the counterparty and the negative one means vice versa. For example, the case where A buys an European call option with the strike K corresponds to

$$f_1^{\text{pay}}(S_{t_1^{\text{pay}}}) = \max\left\{S_{t_1^{\text{pay}}} - K, 0\right\} \tag{1}$$

with a single payment date t_1^{pay}. Note that this type of derivative contract is too simple to cover all trades in financial markets. For example, *callable* contracts, in which either of the parties has a right to terminate the contract at some times, are widely dealt in markets. We leave studies for such *exotic* derivatives for future works and, in this paper, concider only those which can be expressed in the above form.

Following the theory of arbitrage-free pricing, the price V of the contract for A is given as follows [4]:

$$V = E\left[\sum_i f_i^{\text{pay}}\left(S_{t_i^{\text{pay}}}\right)\right], \tag{2}$$

where $E[\cdot]$ represents the expectation value under some probability measure, the so-called *risk-neutral measure*. Here and hereafter, we assume that the risk-free interest rate is 0 for simplicity.

B. LV model

In the LV model, the evolution of the asset price is modeled by the following stochastic differential equation (SDE)

$$dS_t = \sigma(t, S_t)dW_t \tag{3}$$

in the risk-neutral measure[5] W_t is the Wiener process which drives S_t. dX_t is the increment of a stochastic process X_t over an infinitesimal time interval dt. The deterministic function $\sigma : [0, \infty) \otimes \mathbb{R} \to [0, \infty)$ is the local volatility. Note that the BS model corresponds to the case where

$$\sigma(t, S) = \sigma_{\text{BS}} S, \tag{4}$$

where σ_{BS} is a positive constant called *BS volatility*.

The LV model was proposed to explain *volatility smile*. In order to describe this, let us define *implied volatility* first. In the BS model, a price of a European call option with strike K and maturity T at $t = 0$ is given by the following formula:

$$\begin{aligned} V_{\text{call,BS}}(T, K, S_0, \sigma_{\text{BS}}) &= \Phi_{\text{SN}}(d_1)S_0 - \Phi_{\text{SN}}(d_2)K \\ d_1 &= \frac{1}{\sigma_{\text{BS}}\sqrt{T}}\left[\ln\left(\frac{S_0}{K}\right) + \frac{1}{2}\sigma_{\text{BS}}^2 T\right] \\ d_2 &= d_1 - \sigma_{\text{BS}}\sqrt{T}, \end{aligned} \tag{5}$$

where Φ_{SN} is the cumulative distribution function of the standard normal distribution. We can price the option if we determine the BS volatility. Conversely, given the market price of the option $V_{\text{call,mkt}}(T, K)$, we can reversely calculate the BS volatility. That is, we can define the following function:

$$\begin{aligned} &\sigma_{\text{IV}} : (T, K) \mapsto \\ &\quad \sigma_{\text{IV}}(T, K) \ s.t. \ V_{\text{call,BS}}(T, K, S_0, \sigma_{\text{IV}}(T, K)) = V_{\text{call,mkt}}(T, K). \end{aligned} \tag{6}$$

We call BS volatilities drawn back from the market option prices by (6) as implied volatilities.

If the market is described well by the BS model, implied volatilities $\sigma_{\text{IV}}(T, K)$ take a same value for any K and T. Although this is the case for some markets, $\sigma_{\text{IV}}(T, K)$ varies depending on K and T in many markets. Especially, if $\sigma_{\text{IV}}(T, K)$ depends on K, it is said that we observe the volatility smile for the market.

Volatility smiles mean that possible scenarios of asset price evolution in the BS model do not match those which market participants consider. For example, if market participants think that extreme scenarios, big crashes or sharp rises, are more possible than the BS model predicts, the volatility smile arises. In fact, it is often said that the Black Monday, the big crash in the stock markets at 1987, was one of triggers of appearance of volatility smiles.

With the LV model, we can make European option prices given by the model consistent with any market prices, as long as there is no arbitrage in the market. This is intuitively apparent since we can expect that the degree of freedom of the local volatility $\sigma(t, S)$ as a two-dimensional function is available to reproduce the two-dimensional function $V_{\text{call,mkt}}(T, K)$. In fact, if we can get $V_{\text{call,mkt}}(T, K)$ as a function, that is, the market option prices for continuously infinite strikes and maturities, we can determine $\sigma(T, K)$ which reproduces $V_{\text{call,mkt}}(T, K)$ as follows[12]:

$$\sigma^2(T, K) = 2\frac{\frac{\partial}{\partial T}V_{\text{call,mkt}}(T, K)}{\frac{\partial^2}{\partial K^2}V_{\text{call,mkt}}(T, K)}. \tag{7}$$

In reality, the market option prices are available only for several strikes and maturities. Therefore, in the practical business, we usually use a specific functional form of $\sigma(t, S)$ with degrees of freedom sufficient to reproduce several available market option prices. In this paper, we use the following form. First, we set the n_t grid points in the time axis, $t_0 = 0 < t_1 < t_2 < ... < t_{n_t}$. Second, we set the n_S grid points in the asset price axis for each time grid point, that is, $s_{i,1}, ..., s_{i,n_S}$ for t_i. Then, $\sigma(t, S)$ is set as follows:

$$\sigma(t, S) = a_{i,j}S + b_{i,j} \text{ ; for } s_{i,j-1} \le S < s_{i,j}, j = 1, ..., n_S + 1 \tag{8}$$

for $t_{i-1} \le t < t_i$, where $a_{i,j}, b_{i,j}$ are constants satisfying $\sigma(t, S) > 0$ for any t and S and $s_{i,0} = -\infty, s_{i,n_S+1} = +\infty$. In other words, the two-dimensional space of (t, S) is divided in the direction of t and in each region $\sigma(t, S)$ is set to a function which is piecewise-linear with respect to S. In this paper, we assume that $a_{i,j}, b_{i,j}$ are calibrated so that the option prices in the LV model match the market prices.

[5] Note that the drift term does not exist since we are now assuming the risk-free rate is 0.

C. MC simulation in the LV model

We here describe how to calculate the derivative price (2) in the LV model by MC simulation.

First, we have to discretize the time into sufficiently small meshes, since we can deal with the continuous time on neither classical nor quantum computers. For simplicity, we set the time grid points to the above t_i's, those for the LV function. Then, the time evolution given by (3) is approximated as follows:

$$\Delta S_{t_i} := S_{t_{i+1}} - S_{t_i} \approx \sigma(t_i, S_{t_i})\sqrt{\Delta t_i} w_i, \Delta t_i = t_{i+1} - t_i, \quad (9)$$

where $w_1, ..., w_{n_t}$ are independent standard normal random numbers (SNRNs). Among various ways to discretize the SDE, we here adopt the Euler-Maruyama method [22].

Even after time discretization, we cannot consider all of continuously infinite patterns of SNRNs. One solution for this is discretized approximation of SNRNs. We can choose the finite numbers of the grid points and assign probability to each point so that standard normal distribution is approximately reproduced. Now, the patterns of discretized SNRNs are finite, so we can approximate (2) as

$$V \approx \sum_n p_n \sum_i f_i^{\text{pay}}\left(S^{(n)}_{t_i^{\text{pay}}}\right), \quad (10)$$

where p_n is the probability that the n-th pattern of values of SNRNs are realized and $S_t^{(n)}$ is the asset price at time t in the n-th pattern.

There are some possible ways to take petterns considered in (10). In the register-per-RN way, we take *all* patterns. If we take N grids to discretize each of n_t SNRNs, the number of possible patterns of SNRNs is N^{n_t}. Although this is exponentially large, quantum computers can take into account all patterns with a polynomial number of qubits by quantum superposition.

On the other hand, this cannot be adopted on classical computers, since the number of the SNRN patterns are exponentially large. Usually, MC pricing on classical computers is done in the following way, which the PRN-on-a-register way is also based on. We consider *sampled* patterns of SNRNs. That is, we generate finite but sufficiently many sample sets of $(w_1, ..., w_{n_t})$ and use them to generate sample paths of the asset price which evolves according to (9). We then approximate (2) by the average of sums of payoffs in sample paths,

$$V \approx \frac{1}{N_{\text{path}}} \sum_{n=1}^{N_{\text{path}}} \sum_i f_i^{\text{pay}}\left(S^{(n)}_{t_i^{\text{pay}}}\right), \quad (11)$$

where $S_t^{(n)}$ is the value of the asset price at time t on the n-th sample path and N_{path} is the number of sample paths.

III. QUANTUM ALGORITHM FOR MC SIMULATION

A. outline of the algorithm

We here review the quantum algorithm for MC simulation[5, 6]. It can be divided into the following steps. First, we create a superposition of possible values of a random number used to calculate a sample value of the integrand on a register. If multiple random numbers are necessary to calculate the integrand, one register is assigned per random number. As mentioned above, continuous random numbers must be approximated in some discretized way. Second, we calculate the integrand into another register, using the random numbers. Note that the results for many patterns of random numbers are simultaneously calculated in quantum parallelism. Third, by controlled rotation, the integrand value is reflected into the amplitude of the ancilla. Finally, amplitude estimation [6, 23, 24] on the ancilla gives the expectation value of the integrand.

The quantum state is transformed as follows:

$$\begin{aligned}|0\rangle|0\rangle|0\rangle &\to \left(\sum_i \sqrt{p_i}|x_i\rangle\right)|0\rangle|0\rangle \\ &\to \left(\sum_i \sqrt{p_i}|x_i\rangle|f(x_i)\rangle\right)|0\rangle \\ &\to \sum_i \sqrt{p_i}|x_i\rangle|f(x_i)\rangle\left(\sqrt{1-f(x_i)}|0\rangle + \sqrt{f(x_i)}|1\rangle\right) \\ &. \end{aligned} \quad (12)$$

Here, the first, second and third kets correspond to the random number registers, the integrand register and the ancilla, respectively. x_i represents the binary representation of values of random numbers in the i-th pattern and p_i is the probability that it realizes. f is the integrand and $f(x_i)$ is its value for x_i. Note that the probability to observe 1 on the ancilla is $\sum_i p_i f(x_i)$, the integral value which we want. Although we do not explain how to estimate the probability amplitude in this paper, it is studied in many papers. For example, see [6, 23, 24]. Using such methods, we can estimate the integral with the statistical error which decays as $O(N^{-1})$, where N is the number of oracle calls. This decay rate is quadratically faster than that in the classical algorithm, $O(N^{-1/2})$.

B. the scheme using the PRN generator

We here briefly review the quantum way for Monte Calro simulation using the PRN generator. The calculation flow for the current problem, the time evolution of asset price in the LV model, based on this way is described in Section IV A.

It is proposed in [16] in order to reduce the required qubits to generate RNs in the application of the quantum algorithm for MC to extremely high-dimensional integrations. When it is neccesary to generate many RNs to compute the integrand, the naive way, in which we assign a register to each RN and create a superposition of possible values, leads to the increase of qubit numbers in proportion to the number of RNs. In order to aviod this, we can adopt the following way. First, we prepare two registers, R_{samp} and R_{PRN}. Then, we create a superposition of integers, which specify the start point of the PRN sequence, on R_{samp}. With the start point, we sequentially generate PRNs on R_{PRN}. This is possible because a PRN sequence is a deterministic sequence whose recursion equation

is explicitely given, and in [16] we gave the implementation of one of PRN generators on quantum circuits. Using the PRNs, we compute the integrand step by step, which corresponds to time evolution of the asset price and calculation of payoffs in this paper. Finally, the expectation value of the integrand is estimated by quantum amplitude estimation. In this way, since we need only R_{samp} and R_{PRN} to generate PRNs, the required qubit number is now independent from the number of RNs and much smaller than the naive way. The drawback is the increase of the circuit depth.

IV. CIRCUIT DESIGN

Now, we present quantum circuits for time evolution of an asset price in the LV model in the two ways: PRN-on-a-register and register-per-RN.

A. the PRN-on-a-register way

1. calculation flow

We first present the calculation flow in the PRN-on-a-register way. We consider the flow until calculation of the sum of payoffs, which corresponds to from the first to the third line in (12), since the controlled rotation in the fourth line does not depend on the problem.

In the PRN-on-a-register way, PRNs are used for evolution of the asset price (9). More concretely, we preselect some sequence of pseudo standard normal random numbers (PSNRNs) and divide it into subsequences, then evolve the asset price using them.

Before we present the calculation flow, we explain some setups. We prepare the following register:

- R_{samp}: This is a register where a superposition of integers which determine the start point of the PSNRN sequence. We write its qubit number as n_{samp}. $N_{\mathrm{samp}} = 2^{n_{\mathrm{samp}}}$ is the number of sample paths we generate.
- R_W: This is a register where we sequentially generate PSNRNs.
- R_S: This is a register where the value of the asset price is stored and which we update for each time step of the evolution, using R_W.
- R_{payoff}: This is a register into which the payoffs determined by R_S are added.

Note that we need some ancillary registers in addition to the above registers. We assume that the required calculation precision is n_{dig}-bit accuracy and $R_W, R_S, R_{\mathrm{payoff}}$ and ancillary registers necessary to update them have n_{dig} qubits.

We assume that the following gates are available to generate a sequence of PSNRNs.

- P_W: This progresses a PSNRN sequence by one step. In other words, it acts on R_W and updates x_i to x_{i+1}, where x_i is the i-th element of the sequence: $|x_i\rangle \to |x_{i+1}\rangle$.
- J_W: This lets the PSNRN sequence jump to the starting point. That is, it is input an integer i on a register and outputs x_{in_t+1} into another register which is initially set to $|0\rangle$: $|i\rangle|0\rangle \to |i\rangle|x_{in_t+1}\rangle$. Remember that n_t, the number of time steps, is equal to the number of RNs used to generate one sample path.

The concrete implementation of these gates are presented in [1].

Then, the calculation flow is as follows:

1. Initialize all registers to $|0\rangle$ except R_S, which is initialized to $|S_{t_0}\rangle$.
2. Generate a equiprobable superposition of $|0\rangle, |1\rangle, ..., |N_{\mathrm{samp}} - 1\rangle$, that is, $\frac{1}{\sqrt{N_{\mathrm{samp}}}} \sum_{i=0}^{N_{\mathrm{samp}}-1} |i\rangle_{n_{\mathrm{PRN}}}$ on R_{samp}. This is done by operating a Hadamard gate to each of n_{samp} qubits.
3. Operate J_W to set x_{in_t+1} to R_W, where i is determined by the state of R_{samp}. These are the starting points of subsequences.
4. Perform the time evolution (9) using the value on R_W. R_S is updated from $|S_{t_0}\rangle$ to $|S_{t_1}\rangle$.
5. Calculate the payoff at time t_1 and add into R_{payoff}.
6. Operate P_{PRN} to update R_W from x_{in_t+1} to x_{in_t+2}.
7. Iterate operations similar to 4-6 for each time steps until the time reaches t_{n_t}.
8. Finally we obtain a superposition of states in which the value on R_{payoff} is the sum of payoffs in each sample path. Estimate the expectation value of R_{payoff} by methods like [6, 23]. This is an estimate for (11).

2. overview of the circuit

Schematically, the circuit which realizes the above flow is as shown in Figure 1. In the figure, the gate U_j corresponds to the j-th step of asset price evolution, that is, the j-th iteration of step 4-6 in the above calculation flow.

U_j is implemented as shown in Figure 2. P_W is already explained and the gate Payoff$_j$ calculates $f_j^{\mathrm{pay}}(S_{t_j}^{(i)})$ using R_S and adds it into R_{payoff}. In addition to these, U_j has gates $V_1^{(j)}, ..., V_{n_s}^{(j)}$, which update R_S.

The detail of $V_k^{(j)}$ is shown in Figure 3. This gate (i) checks whether the asset price is in the k-th interval $[s_{j,k-1}, s_{j,k})$, (ii) if so, update R_S using the LV in the interval, (iii) clears the intermediate output. It requires ancillary registers $R_{\mathrm{count}}, R_{S'}$ and R_g. They have $\lceil \log_2 n_t \rceil, n_{\mathrm{dig}}$ and 1 qubits respectively. At the start of $V_k^{(j)}$, R_{count} takes $|j\rangle$ or $|j+1\rangle$ and the others take $|0\rangle$. Then the detailed calculation flow is:

1. If R_{count} is j and R_S is in $[s_{j,k-1}, s_{j,k})$, flip R_g.

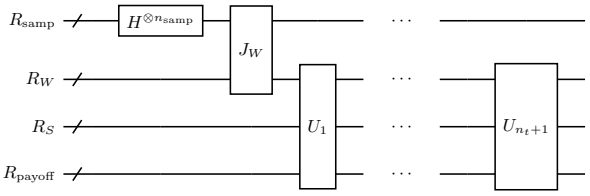

Figure 1: The overview of the circuit for asset price evolution in the LV model in the PRN-on-a-register way. Here and hereafter, ancillary qubits are sometimes omitted for simple display.

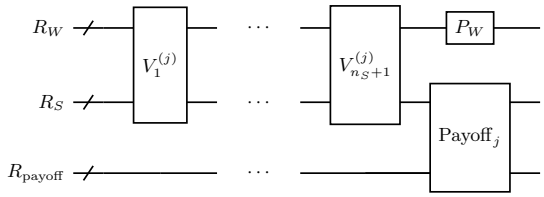

Figure 2: The overview of the U_j, which performs the j-th step of asset price evolution, in the PRN-on-a-register way.

.

2. If R_g is 1, update R_S as

$$S_{t_j} \to S_{t_{j+1}} = S_{t_j} + (a_{j,k} S_{t_j} + b_{j,k}) \sqrt{\Delta t_j} x_{in_t+j} \tag{13}$$

using the value x_{in_t+j} on R_W and add 1 to R_{count}.

3. Calculate

$$\frac{S_{t_{j+1}} - b_{j,k} \sqrt{\Delta t_j} x_{in_t+j}}{1 + a_{j,k} \sqrt{\Delta t_j} x_{in_t+j}} \tag{14}$$

into $R_{S'}$, using the value on R_S as $S_{t_{j+1}}$ and that on R_W as x_{in_t+j}.

4. If R_{count} is $j+1$ and $R_{S'}$ is in $[s_{j,k-1}, s_{j,k})$, flip R_g. This uncomputes R_g.

5. Do the inverse operation of 3.

Let us explain the meaning of this flow. First, R_{count} is necessary as an indicator of whether the j-th step of evolution has been already done or not. Without this, it is possible that the asset price is doubly updated in a time step. If and only if the j-th step has not been done, that is, R_{count} is j and the asset price is in $[s_{j,k-1}, s_{j,k})$, the update of the asset price with the LV function $a_{j,k}S + b_{j,k}$ is done. To do this conditional update, the check result is intermediately output to R_g and the gate corresponding (13) is operated on R_S under control by R_g. Besides, the increment of R_{count} controlled by R_g is also done, so that R_{count} indicates completion of the j-th step if so. Steps 3-5 is necessary to clear R_g. If the asset price has been updated in Step 2, Step 3 draws back it to the value before the update. Conversely, we can determine whether the update has been done in Step 2 from the result of Step 3. That is, for the reason mentioned soon later, the condition that R_{count} is $j+1$ and $R_{S'}$ is in $[s_{j,k-1}, s_{j,k})$ after Step 3 is equivalent to the condition that R_{count} is j and R_S is in $[s_{j,k-1}, s_{j,k})$ before Step 2. Therefore, Step 4 flip R_g if and only if it is $|1\rangle$, so it goes back to $|0\rangle$. In summary, through the sequential operation of $V_1^{(j)}, ..., V_{n_S+1}^{(j)}$, R_S is updated only once at the appropriate $V_k^{(j)}$, R_{count} is updated from $|j\rangle$ to $|j+1\rangle$ and all intermediate outputs on ancillary registers are cleared.

We here mention a restriction on the LV model so that it can be implemented in the PRN-on-a-register way. Note that through $V_1^{(j)}, ..., V_{n_S+1}^{(j)}$, the state is transformed from $|j\rangle|S_{t_j}^{(i)}\rangle$ to $|j+1\rangle|S_{t_{j+1}}^{(i)}\rangle$, where the first and second kets correspond to R_{count} and R_S respectively and other registers are omitted since they are unchanged. This means that the map from $S_{t_j}^{(i)}$ to $S_{t_{j+1}}^{(i)}$ must be one-to-one correspondence, since unitarity is violated if not. Actually, this is not so strong restriction. As shown in Appendix in [1], if we set $a_{i,j}, b_{i,j}$ so that $\sigma(t,S)$ is continuous with respect to S and we set Δt_j small enough that the increment ΔS_{t_j} is much smaller than S_{t_j} itself, the above condition is satisfied.

This one-to-one correspondence lets Step 3 work. That is, since the map between $S_{t_j}^{(i)}$ and $S_{t_{j+1}}^{(i)}$ is one-to-one correspondence, the result of Step 3 is in $[s_{j,k-1}, s_{j,k})$ if and only if the value on R_S before Step 3 is in the image of $[s_{j,k-1}, s_{j,k})$ under the map.

Although the circuit is decomposed into more smaller parts, we do not describe the further details here. [1] presents the more detailed implementation.

B. the register-per-RN way

1. *calculation flow*

Also for the register-per-RN way, we start from presenting the calculation flow, which is somewhat simpler than the PRN-on-a-register way. Again, we consider the flow until calculation of the payoff sum.

Before we present the calculation flow, we explain the required registers.

- $R_{W_i}, i = 1, ..., n_t$: This is a register for the i-th SNRN. We need such a register per random number, so the total number is n_t.
- $R_{S_i}, i = 0, 1, ..., n_t$: This is a register where the value of the asset price at time t_i is held.

- $R_{\text{payoff},i}, i = 1, ..., n_t$: This is a register where the value of the sum of payoffs by t_i is held.

We again omit ancillary registers here. For the full description, see [1]. Besides, we again assume that these and ancillary registers necessary to update them have n_{dig} qubits.

We here concretely define a superposition of SNRN values as the following state. In advance, we set the equally spaced $N_{\text{SN}} + 1$ points for discretization of the distribution $x_{\text{SN},0} < x_{\text{SN},1} < ... < x_{\text{SN},N_{\text{SN}}}$, where $x_{\text{SN},0}$ and $x_{\text{SN},N_{\text{SN}}}$ are the upper and lower bounds of the distribution and set to, say, -4 and +4, respectively. We here assume $N_{\text{SN}} = 2^{n_{\text{dig}}}$ for simplicity. Then, we define $|\text{SN}\rangle$ as $|\text{SN}\rangle = \sum_{i=0}^{N_{\text{SN}}-1} \sqrt{p_{\text{SN},i}}\,|i\rangle$, where $p_{\text{SN},i} = \int_{x_{\text{SN},i}}^{x_{\text{SN},i+1}} \phi_{\text{SN}}(x)dx$ and $\phi_{\text{SN}}(x)$ is the probability density function of the standard normal distribution. The circuit to create such a state is presented in [1]. Since $x_{\text{SN},i}$ can be easily calculated from the index i by a linear function, we identify i as $x_{\text{SN},i}$.

Then, the calculation flow is as follows:

1. Initialize all registers to $|0\rangle$ except R_{S_0}, which is initialized to $|S_{t_0}\rangle$.
2. Generate superpositions of SNRNs on $R_{W_1}, ..., R_{W_{n_t}}$. That is, set each of them to $|\text{SN}\rangle$.
3. Perform the time evolution (9) using the value on R_{W_1} as w_1. The result is output to R_{S_1} as $|S_{t_1}\rangle$.
4. Calculate the payoff at time t_1 using R_{S_1} and output the sum of it and the previous payoffs to $R_{\text{payoff},i}$.
5. Iterate operations similar to 3-4 for each time step until the time reaches t_{n_t}.
6. Finally we obtain a superposition of states in which the value on R_{payoff,n_t} is the sum of payoffs for each pattern of values of SNRNs. Estimate the expectation value of R_{payoff,n_t} to get (10).

2. *overview of the circuit*

The outline of the circuit in the register-per-RN is as shown in Figure 4. First, $|\text{SN}\rangle$ is created on each R_{W_j} by the gate SN. After that, the gate U_j performs j-th step of asset price evolution and payoff calculation. For each evolution step, ancillary registers $R_{\text{flg},j}$ and $R_{\text{LV},j}$, which have 1 and $2n_{\text{dig}}$ qubits respectively, are necessary. U_j is then implemented as Figure 5. In this gate, the sequence of comparators and "Load" gates set $a_{j,k}, b_{j,k}$ in (8) into $R_{\text{LV},j}$. Then, the operation $x \leftarrow x + (ax + b)y$ updates the asset price on $R_{S_{j-1}}$ according to (9). Since the register-per-RN way does not aim to uncompute ancillary qubits in order to reduce them, the updated asset price is output to another register R_{S_j}. At the end of U_j, the payoff is calculated. For more details, see [1].

V. ESTIMATION OF REQUIRED RESOURCES

Then, let us estimate the machine resources required for the implementation in the PRN-on-a-register way and the register-per-RN way. We consider the two metrics, qubit number and T-count, as many papers do. Following the many previous researches on elementary arithmetic on quantum computer (see references in [1]), we can estimate such metric. Referring to [1] for the detailed discussion, we here only present the result.

Qubit numbers and T-counts for the two ways are shown in Table I. Here, we have made following assumptions, some of which have been already mentioned:

- As PSNRN, we use PCG[25] combined with the inversion sampling. We use n_{PRN}-bit PCG, where n_{PRN} is large enough that the PRN sequence has good statistical property, e.g. long period. Therefore, the register which hold the PCG sequence and ancillary registers necessary for calculation of PRN sequence have n_{PRN} qubits. However, we use only top n_{dig} digits of PCG, since lower bits have poorer statistical properties.
- Other registers which store numerical numbers, R_W, R_S etc., and ancillary registers concerning them have n_{dig} qubits.
- R_{samp} has n_{samp} qubits.
- Other registers have only several qubits and we neglect their contributions to the whole qubit number.
- For the inverse cumulative distribution function of standard normal distribution, we use the piecewise polynomial approximation presented in [26] with n_{ICDF} intervals.

From Table I, wee see that, naturally, qubit number is independent from n_t in the PRN-on-a-register way but proportional to n_t in the register-per-RN way and T-count is proportional to n_t in the both ways. If we take the setting $n_{\text{samp}} = 16, n_{\text{dig}} = 16, n_{\text{PRN}} = 64, n_{\text{ICDF}} = 109, n_t = 360, n_S = 5$, which is typically necessary for practical use in derivative pricing[6], the values in Table I becomes as follows:

- qubit numbers: 2.4×10^2 for the PRN-on-a-register way and 9.2×10^5 for the register-per-RN way
- T-counts: 3.7×10^8 for the PRN-on-a-register way and 2.1×10^8 for the register-per-RN way

The total T-count is of same order of magnitude in the both way but larger for the PRN-on-a-register way by a factor 2 roughly. We here comment on the parts which consume T-count most heavily in each way. In the PRN-on-a-register way, there are two parts which contribute to T-count equally and dominantly. The first is the update of the asset price in $V_k^{(j)}$. Note that additional operations for reduction of qubits, such as inverse division in self-update multiplication and drawing back the asset price to clear R_g, increase T-count compared with the register-per-RN way. The second is modular multiplications in update of the PCG sequence. Since we

[6] $n_{\text{samp}} = 16$ corresponds to 65536 sample paths.

Table I: Qubit number and T-count required in the PRN-on-a-register and register-per-RN ways. Aiming to estimate the orders of the metrics, we take only the leading term with respect to $n_{\rm dig}, n_{\rm PRN}$ and so on.

	PRN-on-a-register	register-per-RN
qubit	$n_{\rm samp} + 2n_{\rm dig} + n_{\rm PRN} + \max\{2n_{\rm PRN}, 7n_{\rm dig}\}$	$(3n_{\rm dig}^2 + 111n_{\rm dig})n_t$
T-count	$(245n_{\rm dig}^2 n_S + 140n_{\rm PRN}^2 + 210n_{\rm dig}^2 + 56n_{\rm dig}n_{\rm ICDF})n_t$	$(7n_{\rm dig}^2 + 63n_{\rm dig} + 28n_S + 3.4\times10^4)n_{\rm dig}n_t$

must take the bit number pf PCG large enough, the T-count of operations for PCG becomes large. On the other hand, in the register-per-RN way the dominant contribution to T-count comes from arccos's in SN gates (see [1]). Because not only an arccos itself is T-count consuming but also it is used in each iteration in the SN gate, the total T-count piles up.

VI. SUMMARY

In this paper, we considered how to implement time evolution of the asset price in the LV model on quantum computers. Similar to other problems in finance, derivative pricing by MC simulation requires a large number of random numbers, which is proportional to n_t, the number of time steps for asset price evolution, and this may cause difficulty in implementation. We then considered two ways of implementation: the PRN-on-a-register way and the register-per-RN way. In the former we sequentially generate pseudo random numbers on a register and use them to evolve the asset price. In the latter, standard normal random numbers necessary to time evolution are created as superpositions on separate registers. For both ways, we present the concrete quantum circuits. For not only random number generation but also other aspects, we try to save qubit numbers permitting some additional procedures in the PRN-on-a-register way and do opposite in the register-per-RN way. We then give estimations of qubit number and T-count required in each way. In the PRN-on-a-register way, qubit number is kept constant against n_t. On the other hand, in the register-per-RN way qubit number is proportional to n_t. Each way has T-count consuming parts and the total T-counts for both ways are of same order of magnitude, expect the PRN-on-a-register way has the larger T-count by a factor about 2, in some specific setting.

Note that analyses of resources required for implementation of the LV model in this paper strongly depend on designs of elementary circuits for arithmetic. For example, in the register-per-RN way the dominant contribution to T-count comes from arccos's in preparing SNRNs. If more efficient circuits are proposed and we replace the current choice with them, required resources may change.

Finally, we note that this study is not enough for application of quantum algorithm for MC to pricing in the LV model. Although we assumed that the LV function is given, in practice we have to calibrate the LV as mentioned above. Besides, we have not considered how to evaluate terms in exotic derivatives, for example, early exercise. In future works, we will consider such things and aim to present how to apply quantum computers in the whole process of exotic derivative pricing.

[1] K. Kaneko et al., arXiv:2007.01467
[2] R. Orus et al., Reviews in Physics 4, 100028 (2019)
[3] J. C. Hull, "Options, Futures, and Other Derivatives", Prentice Hall (2012)
[4] S. Shreve, "Stochastic Calculus for Finance I & II", Springer (2004)
[5] A. Montanaro, Proc. Roy. Soc. Ser. A, 471, 2181 (2015)
[6] Y. Suzuki et. al., Quantum Information Processing, 19, 75 (2020)
[7] P. Rebentrost et. al., Phys. Rev. A 98, 022321 (2018)
[8] N. Stamatopoulos et al., arXiv:1905.02666
[9] S. Ramos-Calderer et al., arXiv:1912.01618
[10] F. Black and M. Scholes, Journal of Political Economy 81, 637 (1973).
[11] R. C. Merton, The Bell Journal of Economics and Management Science 4, 141 (1973)
[12] B. Dupire, Risk, 7, 18-20 (1994)
[13] L. Grover et. al., arXiv:quant-ph/0208112
[14] F. Arute,et al., Nature, 574, 505 (2019)
[15] E. T. Campbell et al., Nature, 549, 172 (2017)
[16] K. Miyamoto and K. Shiohara, arXiv:1911.12469
[17] M. Amy et al., IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(6): 818-830 (2013)
[18] P. Selinger, Phys. Rev. A 87, 042302 (2013)
[19] S. Bravyi and J. Haah, Phys. Rev. A 86, 052329 (2012)
[20] A. G. Fowler and C. Gidney, arXiv:1808.06709
[21] D. J. Egger at al., arXiv:1907.03044
[22] G. Maruyama, Rendiconti del Circolo Matematico di Palermo 4, 48 (1955)
[23] G. Bassard et. al., Contemporary Mathematics 305, 53 (2002)
[24] K. Nakaji, arXiv:2003.02417
[25] M. E. O'Neill, Harvey Mudd College Computer Science Department Tachnical Report (2014); http://www.pcg-random.org/
[26] W. Hörmann and J. Leydold, ACM Transactions on Modeling and Computer Simulation 13(4):347, (2003)

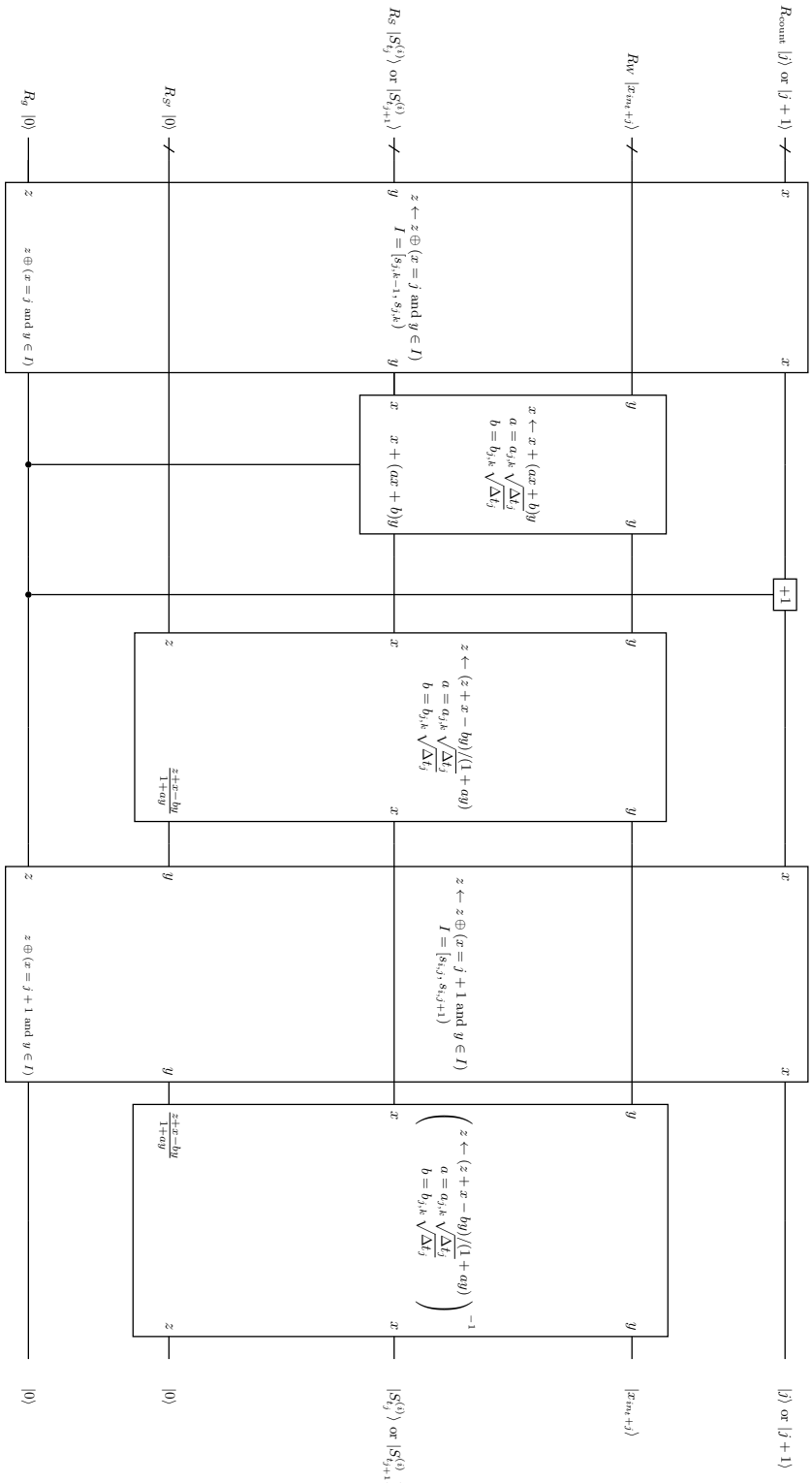

Figure 3: $V_k^{(j)}$, which updates R_S if the asset price is in the k-th grid of the LV function. Here and hereafter, the wire going over a gate means that the corresponding register is not used in the operation of the gate. A formula at the center of a gate represents the operation the gate performs and '-1' means the inverse operation. A formula beside a wire and in a gate means the input or the output of the gate.

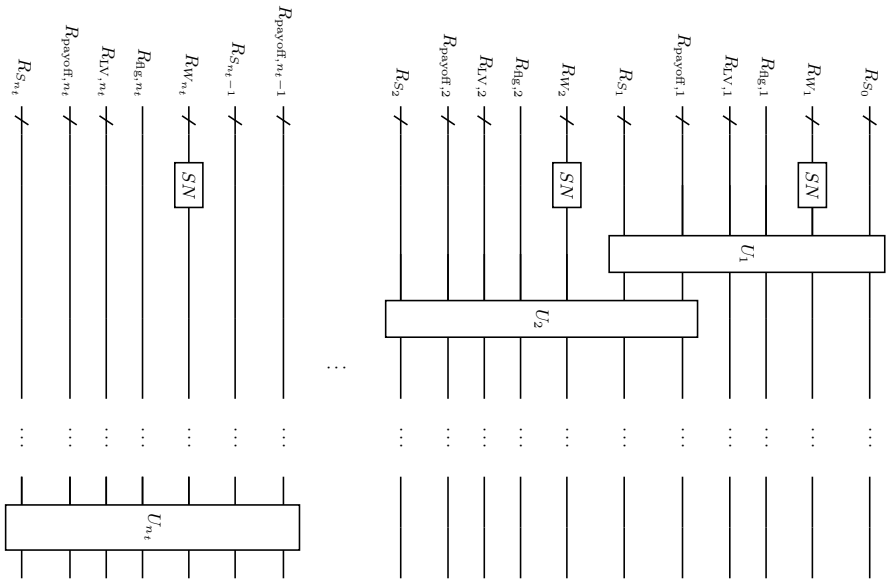

Figure 4: The overview of the circuit for asset price evolution in the LV model in the register-per-RN way.

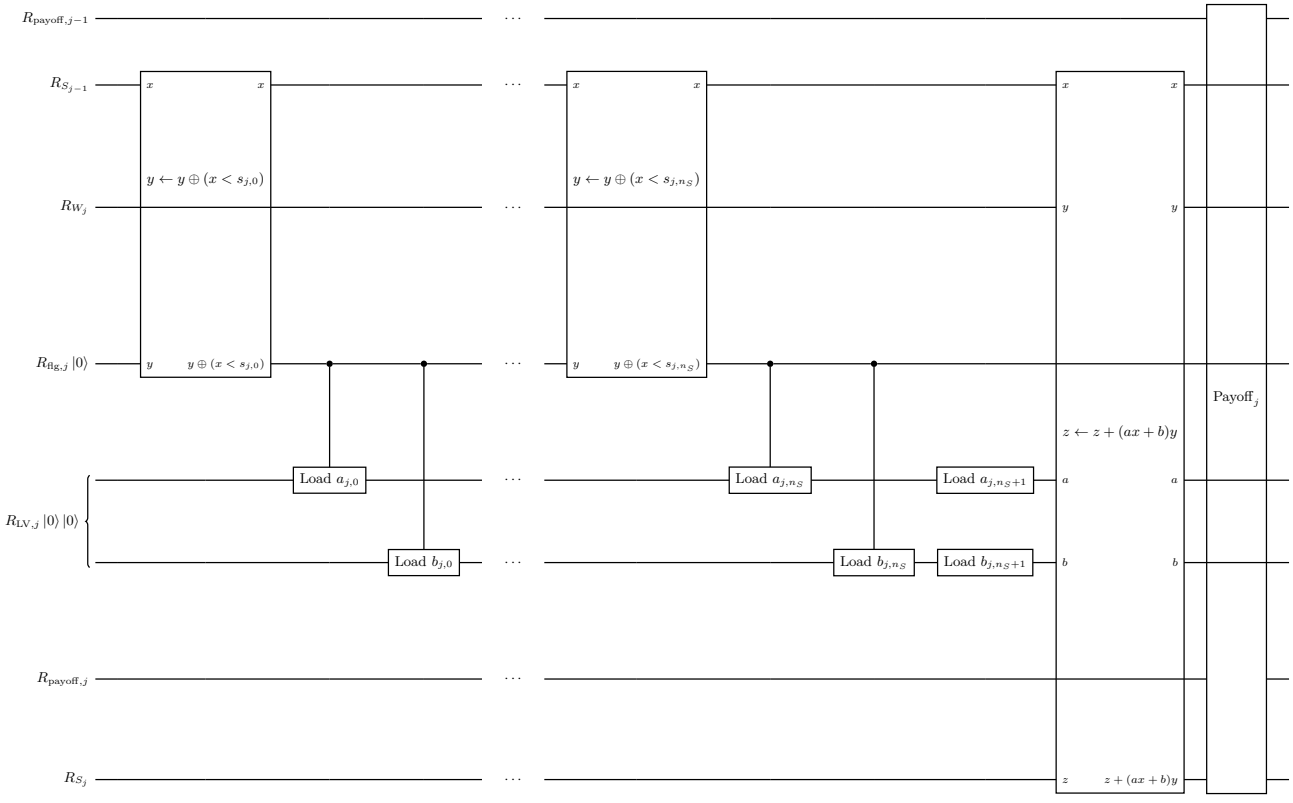

Figure 5: U_j, which performs the j-th step of asset price evolution, in the register-per-RN way.