Staff Working Paper/Document de travail du personnel—2022-29

Last updated: June 28, 2022

Quantum Monte Carlo for Economics: Stress Testing and Macroeconomic Deep Learning

by Vladimir Skavysh,[1] Sofia Priazhkina,[2] Diego Guala[3] and Thomas R. Bromley[3]

[1]Corporate Services Department
Bank of Canada
VSkavysh@bankofcanada.ca

[2]Banking and Payments Department
Bank of Canada
SPriazhkina@bankofcanada.ca

[3]Xanadu, Toronto

DOI: https://doi.org/10.34989/swp-2022-29 | **ISSN** 1701-9397

Acknowledgements

We thank Sepehr Taghavi, Juan Miguel Arrazola, Jonathan Chiu, and Matias Vieyra for fruitful discussions.

Abstract

Computational methods both open the frontiers of economic analysis and serve as a bottleneck in what can be achieved. Using the quantum Monte Carlo (QMC) algorithm, we are the first to study whether quantum computing can improve the run time of economic applications and challenges in doing so. We identify a large class of economic problems suitable for improvements. Then, we illustrate how to formulate and encode on quantum circuit two applications: (a) a bank stress testing model with credit shocks and fire sales and (b) a dynamic stochastic general equilibrium (DSGE) model solved with deep learning, and further demonstrate potential efficiency gain. We also present a few innovations in the QMC algorithm itself and in how to benchmark it to classical MC.

Topics: Business fluctuations and cycles, Central bank research, Econometric and statistical methods, Economic models, Financial stability
JEL codes: C, C15, C6, C7, C61, C63, C68, G, G21, G17, E, E13

1. Introduction

Monte Carlo methods are a group of algorithms that harness randomness to perform a number of computational tasks (Kalos and Whitlock (2009)).[1] This approach is common in various sciences that use statistics, data sampling, and mathematical complexity. When major improvements in computers began to occur, economics was among the first of the social sciences to take advantage of Monte Carlo techniques. However, many numerical economic applications are still not feasible today given classical computational limitations. The primary purpose of this paper is to explore whether the Monte Carlo methodology can be improved with the usage of quantum technology in the context of economic applications, and to determine when a quantum advantage can be achieved for these problems. Being pioneers in quantum applications in economics, we also aim to lay down the basic foundation of quantum programming so that economists can benefit from future technological innovations.

The intuition and challenges of Monte Carlo simulations can be grasped by considering a simple problem: calculation of an expectation of a random variable. Simulations of this kind are simple to perform—samples are generated according to the underlying probability distribution, the random variable is then evaluated, and an average is taken. Clearly, the error of the approximation depends on the number of samples generated, resulting in a tradeoff between accuracy and time. The tradeoff persists even when the sampling is done in parallel on a CPU/GPU cluster due to the computational cost, costs of purchasing hardware and software, and maintenance costs. In practical applications, this tradeoff can lead to the Monte Carlo estimation becoming the main bottleneck in a workflow.

Quantum computations may become a remedy for these problems. Quan-

[1]The term was first used in Metropolis and Ulam (1949) in the study of differential equations.

tum mechanics provides a new class of algorithms that can potentially outperform their classical (non-quantum) counterparts (Nielsen and Chuang (2010)). These algorithms must be executed on quantum hardware and require development of new technologies to control for excessive noise in computations, referred to as fault-tolerance. One algorithm of interest is the so-called quantum Monte Carlo (QMC) algorithm (Montanaro (2015), Xu et al. (2018), Rebentrost et al. (2018)), which has the potential to estimate an expectation value with a quadratic speedup (in query complexity) compared to the classical Monte Carlo.[2] The QMC algorithm adopts established techniques from quantum computing, including amplitude estimation (Brassard et al. (2000)), which we use in this paper.[3] Implementing QMC requires encoding the problem as a combination of unitary operators (complex matrices that preserve the inner product). These unitaries must be then decomposed into elementary quantum gates—analogous of non-quantum Boolean logical functions—for QMC to be compatible with hardware. Importantly, such decomposition must be efficient for the quadratic speedup to persist. Hence, a major focus has been on identifying efficient encodings of the Monte Carlo problem.[4]

In recent years, the most relevant use-cases developed alongside the QMC algorithm have involved solving problems in finance, including options pricing by Rebentrost et al. (2018), Stamatopoulos et al. (2020b) and Chakrabarti et al. (2021) and risk analysis by Woerner and Egger (2019) and Egger et al. (2020). However, little work has been carried out in the larger field of economics. More generally, applications of quantum computing to economics have been extremely scarce. Among the few examples of quantum-driven

[2]Note that the algorithm discussed here is conceptually different from the quantum Monte Carlo techniques used to analyze quantum many-body systems (Pang (2016).)

[3]Other methods include quantum search, as in Grover (1996), and phase estimation, as in Kitaev (1995).

[4]See Grover and Rudolph (2002), Zoufal et al. (2019), Herbert (2021a) for encoding probability distribution and Rebentrost et al. (2018), Vedral et al. (1996), Herbert (2021b), Woerner and Egger (2019), Stamatopoulos et al. (2020a) for encoding a random variable.

economics known to us are Hull et al. (2020) with the general discussion of quantum algorithms in the context of economics and specifically quantum money, Orús et al. (2019) on modeling contagion in financial networks, and Alaminos et al. on deep learning techniques for GDP forecasting. This leaves many unexplored opportunities for economists to apply quantum computing in their research and policy.

In this work we focus on two policy-relevant applications in the context of QMC: stress-testing and general equilibrium macro-modeling. For the stress testing, we develop a model where banks experience severe credit losses and engage in fire sales. The credit losses are random and determined by the stress scenario. This setup resembles a typical exercise performed by regulators and central banks to assess financial stability. The goal of the exercise is to evaluate overall capital losses that banks experience following a multi-year stress scenario. We encode this problem onto the quantum circuit and compare the quantum solution with the theoretical prediction.

For the macro-modeling problem, we solve the stochastic neoclassical growth model using machine (deep) learning. The model is intended to capture relationships between major macroeconomic variables: consumption, capital, production, and productivity shocks. A central part of the deep learning approach is the Monte Carlo estimation of stochastic gradients as in the recent work of Maliar et al. (2021). We substitute the QMC algorithm for this part and perform a fair comparison between quantum and classical algorithms by decomposing the problem into elementary gates and calculating the physical runtime. This is the first paper, as far as we are aware, comparing physical runtimes between the two algorithms.

We begin in section 2 by giving an overview of Monte Carlo applications in economics. In section 3, we briefly discuss quantum computation concepts and notation for economists unfamiliar with the subject. In section 4, we provide a mathematical formulation of the quantum Monte Carlo algorithm and describe how it can be performed on a quantum computer. This section is more technical and can be skipped by readers less interested in the details of the algorithm. Section 5 introduces two economics applications of QMC: (a) stress testing of banks and (b) solving DSGE models with deep learning. In section 6, we benchmark the QMC solution of the DSGE model against the classical Monte Carlo solution. We conclude the paper in section 7. Many mathematical details of the QMC algorithm, how QMC can be decomposed into hardware-compatible gates, and our code implementing the QMC algorithm are all provided in the Appendices.

2. Monte Carlo Methods in Economics

Monte Carlo (MC) techniques have broad applicability across the field of economics. We provide a few illustrative examples to define the scope of how QMC could help.

A common application of MC among economists is to extend the sample size of small datasets. This can be useful when a researcher relies on using methods with statistical properties that only hold asymptotically and when collecting a large sample of the data is expensive or not feasible. A significant part of these kinds of computations involves random sampling. Bootstrapping developed by Efron (1979) is a Monte Carlo technique of this sort. A simple version of this method is essentially random draws with replacement. Observations are drawn from a collected dataset with the purpose of creating N alternative data sets. The goal is then to infer the distribution of some statistic of the population data, such as mean or variance. This simple technique can be advanced in many ways (refer, for instance, to Rubin (1981) for the Bayesian analogue). In applied economics, bootstrapping also accompanies regression analysis. For instance, an economist may specify the model as a linear regression and use sampling techniques to infer the properties of the error term and the estimate. The specifics of such procedure and their impact on inference are studied in detail by econometricians (Hendry (1984) and Davidson et al. (1993) have relevant reviews).

Economists also use MC for modeling non-standard distributions and calculating their probability measures. As doing so often involves integration, economists are left with numerical methods and simulation techniques as the only remedy. Such techniques can be very sophisticated and go beyond the scope of this paper. While here we focus on independent draws, Markov Chain Monte Carlo methods applied in Bayesian econometrics provide a perfect illustration of how sampling can be done when Monte Carlo draws are auto-correlated (see Metropolis and Ulam (1949) and Hastings (1970) for the commonly used Metropolis-Hastings algorithm).

Overcoming the complexity of economic models is another reason for utilizing computational simulations. More and more, economists are relying on realistic modeling assumptions, utilizing big data sets, and including a large number of agents and interactions between them. To solve such models, computational techniques such as MC become indispensable. There are two main sources of complexity in the current models: (i) behavioral or theoretical and (ii) relationship-based or computational.

The first type of complexity comes from mathematical complexities that economists face when modeling rational behaviors of agents. For instance, the most commonly used class of macroeconomic models, called dynamic stochastic general equilibrium models (DSGE), cannot be solved explicitly apart from some trivial cases, and require fixed-point convergence algorithms for equilibrium search. Because the behaviors of agents in such models are assumed to be interdependent across time and economic sectors, these models become computationally challenging even with little heterogeneity of actions and agents. Thus, computational methods, such as MC techniques, are essential and receive much consideration from macroeconomists (e.g., see Judd (1998) for the chapters on MC methods). Later in this paper, we show how machine-learning techniques can be applied to solve a DSGE model and where QMC could play a pivotal role.

The second source of complexity arises in the agent-based models (ABM) and other computational economics models (Hommes (2006) provides a comprehensive review). Unlike in DSGE models, agents now behave myopically according to simple heuristics. Thus, the researchers are less focused on finding the equilibrium fixed point. However, the number of agents is often large, and the computational demands grow rapidly as the network of interactions between agents grows in size. Models of this kind are helpful for including bounded-rationality and heterogeneity among agents. Thus, they are less predictable in terms of computational output and intrinsically rely on the simulations. This creates even more demands for the hardware performance, as a researcher may need to run the model many times to test various calibrations and model treatments. We use an approach similar to ABM for our second example when discussing stress testing of banks.

Finally, Monte Carlo methods are essential for reducing the mathematical complexity of many economic models. The instances of this kind are too many to list. Economists tend to rely on numerical methods for calculating multidimensional integrals, finding fixed points, solving differential equations, etc. In finance alone, the applications range from option pricing to optimal portfolio selection. As such, overcoming the computational costs would improve the modeling abilities of economists.

3. Quick Introduction to Quantum Computing for Economists

3.1. Quantum advantage

Quantum computing applies the laws of quantum mechanics to perform computations. Quantum mechanics is a theory describing microscopic systems with low gravity and speeds much lower than the speed of light. Within quantum computation, numerous algorithms have been developed with proven theoretical speedup over the best available classical algorithms (Nielsen and Chuang (2010), Jordan (2022)). As quantum hardware improves, this speedup should be realised in practice. This will usher in the age of "quantum advantage" (or "supremacy") where even the best classical supercomputers cannot compete against quantum computers for certain problems.

3.2. Qubit, superposition, measurement, and entanglement

The basic unit of quantum computation is a *qubit*. Qubit is a quantum counterpart of a bit in classical computation. A classical bit, realized with the transistor, can have the value of either 0 or 1. By contrast, a quantum bit, which can be realized with a superconductive loop or a trapped ion, is a linear combination or *superposition* of these two states. Mathematically, a qubit can be described as $a\,|0\rangle + b\,|1\rangle$, where a and b are complex constants and $|0\rangle$ and $|1\rangle$ refer to states with values 0 and 1, respectively.

Superposition can be understood geometrically. A geometrical representation of a qubit is a radius vector of a sphere with radius one (see Figure 1). Poles of the sphere correspond to the basis states of the qubit: $|0\rangle$ and $|1\rangle$. A qubit is in a superposition whenever it is not in the basis states.

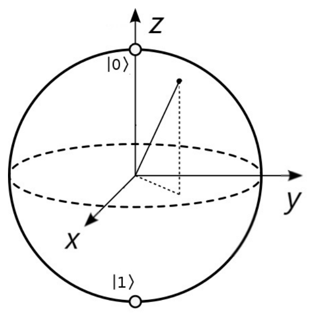

Figure 1: Geometric representation of a qubit as a complex vector of length one. Poles of the sphere correspond to observable states $|0\rangle$ and $|1\rangle$.

Whereas a qubit may initially exist in a superposition of states $|0\rangle$ and $|1\rangle$, the process of *measuring* the qubit forces the qubit to assume the value of either $|0\rangle$ or $|1\rangle$. Qubits tend to be measured at the end of a quantum algorithm to obtain the answer being sought. To explain the intuition behind the superposition and measurement, consider the famous thought experiment of Erwin Schrödinger: place a cat with a deadly radioactive element in a sealed box. In the quantum world, the cat is both dead and alive before the box is opened. It is only when the state of the cat is measured (when the box is opened) that the cat becomes "dead" or "alive."

Another important concept of quantum mechanics is *entanglement* described by Albert Einstein as a "spooky action at a distance." When two qubits are linked together, or *entangled*, action applied to one of the qubits immediately impacts the other qubit. If two qubits are entangled such that their spins are reversed, measurement of $|0\rangle$ for the first qubit would correspond to the measurement of $|1\rangle$ for the second one.

Superposition and entanglement makes it possible to manipulate an exponentially large number of states simultaneously. Whereas a single qubit is a superposition of $2 = 2^1$ states, using N qubits enables one to manipulate 2^N states simultaneously. As a result, with only 300 qubits, it is possible to operate on more states than there are number of atoms in the universe. This exponential increase in the computational power with each additional qubit is what brings computational advantage to quantum computation over classical machines.

3.3. Bra-ket notation

In quantum physics, quantum states are defined to be complex unit vectors in a Hilbert space—a vector space with an inner product that defines the distance between two vectors.

For the convenience of linear algebra notations, physicists use angle brackets ("bra-ket" notation). In particular, these brackets can be used to distinguish a vector space from its dual vector space. For instance, an arbitrary qubit can be presented as a linear combination of the two basis states

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{1}$$

where α and β are complex numbers. For vector $|\psi\rangle$, there is a corresponding covector in the dual space

$$\langle\psi| = \alpha^* \langle 0| + \beta^* \langle 1| , \tag{2}$$

where α^* and β^* are conjugate scalars to α and β. To preserve probability, quantum states are taken to be unit vectors. Therefore, the inner product of a vector and its covector yields

$$\langle\psi|\, \psi\rangle = \alpha^*\alpha + \beta^*\beta = 1. \tag{3}$$

It might be more familiar to think of the bra $\langle\psi|$ as a row vector and $|\psi\rangle$ as a column vector. Thus, the product $\langle\psi|\, \phi\rangle$ of any two quantum states $|\phi\rangle$ and $\langle\psi|$ returns a scalar while the product $|\phi\rangle\langle\psi|$ returns a matrix.

Any state of a qubit can be represented as a linear combination of the basis states. In the standard basis, the qubit will assume a value of either $|0\rangle$ or $|1\rangle$ when measured. However, an infinite number of other bases can be defined and used for measuring qubits. In this paper, we will be also using the Hadamard or x-basis:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$

We can apply usual linear algebra operations to quantum states. Moreover, the definitions above can be extended to systems of multiple qubits. We will be using the outer product $\otimes$ whenever we speak about the joint state of multiple qubits. For simplicity, we will write:

$$|0\rangle^{\otimes m} = |0\rangle \otimes ... \otimes |0\rangle$$

for m-dimensional qubit system. Note that an m-dimensional qubit system has $M = 2^m$ basis vectors, which we will denote as:

$$|1\rangle, |2\rangle, ..., |M\rangle.$$

3.4. Quantum gates and unitary operators

Quantum gate is an operation that changes a state of a qubit. It is analogous to a non-quantum Boolean logical function. Current quantum computers can implement around a dozen different quantum gates. Whenever a problem is executed on a quantum computer, it needs to be represented as a mathematical composition of the quantum gates. We define several quantum gates that we use to implement the QMC algorithm.

RX, RY, RZ gates—gates that rotate the state of a qubit around axes X, Y, Z by a given angle (refer to the sphere representation of a qubit).

CNOT ("Controlled NOT") gate—a two-qubit gate that flips the second qubit from $|1\rangle$ to $|0\rangle$ or from $|0\rangle$ to $|1\rangle$ if and only if the first qubit is $|1\rangle$. For instance,

$$CNOT(|1\rangle \otimes |1\rangle) = |1\rangle \otimes |0\rangle$$

$$CNOT(|0\rangle \otimes |1\rangle) = |0\rangle \otimes |1\rangle$$

Hadamard gate—a gate that creates a superposition if given a basis state:

$$|0\rangle \mapsto \tfrac{|0\rangle+|1\rangle}{\sqrt{2}}|0\rangle$$

$$|1\rangle \mapsto \tfrac{|0\rangle-|1\rangle}{\sqrt{2}}|1\rangle$$

Later in the paper, we will often refer to a *quantum circuit*—a sequence of quantum gates, measurements of output qubits, and other actions needed for a given task to be executed. In translating a theoretical problem onto real quantum hardware, the challenge is minimizing *circuit depth*, the number of sequential gate executions necessary to run the quantum circuit. Within the quantum circuit, a collection of qubits assigned to some computational task are referred to as a *register* (of qubits). Meanwhile, *ancilla qubits* serve more of a supporting role, enabling some specific computational goal within the circuit, such as storing the expectation value of some random variable.

In quantum computation, mathematical tasks are often performed with the help of *unitary operators*, complex matrices that preserve the inner product. The reason why unitary operators are applied to quantum states is that after the operator is applied, quantum states (unit vectors) remain unit vectors, meaning other valid quantum states. In practice, hardware is set up such that the each qubit is initialized in the zero state, $|0\rangle$. As such, it is sufficient to only describe how a unitary operates on the $|0\rangle$ state whenever a new qubit is introduced in the computation.

3.5. Transition between quantum states

Mathematical representation of the quantum world can be linked to experimental data. For a qubit in arbitrary state $\psi = \alpha|0\rangle + \beta|1\rangle$, the probability of measuring the qubit to be in state $|0\rangle$ is $|\langle 0|\psi\rangle|^2 = \alpha^2$. More generally, the probability of transitioning from state $|\psi\rangle$ to state $|\phi\rangle$ is $|\langle\phi|\psi\rangle|^2$.

For gaining more intuition about unitaries and probabilities, consider unitary $\mathcal{A}$ defined as the operator that performs transformation of initial state $|0\rangle$ to state

$$\mathcal{A}|0\rangle = \sqrt{p(0)}|0\rangle + \sqrt{p(1)}|1\rangle. \tag{4}$$

This unitary encodes the probability distribution $p(\cdot)$ defined on the basis states. The probability of transitioning from the output state $\mathcal{A}|0\rangle$ to the basis state $|0\rangle$ is exactly $p(0)$, and the probability of arriving from the output state $\mathcal{A}|0\rangle$ to the basis state $|1\rangle$ is exactly $p(1)$.

3.6. Limitations of current quantum hardware

Although controversy remains (Martin (2021)), claims of quantum advantage have recently been reported (Arute et al. (2019), Zhong et al. (2020), Madsen et al. (2022)). Despite these claims, current quantum hardware can only execute shallow circuits before the results are drowned by computational noise. This makes it impossible to execute algorithms such as quantum Monte Carlo on full-scale problems. However, low-qubit quantum circuits can be simulated on a classical computer, as we have done in this paper.

3.7. PennyLane library

Throughout this paper, code blocks are provided to show how relevant parts of the algorithm can be implemented using the PennyLane software library (Bergholm et al. (2018)). PennyLane is open-source library with thousands of contributors from around the world. In fact, the implementation of the QMC algorithm within PennyLane has been our contribution to the library. Pennylane's documentation and installation instructions are available at pennylane.ai.

4. Quantum Algorithm for Monte Carlo Simulations of Moments

This section provides a review of Monte Carlo (MC) estimation and how it can be performed on a quantum computer. Starting from the basics of MC estimation, we show how the problem can be encoded as a quantum algorithm and then subsequently sped up using amplitude estimation (Brassard et al. (2000), Montanaro (2015), Xu et al. (2018), Rebentrost et al. (2018)).

4.1. Classical Monte Carlo sampling

In practice, Monte Carlo draws are often used to evaluate moments of a random variable. We show how this can be done on a quantum computer using mean function as an example. Consider a random variable $f(\boldsymbol{x})$, defined as a measurable function of $\boldsymbol{x}$ specified on $\mathbb{R}^d$ with a probability density function $p(\boldsymbol{x})$. The expectation value of $f(\boldsymbol{x})$ can be written as

$$\mu := \mathrm{E}[f(\boldsymbol{x})] = \int_{\mathbb{R}^d} p(\boldsymbol{x}) f(\boldsymbol{x}) \ d\boldsymbol{x}. \tag{5}$$

One approach to approximating $\mathrm{E}[f(\boldsymbol{x})]$ is using MC estimation. Here and later in the paper, we assume that $f(x)$ is well-behaved with finite values and non-zero variance. To approximate $\mathrm{E}[f(\boldsymbol{x})]$, we randomly sample N points $\boldsymbol{x}_i$ according to density p and calculate the average:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{x}_i). \tag{6}$$

The estimate has an absolute error $|\mu - \hat{\mu}|$. The probability that this error is larger than a fixed $\varepsilon > 0$ can be upper bounded using Chebyshev's inequality as

$$\Pr\left(|\mu - \hat{\mu}| \geq \varepsilon\right) \leq \frac{\sigma^2}{N\varepsilon^2}, \tag{7}$$

where σ is the standard deviation of $f(\boldsymbol{x})$ (see Montanaro (2015) for details). Hence, for a constant probability, we set

$$N \propto \frac{1}{\varepsilon^2}. \tag{8}$$

Therefore, the number of samples we must generate is inversely proportional to the square of the target error. Next, we will see how the number of unitaries required in QMC scales as $1/\varepsilon$, that is, a quadratic speedup relative to that above.

4.2. Quantum Monte Carlo sampling

We now show how the Monte Carlo estimation algorithm can be carried out using a qubit-based quantum circuit. Due to the binary nature of a qubit, we first discretize the problem into a space X consisting of M grid points, with probability mass function $p(i)$ and random variable $f : X \to [0, 1]$, so that the objective is to measure the expectation value

$$\mu = \sum_{i \in X} p(i) f(i). \tag{9}$$

As before, Monte Carlo can be used to provide an estimate $\hat{\mu}$. Discretization of continuous-valued problems is an established area with a variety of approaches to define $p(i)$ above (Chakraborty (2015), Chakrabarti et al. (2021), Rebentrost et al. (2018)). Note that discretization may introduce an error $\varepsilon_{\text{disc}}$ that should be kept below the Monte Carlo error ε.

We restrict our problem to a random variable that maps to the interval $[0, 1]$ so that the problem is compatible with encoding into a quantum circuit. When the maximum f_{max} and minimum of the function f_{min} are available for the discrete space X, this restriction can be achieved by renormalizing $f(\boldsymbol{x})$ as

$$f_{norm}(\boldsymbol{x}) = \frac{f(\boldsymbol{x}) - f_{min}}{f_{max} - f_{min}}. \tag{10}$$

For building a quantum alternative of the problem, we define two unitary operators: unitary $\mathcal{A}$ for modeling function $p(\cdot)$ and unitary $\mathcal{R}$ for modeling function $f(\cdot)$ in equation (5).

For defining $\mathcal{A}$, consider a register of m qubits, so that the grid size we selected is $M = 2^m$. We define unitary $\mathcal{A}$ as the one that performs transformation of initial state $|0\rangle^{\otimes m}$ to state

$$\mathcal{A}\,|0\rangle^{\otimes m} = \sum_{i \in X} \sqrt{p(i)}|i\rangle. \tag{11}$$

This unitary encodes the probability distribution $p(\cdot)$ because the probability of arriving from the output state $\mathcal{A}\,|0\rangle^{\otimes m}$ to basis state $|i\rangle$ is exactly $p(i)$.

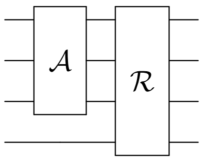

Figure 2: The $\mathcal{F}$ unitary for performing a Monte Carlo estimation. Here, the $\mathcal{A}$ unitary encodes a 2^m-dimensional probability distribution using $m = 3$ qubits and the $\mathcal{R}$ unitary encodes the expectation value μ onto the ancilla qubit.

The random variable $f(\cdot)$ is encoded by adding an additional *ancilla* qubit to output $i = \mathcal{A}\,|0\rangle^{\otimes m}$ and applying another unitary $\mathcal{R}$ that performs the transformation

$$\mathcal{R}|i\rangle|0\rangle = |i\rangle\left(\sqrt{1-f(i)}|0\rangle + \sqrt{f(i)}|1\rangle\right). \tag{12}$$

Similarly, this unitary encodes the probability distribution $f(\cdot)$, because the probability of measuring the ancilla qubit in the state $|1\rangle$ is equal to $f(i)$.

The expectation value μ can then be encoded onto the ancilla qubit by combining the $\mathcal{A}$ and $\mathcal{R}$ unitaries together according to

$$\mathcal{F} := \mathcal{R}\left(\mathcal{A} \otimes \mathbb{1}_2\right), \tag{13}$$

as shown in Figure 2.

Then the output state of the Monte Carlo problem is defined as

$$\begin{aligned} |\chi\rangle &:= \mathcal{F}\,|0\rangle^{\otimes m+1} \\ &= \sum_{i\in X} \sqrt{p(i)}\ \ |i\rangle \otimes \left(\sqrt{1-f(i)}|0\rangle + \sqrt{f(i)}|1\rangle\right). \end{aligned} \tag{14}$$

The probability of measuring the ancilla qubit in state $|1\rangle$ is given by

$$P_1 = \langle\chi|\left(\mathbb{1}^{\otimes m} \otimes |1\rangle\langle 1|\right)|\chi\rangle = \sum_{i\in X} p(i)f(i) = \mu. \tag{15}$$

In practice, we cannot obtain the probability P_1 exactly and hence the expectation value μ. Instead, we need to sample from the circuit multiple times, resulting in a binary string from which the probability of observing $|1\rangle$ can be inferred. Suppose we perform N measurements of the ancilla qubit and calculate an estimate $\hat{P}_1 = \hat{\mu} = N_1/N$, with N_1 being defined as the number of times $|1\rangle$ was measured. Since we are performing Bernoulli trials, the variance of our estimate will be

$$\mathrm{Var}(P_1) =: \varepsilon^2 = \frac{P_1(1-P_1)}{N}. \tag{16}$$

Hence, the number of trials scales with an inverse-squared relationship relative to the standard deviation ε:

$$N \propto \frac{1}{\varepsilon^2}. \tag{17}$$

This approach has successfully encoded the Monte Carlo estimation problem into a quantum circuit, but provides no speedup.

4.3. Applying amplitude estimation

In the previous section, we have encoded the Monte Carlo problem onto a register of qubits. In this section, we show how speedup can be obtained by applying the algorithm of amplitude estimation (Brassard et al. (2000)) and using an additional register of qubits to store the result.[5]

Consider the unitary operator $\mathcal{V} = \mathbb{1}_{m+1} - 2\mathbb{1}_m \otimes |1\rangle\langle 1|$ applied to the output of the Monte Carlo circuit we have so far.[6] The unitary nature of $\mathcal{V}$ allows us to write

$$\mathcal{V}|\chi\rangle = \cos(\pi\theta)\,|\chi\rangle + e^{i\phi}\sin(\pi\theta)\,|\chi^{\perp}\rangle, \tag{18}$$

[5]Subsequent works have focused on improving efficiency by searching for methods to lower the circuit depth or to remove the additional register of qubits, potentially at a cost of decreasing the speedup (Suzuki et al. (2020), Burchard (2019), Grinko et al. (2021), Giurgica-Tiron et al. (2020)).

[6]The important property of this operator is that it is Hermitian, meaning that the transpose and complex conjugate of the operator will return the operator itself.

with θ and ϕ being angles and $|\chi^{\perp}\rangle$ being a state orthogonal to $|\chi\rangle$. Hence, measuring the expectation value of $\mathcal{V}$ for the state $|\chi\rangle$ gives

$$\langle\chi|\mathcal{V}|\chi\rangle = \cos(\pi\theta) = 1 - 2\mu. \tag{19}$$

If angle θ, referred as phase, was known to us, we would use the last result to solve the Monte Carlo problem, meaning we would derive the expected value

$$\mu = \frac{1 - \cos(\pi\theta)}{2}. \tag{20}$$

Phase θ can be found through specifying another unitary $\mathcal{Q}$ with eigenvalues $e^{\pm 2\pi i\theta}$. This can be achieved with the help of Grover's diffusion operators (Grover (1996)) that perform a rotation in the subspace spanned by $|\chi\rangle$ and $|\chi^{\perp}\rangle$. Following Brassard et al. (2000) and Rebentrost et al. (2018), it can be shown that

$$\mathcal{Q} = (\mathcal{F}\mathcal{Z}\mathcal{F}^{\dagger}\mathcal{V})^2, \tag{21}$$

where $\mathcal{Z} = \mathbb{1}_{m+1} - 2(|0\rangle\langle 0|)^{\otimes m+1}$.[7]

With the help of the quantum phase estimation algorithm (Kitaev (1995)), we can estimate θ as shown in Figure 3. The algorithm introduces an additional register of n qubits ("phase estimation qubits"). These qubits control the application of the $\mathcal{Q}$ operator, which has been raised to various powers of 2^k (where $k = 1 \ldots n-1$).[8] The estimation qubits are in states of a Fourier basis. To return to the standard basis composed of $|0\rangle$ and $|1\rangle$, which can be measured as output, the inverse quantum Fourier transform is applied. This makes it possible to calculate θ by measuring the phase estimation qubits.[9] The process of measuring the phase estimation qubits

[7]Moreover, it holds that $|\chi\rangle = (|\chi^{+}\rangle + |\chi^{-}\rangle)/\sqrt{2}$, where $|\chi^{\pm}\rangle$ are the eigenstates of $\mathcal{Q}$ with eigenvalues $e^{\pm 2\pi i\theta}$, respectively (Xu et al. (2018)).

[8]This process is known as "phase kickback" (Cleve et al. (1998)).

[9]QFT is not the only choice to recover the phase. Alternatives are listed, for example, in Gómez et al. (2022).

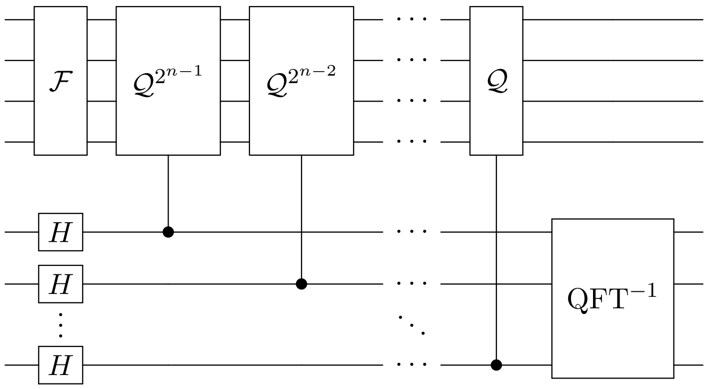

Figure 3: The QMC algorithm applies phase estimation for the unitary $\mathcal{Q}$ onto an input state prepared by $\mathcal{F}$ using n phase estimation qubits (bottom half of circuit). By sampling the phase estimation qubits in the standard basis, the eigenvalues of $e^{\pm 2\pi i\theta}$ of $\mathcal{Q}$ can be estimated.

forces each of these qubits to be either $|0\rangle$ or $|1\rangle$. This results in a binary string $\{b_1, b_2, \ldots, b_n\}$, where b_k is the measured state of the k-th estimation qubit. Analogously with the Taylor approximation, the phase θ can then be computed from this binary string using the formula

$$\theta = \sum_{i=1}^{n} \frac{b_i}{2^i}. \tag{22}$$

Knowing θ, the expected value μ can be calculated as specified in equation (20).

However, it should be remembered that the quantum processes are inherently probabilistic. As a result, it is necessary to sample the quantum circuit numerous times. This results in a distribution of binary strings (see Figure 4). The most likely of these can then be used to calculate the estimate of the phase. This concludes the summary of the quantum amplitude estimation algorithm.

From equation (22), it follows that that the error $\varepsilon_\theta = |\theta - \hat{\theta}|$ in estimating θ scales with the number of estimation qubits n as $\varepsilon_\theta \propto \frac{1}{2^n}$. Additionally, the number of applications N of the unitary $\mathcal{Q}$ is approximately 2^n. This comes from the fact that applying $\mathcal{Q}^k$ requires k sequential applications of $\mathcal{Q}$ and that the sum of all applications of Q for all estimation qubits is

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1. \tag{23}$$

Thus, $N \propto \frac{1}{\varepsilon_\theta}$. It can also be shown that the error $\varepsilon = |\mu - \hat{\mu}|$ in estimating the expectation value μ scales as $\varepsilon = \mathcal{O}(\varepsilon_\theta)$, and therefore

$$N = \mathcal{O}\left(\frac{1}{\varepsilon}\right). \tag{24}$$

Hence, the QMC algorithm provides a quadratic speedup in the number, N, of applications of $\mathcal{Q}$ (a.k.a. "oracle calls").

The implementation of the algorithm into elementary gates is given in Appendix C. For readers interested in the computational details of the QMC algorithm, the appendix shows how a quantum variational circuit can be trained to approximate the unitary $\mathcal{A}$. We also show how the controlled $\mathcal{Q}$ gate can be achieved without the controlled $\mathcal{F}$.

4.4. Simple numerical example

We now describe a simple Monte Carlo estimation problem and show how it can be solved using the QMC algorithm with a simulator in PennyLane. Suppose we have a Gaussian probability distribution with zero mean and unit variance:

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \tag{25}$$

as well as a trigonometric random variable:

$$f(x) = \sin^2(x). \tag{26}$$

The expectation value μ and variance σ^2 of $f(x)$ can be evaluated analytically as

$$\mu = \frac{\sinh(1)}{e} \approx 0.432 \qquad \sigma^2 = \frac{\sinh^2(2)}{2e^4} \approx 0.120. \tag{27}$$

The first step is to discretize the problem. We set the grid variable to

$$x_i = -x_{\max} + i \times \frac{2x_{\max}}{2^M - 1},$$

for p and f and (at the expense of abusing notation) define their values on the grid as

$$p(i) = \frac{p(x_i)}{\sum_{i \in X} p(x_i)},$$

$$f(i) = \sin^2(x_i).$$

Appendix A shows how the problem can be discretized for $x_{\max} = \pi$ using $m = 5$ qubits ($M = 32$) and $n = 6$ phase estimation qubits. The QMC algorithm can be simulated in PennyLane using the `QuantumMonteCarlo` template, as shown the appendix.

In Appendix A, we present the code that outputs the probability distribution of sampling the register of phase estimation qubits in the standard basis. This distribution is plotted in Figure 4. Different from classical estimations, the distribution of θ is not unimodal but bimodal and symmetric around 0.5. These two possibilities result because we perform phase estimation with an input state $|\chi\rangle$ that is in an equal superposition of the eigenstates $|\chi\rangle^{\pm}$ with eigenvalues $e^{\pm 2\pi i \theta}$. If we expect $\theta > 0.5$, we focus on the right-hand side of the distribution; otherwise, if $\theta \leq 0.5$, we focus on the left-hand side. If this bimodal property creates many difficulties, the problem can be re-specified such that θ is always larger than 0.5 by suitably renormalizing $f(i)$. This results in a loss of accuracy, but this can be compensated for by adding another qubit to the phase estimation register.

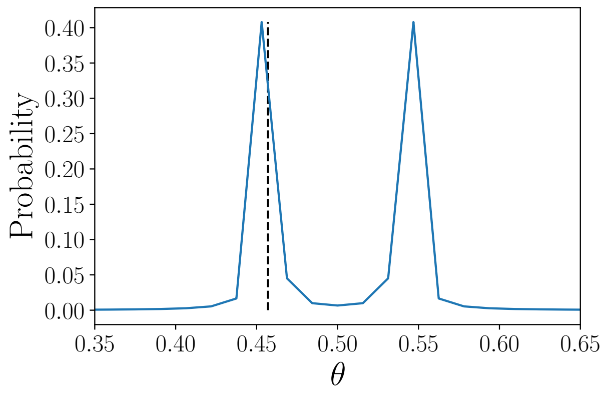

Figure 4: Estimating θ using the QMC algorithm in PennyLane. The blue line indicates the probability of estimating a given value of θ, which is derived from the bitstring probabilities according to equation (22). The dashed vertical line shows the theoretical value of $\theta = 0.457$. The second peak can be removed by renormalizing $f(i)$, as described in section 4.3.

5. Two QMC Applications in Economics

In this section, we elaborate on two problems: stress testing of banks and deep learning applied to dynamic stochastic general equilibrium (DSGE) macroeconomic models. We first give a brief introduction to each problem and then show how the QMC algorithm can be applied, presenting outcomes from small-scale instances of the problems.

5.1. Stress testing of banks

We begin by performing a stylized macro-prudential stress test of banks given a dynamic stochastic stress scenario. Such stress tests are usually performed by a macroprudential regulator or a central bank to determine whether the financial industry can withstand large economic shocks and propagate the systemic risk to the real economy. The exercise typically begins with a specific narrative. It is then quantified with the time series projections of key macroeconomic and financial variables, such as inflation, GDP, interest rates, unemployment rate, etc. For simplicity, we will call such projections the macro scenario. The scenario is then used to quantify realized losses and expectations shocks that banks face on loans, securities, and funding exposures. We will refer to these inputs as the balance sheet shocks.

Monte Carlo simulations can be suitable for a stress test exercise for two main reasons. First, for a given time point and loss type, a stress designer may prefer to use a distribution of inputs rather than a single-value projection when creating the balance sheet shocks. In this paper, we focus on simulating probabilities of defaults of loans—a critical input to any stress test of banks. This means the macro scenario would be used for the projection of the distribution of defaults of loans rather than just their realized values. Multiplicity in the inputs allows for multiplicity of stress-test outputs, which enriches the interpretations of how economic and financial vulnerabilities may respond to the stress. In countries with only a few stress episodes in the past, such as Canada, simulations in the form of bootstrapping can be a natural way of accounting for the under-representation of high probabilities of default in the historical sample. Similarly, distribution can be introduced around the income projections of banks. For the results of a real stress-test application with bootstrapping of this kind, see Duprey et al. (2018).

The second reason why Monte Carlo simulations are suitable for stress tests is that they allow for the modeling of complex behaviors of banks observed in real financial markets. This is specifically important for the macro- rather than micro-prudential stress tests, with former ones focusing on the externalities that financial institutions create when responding to stress. Different central banks tend to include different behavioral responses of banks and other financial institutions in their stress tests (see Farmer et al. (2022)). The behavioral adjustments utilized the most are sales of securities for the purpose of maintaining capital or liquidity positions of banks. When additional financial mechanisms are added to the model, such as funding cost tightening and contagion of defaults, it often becomes difficult to find an explicit solution for the model. This happens because of the complex

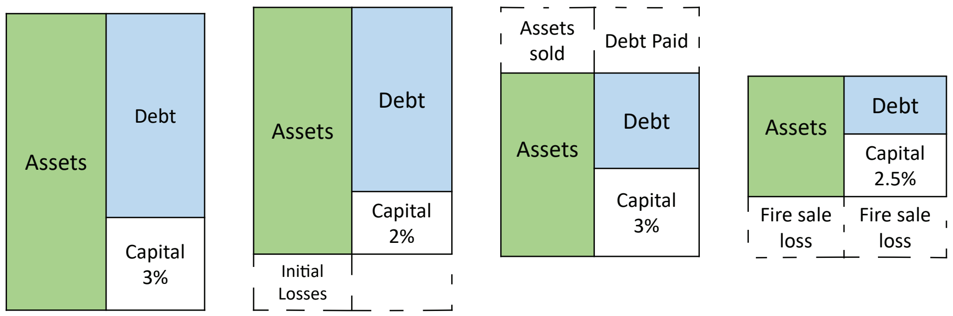

Figure 5: Stylized evolution of the balance sheet of a bank following a financial shock. A sudden drop in the price of the assets reduces the bank's leverage ratio below the regulatory requirement. The bank sells assets to restore its leverage ratio. However, the rapid selling has an effect on prices of the assets, again reducing the leverage ratio and creating capital losses.

side-effects that the behaviors of financial institutions have on each other. While some behavioral stress-testing models provide explicit equilibrium solutions without the need to utilize Monte Carlo techniques, such as Hałaj and Priazhkina (2021), the majority of models continue to rely on Monte Carlo methods for two reasons. First, development of stress-testing models with theory-powered predictions is time-consuming and requires precise calibration of parameters. Second, Monte Carlo techniques may still be beneficial to parallel the scenario inputs and perform sensitivity analysis. This highlights the relevance of our application for many central bank models.

To illustrate how QMC might be applied to stress testing, we calculate the expected capital losses of two banks over the span of two time periods of a stress scenario.[10] Without loss of generality, assume that before stress is applied, bank i holds assets a_i^{all}, out of which mortgages constitute a_i^m, business loans a_i^b, and securities a_i^s. The bank partially funds itself with equity e_i; the rest is funded with long-term debt.

The exercise is performed period-by-period with the following sequence

[10]The current quantum current algorithm can easily manage more banks and more periods, albeit at the cost of additional qubits.

of events within each period t. At the beginning, bank i faces credit losses

$$loss^{cr}_{i,t} = a^b_i d^b_t + a^m_i d^m_t \tag{28}$$

with default rates for mortgages and business loans

$$d^m_t = d^m_{t-1}(1 + d_t)$$

$$d^b_t = d^b_{t-1}(1 + d_t).$$

Each period, more loans default at random, depending on the realization of the random variable d_t.[11]

We assume that pre-stress, banks satisfy the regulatory leverage requirement to hold share β of capital relative to total assets. After credit losses are applied, banks fall below their leverage requirement and are forced to adjust their balance sheets to come back to the required ratios. They do so differently in the short-term and in the long-term.[12]

In the short-term, each bank i sells securities in the amount $\Delta a^s_{i,t}$, sufficient to restore the leverage ratio to threshold β. Because securities need to be sold quickly, this impacts the securities' market price and produces fire sales losses for the bank. For simplicity, we assume a linear price-response function:

$$\Delta p_t = -\alpha \sum_{i=1}^{2} \Delta a^s_{i,t}. \tag{29}$$

Remaining securities on the balance sheets of all banks are also subject to re-evaluation due to mark-to-market accounting (see Figure 5 for the balance sheet changes). As such, banks impose externalities on each other by selling securities simultaneously, and their capital ratios may fall below

[11] The difference between generating stochastic probabilities of default and credit losses is subtle. With the assumption that loss-given-default and utilization rate are both equal to one, the credit loss rate can be assumed to be equal to the probability of default.

[12] This split is an artificial modeling tool rather than a natural sequential order of actions.

the regulatory requirement. This concludes the short-term response. In the long-term stage of each period, bank i receives income and recapitalizes the equity to the initial level. The bank also restores the original portfolio of assets and liabilities. Capital injection may come from private sources or government support. Long-term adjustments of this kind are less typical for macro-prudential stress testing. Here, we introduce them for computational simplicity. Without going into the specifics of such costs, we assume that the regulator still cares about the capital losses of the first period.

The changes between the initial and re-optimizing balance sheet can be found explicitly by doing some basic algebra. First, the amount of securities liquidated by each bank can be calculated as

$$\Delta a_{i,t}^{s} = \frac{c_{i,t} - g_i c_{j,t}}{1 - g_i g_j}$$

with coefficients $c_{1,t}$ and $c_{2,t}$ being defined as linear function of initial credit losses:

$$c_{i,t} = \frac{e_i - (1-\beta) loss_{i,t}^{cr} - \beta a_i^{all}}{(1-\beta)\alpha a_i^s - \beta}, \tag{30}$$

and coefficients g_1 and g_2 being independent of the scenario:

$$g_i = \frac{\alpha(1-\beta) a_i^s}{(1-\beta)\alpha a_i^s - \beta}. \tag{31}$$

The function of interest, total one-period loss as a fraction of industry assets, is thus linear in the default rates d_t^m and d_t^b :

$$\frac{loss_{1,t}^{cr} + loss_{2,t}^{fs} + loss_{1,t}^{cr} + loss_{2,t}^{fs}}{a_1^{all} + a_2^{all}}. \tag{32}$$

Therefore, computing the total expected system-wide loss over T periods is equivalent to calculating the polinomial operator of random draws d_t:

$$E_0[loss_T^{\%}] = E_0[\gamma_1 \left(1 + \gamma_0 \frac{1-\beta}{\beta}\right) \sum_{t=1}^{T} \prod_{\tau=1}^{t} (1 + d_\tau)], \tag{33}$$

where γ_0 captures fire sales impact

$$\gamma_0 = \alpha \frac{(1-g_1)(1-g_2)}{1-g_1 g_2}(a_1^s + a_2^s)$$

and γ_1 captures credit risk impact

$$\gamma_1 = \frac{(a_1^b + a_2^b)d_0^b + (a_1^m + a_2^m)d_0^m}{a_1^{all} + a_2^{all}}.$$

Table 1: Stress test model calibration parameters.

		Bank 1	Bank 2
Mortgages	a^m	50	30
Business Loans	a^b	50	70
Securities	a^s	50	50
Total assets	a^{all}	150	150
Equity	e	4.5	4.5
		System-wide	
Price sensitivity	α	0.0005	
Leverage ratio	β	0.03	
Benchmark mortgage loan loss	d_0^m	0.5%	
Benchmark business loan loss	d_0^b	1.5%	
Monte Carlo credit loss rate	d	Beta(2,10)	
Fire sales term	γ_0	0.0060	
Credit risk term	γ_1	0.0053	

We now discuss how this problem can be tackled with the QMC algorithm for two periods of stress with parameters as in Table 1. We specifically choose non-symmetric distribution of loss rates (Beta(2,10)) to account for fat tails typical for systemic risk events and to illustrate how the quantum methods can be applied to non-Gaussian distributions.

After substituting the parameters, equation (33) reduces to finding the expectation of

$$0.0064 \times (2 + d_2) \times (1 + d_1). \qquad (34)$$

For this we need to find $\mathcal{F}$ unitary that encodes the probability distribution—in our case $Beta(2, 10)$—and the random variable—equation (34). As a first step, we see that the problem can be written as a T-dimensional expectation value as in equation (5), with a product distribution over the independent d_t. We can hence discretize the input d_t values each into $M = 2^m$ points so that there will be T registers of m qubits, and apply the $\mathcal{A}$ unitary to each register to prepare the appropriate distribution (see Appendix C for more details). A schematic for the procedure is provided in Figure 6 for $T = 2$ periods.

Our next step is to provide access to the random variable given in equation (33) from which the statistics are calculated. This can be achieved using the quantum arithmetic approach discussed in Appendix C, that is, so that each register represents a fixed-point number and registers can be operated upon and combined to find the random variable. We first adjust the $\mathcal{A}$ unitaries to shift the mean by 1 or 2. We can then apply a multiplication operation that multiplies the two registers, as shown in Figure 6. Note that quantum arithmetic operations may require additional registers of calculation qubits due to the reversible nature of quantum computing. These registers are omitted here, and we assume that the output register of the calculation is situated on the bottom wire in the diagram.

To finish calculating the random variable, we apply operations that multiply by constant. Finally, the square root and arcsine operations are applied and the result is imprinted onto the ancilla qubit using a controlled-Y cascade. The controlled-Y cascade places the ancilla into the state $\cos(i)|0\rangle + \sin(i)|1\rangle$, whereas the goal is to place the ancilla into the state $\sqrt{1-i}\,|0\rangle + \sqrt{i}\,|1\rangle$. This is why the cascade is preceded by the square-root and the arcsine operations. The work of Chakrabarti et al. (2021) provides a summary of decompositions for typical operations, including multiply, square root, and arcsine.

With $\mathcal{F}$ defined, the QMC algorithm can then be carried out by con-

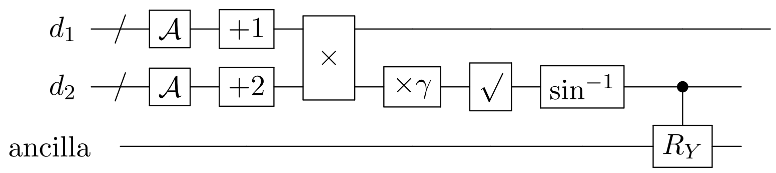

Figure 6: A schematic circuit for the $\mathcal{F}$ unitary of the stress testing problem when estimating the total system-wide loss after $T = 2$ periods. Note that the d_t lines represent registers of qubits and that $\gamma = \gamma_1 \left(1 + \gamma_0 \frac{1-\beta}{\beta}\right)$.

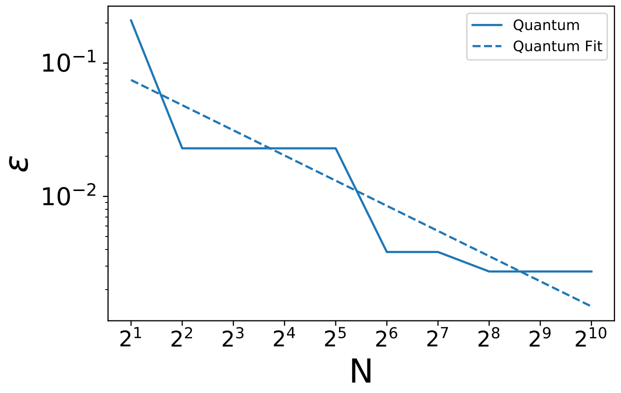

Figure 7: The error of estimating the total loss of the banking industry as a fraction of the actual (theoretical) value versus the number of oracle applications within the QMC algorithm.

structing $\mathcal{Q}$ according to the decomposition in Appendix C. The results of this are shown in Figure 7.

With 10 estimation qubits, QMC finds the total banking industry loss over two periods as a fraction of total assets to be 1.622%, as compared to the theoretical value of 1.618%. This corresponds to the error of 0.0027 as a fraction of the theoretical value. However, even just 2 estimation qubits give a good approximation, resulting in the fractional error of 0.023.

We have hence built up the protocol for an example instance of stress testing using quantum computation. Notably, we find that the total number of (logical) qubits required for this problem is manageable in theory. However, in practice, the current quantum hardware is noisy. The noise begins to dominate the results after around 100 gates (Lubinski et al. (2021)). Meanwhile, our problem requires at least $10^4 - 10^5$ gates, as we will show in the next section.

5.2. Deep learning solutions of DSGE models

Another promising application of the QMC algorithm within economics is solving DSGE models with deep learning as in Maliar et al. (2021), Fernández-Villaverde et al. (2019), and Azinovic et al. (2022).[13] DSGE models are used in macroeconomics to study relationships between aggregated economic variables such as inflation, gross domestic product, consumption, and capital goods, often in the context of government policies. Standard methods of solving such models suffer from the curse of dimensionality: computational time grows exponentially with the number of state variables and can become unmanageable beyond a small number of variables. By contrast, methods based on deep neural networks can break this curse (Bach (2017)) and are now being applied to solve large economic models (see, for instance, Lepetyuk et al. (2020)).

Monte Carlo simulations play an essential role within these deep learning approaches. Monte Carlo draws provide an unbiased estimator of the stochastic gradient with respect to all variables, making it possible to simultaneously approximate the decision function and to integrate with respect to future shocks. The shortcoming of the Monte Carlo simulations is the low square-root convergence of the solution (Maliar et al. (2021)). The quadratic speedup offered by quantum Monte Carlo could overcome this shortcoming.

To get an understanding of when quantum Monte Carlo might offer an advantage over the standard approach for this application, we consider a simple neoclassical stochastic growth model (Maliar and Maliar (2015), Stokey et al. (1989)). In this model, the representative agent starts each period with capital, k, and current level of productivity, z. The agent's

[13]For an introduction into deep learning, see Goodfellow et al. (2016).

task is to decide how much to consume right away, c, and how much capital to leave for next period, k'. The capital will be invested into production, which will deliver future payoff according to the production function $f(k)$. Part of the capital δ will depreciate, while the rest can be re-used. Output of production will also vary due to the randomness in the productivity of the agent. The future level of productivity, z', depends on the current level of productivity, z. For simplicity, assume that agents value their current consumption according to the utility function $u(c)$ and the flow of their future consumption according to the discounting factor β. This means that the agent computes the value function from the following Bellman equation:

$$V(k,z) = \max_{c,k'} u(c) + \beta E[V(k',z')] \tag{35}$$

subject to:

$$c + k' = zf(k) + (1-\delta)k \tag{36}$$

$$\ln z' = \rho \ln z + \sigma\epsilon, \quad \epsilon \sim \mathcal{N}(0,1) \tag{37}$$

where u and f are strictly increasing, continuously differentiable, and concave; $\beta \in (0,1)$; $\delta \in (0,1]$; $\rho \in (-1,1)$; and $\sigma \geq 0$. Under these assumptions, the problem has a unique solution (Stokey et al. (1989)).

The general algorithm for solving this problem with deep learning proceeds as follows:

1. Interpolate $V(k,z)$ by a neural network with learnable parameters $\{s_i\}$. In the current work, the interpolation part is treated completely classically. However, in the future, a quantum variational algorithm (Cerezo et al. (2021)) could be a viable alternative to the classical neural network.
2. Draw N_s random samples of (k, z, z_1', z_2'), where $k \sim \mathcal{U}_{[k_{min},k_{max}]}$, $z \sim \mathcal{N}(1,\sigma^2)$, and z_1' and z_2' are two possible realizations of the random process (37).

3. For each sample, use the first-order conditions and the envelope condition to compute consumption c:

$$u_c(c) = \frac{V_k(k,z)}{zf_k(k) + 1 - \delta} \tag{38}$$

$$\Rightarrow c = u_c^{-1}\left(\frac{V_k(k,z)}{zf_k(k) + 1 - \delta}\right) \tag{39}$$

where $u_c(c) = \frac{\partial u(c)}{\partial c}$, $V_k(k,z) = \frac{\partial V(k,z)}{\partial k}$, $f_k(k) = \frac{\partial f(k)}{\partial k}$.

4. Solve for k' using the budget constraint. (36)

5. Compute the Bellman error for each next-period productivity z_i':

$$B_E(k,z,z_i') = u(c) + \beta V(k', z_i') - V(k,z). \tag{40}$$

6. Calculate the mean squared error (MSE) over the entire Monte Carlo sample, defined as

$$\frac{1}{N_s}\left|\sum_j^{N_s} B_E(k_j, z_j, z_j'1) B_E(k_j, z_j, z_j'2)\right|. \tag{41}$$

7. If the MSE is less than the pre-defined threshold, stop the computation. Otherwise, update (via backpropagation or parameter-shift rule (Crooks (2019)) parameters $\{s_i\}$ to minimize the MSE.

Solving this problem on the quantum computer is currently beyond the capabilities of the quantum hardware. However, by simplifying the problem and expanding it in terms of quantum gates, it is possible to find where (for various quantum gate times) QMC gains an advantage over classical MC. Toward this end, we make the standard choice of the utility and production functions:

$$u(c) = \begin{cases} \frac{c^{1-\theta}-1}{1-\theta} & \theta \neq 1 \\ \ln c & \theta = 1 \end{cases} \tag{42}$$

$$f(k) = k^\alpha. \tag{43}$$

Furthermore, we make a simplifying choice of $\theta = 1$, and $\delta = 1$. With this simplification, the problem has an analytical solution:

$$V(k, z) = R \ln k + S \ln z + Q \tag{44}$$

where $R = \frac{\alpha}{1-\alpha\beta}$, $S = \frac{1+\beta R}{1-\beta\rho}$, and $Q = \frac{\ln(1-\alpha\beta)}{1-\beta} + \frac{\beta R \ln(\alpha\beta)}{1-\beta}$. This analytical solution can be obtained by a neural network composed of a single linear layer (with weights s_1, s_2, and bias s_0), as long as the input variables k and z are first transformed by the logarithm function. This makes it possible to easily express the Bellman error in terms of the input random variables k, z, z'_1, z'_2. Leaving the solution of the full problem for later work when quantum computation technology is more advanced, we take k and z to be constant within each random Monte Carlo sample. Moreover, we replace equation (41) with the following loss function:

$$\frac{1}{N_s} \left| \sum_{j}^{N_s} B_E(k_j, z_j, z'_j) \right| . \tag{45}$$

This loss function has a minimum at the same place as the original MSE loss for this particular problem, allowing us to only keep a single z' random variable. As a final simplification, since $z' \approx 1$, we assume $\ln z' \approx z' - 1$.

Interpolating the value function in terms of the chosen neural network architecture,

$$V(k, z) = s_1 \ln k + s_2 \ln z + s_0 \tag{46}$$

and with the simplifications above, the chosen loss function (45) reduces to

$$\left| \frac{1}{N_s} \sum_{j=1}^{N_s} (C_1 + C_2 z'_j) \right| \tag{47}$$

where

$$C_1 = (s_1 - \alpha(1+\beta s_1))\ln(k) + (s_2 - (1+\beta s_1))\ln(z) \tag{48}$$
$$+(1-\beta)s_0 - \ln\left(\frac{\alpha}{s_1}\right) - \beta s_1 \ln\left(1 - \frac{\alpha}{s_1}\right) + \beta s_2$$

$$C_2 = -\beta s_2 \tag{49}$$

The simplified neoclassical model is therefore expressed as the expectation value of a linear random variable with respect to the normal distribution. This provides us with a simple, application-relevant use-case to investigate and benchmark the QMC approach. Our first step is to set the values $C_1 = 30$ and $C_2 = -29$. The exact choice of these constants is not important, but we pick these values since they are close to the optimal values (as given in equation (47)) and could realistically occur during the training of the neural network. The distribution of the random variable z' is normal with mean $\mu = 1$, and we choose $\sigma = 0.02$. We then implement the QMC algorithm as outlined in Appendix C. The details of our QMC implementation for this problem can be found in Appendix D.

The QMC algorithm provides an estimate of $\hat{\theta}$ that can be converted into an estimate of $\hat{\mu}$ using equation (19) and accounting for the normalization of $f(x)$. Figure 8 illustrates how the error $\varepsilon = |\mu - \hat{\mu}|$ scales with the number N of applications of $\mathcal{Q}$.

We plot the error in two cases: (orange line) when using the linear approximation detailed in Appendix D, and (blue line) when using an exact simulation of $\mathcal{F}$. The linear approximation has a worse theoretical speedup, but does not require as many quantum resources and so can be realized sooner in practice. The plot shows different scaling in each case, as expected. The linear fit in log-log scale gives a slope of $\alpha = -0.642$ in the approximate case (theoretical value: $-\frac{2}{3}$) and a slope of $\alpha = -0.996$ in the exact case

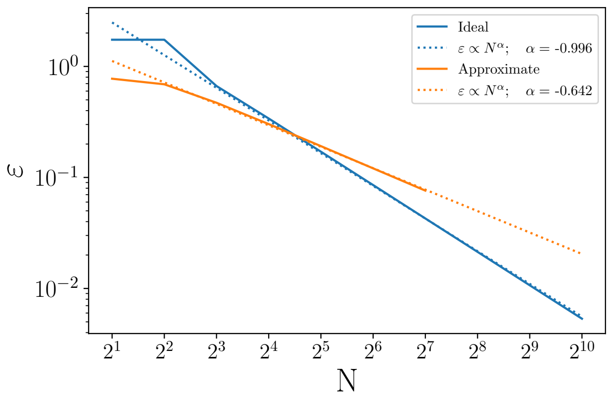

Figure 8: Finding the estimation error ε when using the QMC algorithm with a time complexity of N. The blue line shows the ideal case when the unitary $\mathcal{F}$ can be realized exactly, while the orange line illustrates the use of the linear approximation outlined in these two papers: Egger et al. (2020), Stamatopoulos et al. (2020b). The dotted lines show a linear fit in log-log scale

(theoretical value: -1), both of which are better than the scaling of classical Monte Carlo estimation, $\alpha = -\frac{1}{2}$.

6. Benchmarking of Quantum and Classical Methods

We have established that the QMC algorithm has a speedup in terms of oracle calls N when compared to its classical counterpart. However, in the QMC algorithm, N is the number of applications of $\mathcal{Q}$, while in classical Monte Carlo N is simply the number of samples drawn. To provide a fair comparison between the two approaches requires working out the real time requirements for both implementing $\mathcal{Q}$ and for drawing a sample from a classical random number generator. In this section we provide a time-based benchmark of QMC using the setting of the macroeconomic deep learning problem from the previous section.

Our first step is to understand the resource requirements of the QMC algorithm for varying numbers n of phase estimation qubits. Using the decompositions discussed in Appendix C, it is possible to break down the QMC algorithm into elementary gates that are compatible with hardware

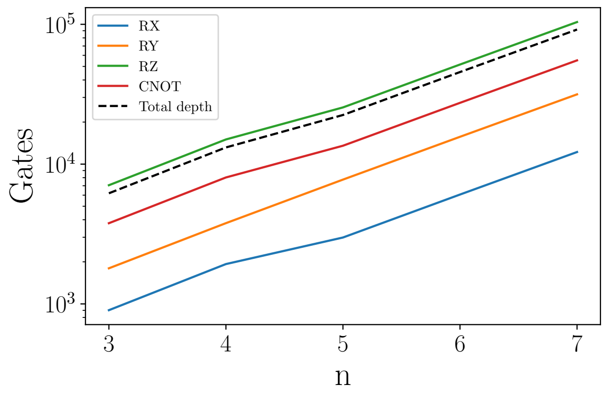

Figure 9: Counting the number of elementary gates required to implement the QMC algorithm (approximate $\mathcal{F}$) for varying numbers of phase estimation qubits n. The circuit depth is also drawn as a dashed line.

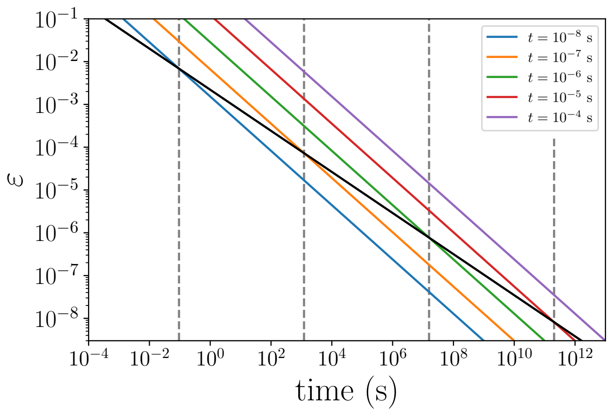

Figure 10: Providing a time-based comparison between the QMC algorithm for a variety of gate times t (colored lines) and classical Monte Carlo estimation (black line) for the macroeconomic model discussed in section 5.2. The estimation error ε is plotted against the algorithm time T_{tot}. These quantities are extrapolated for the QMC algorithm using the linear approximation of $\mathcal{F}$ for a range of phase estimation qubits n. The black line shows a classical extrapolation using one CPU core. Dashed vertical lines illustrate where the quantum and classical algorithms coincide for each gate time t.

implementation. The code for doing so can be found in Appendix D. We choose a gate set composed of the single qubit rotations R_X, R_Y, and R_Z, as well as the two-qubit CNOT gate. Figure 9 provides a gate count for implementing the QMC algorithm for the macroeconomic problem using the linear approximation with $m = 5$ discretization qubits and a range of phase estimation qubits n. We focus on the approximate case because it requires significantly fewer quantum resources than the exact case and so is likely to be realized in practice first. As expected, the gate counts scale exponentially since the number of applications of $\mathcal{Q}$ is 2^n.

As well as counting individual gates, we can also calculate the depth of the QMC algorithm, that is, the longest sequential path of gates through the circuit. The depth is also shown in Figure 9 as the dashed line and can be extrapolated using a log-linear fit to arbitrary n.

From the depth d, we can calculate the algorithm time T_{tot} by assuming each gate can be run in time at most t and finding $T_{\text{tot}} = td$. We consider gate times ranging from 10 ns to 100 μs, but it is not unreasonable to expect gate times on the order of 100 ns or lower based on the latest experimental research (Satoh et al. (2022)).

To obtain the plot of the error ε vs total time T_{tot}, recall that the fit in Figure 8 can be used to find the error scaling of the approximate approach for a given number of estimation qubits n. The number of estimation qubits is related to the circuit depth (Figure 9), which is related to the total time. This leads to Figure 10, where we plot ε against T_{tot} for a variety of gate times. This figure provides the basis for a time-based comparison of the QMC algorithm to standard Monte Carlo estimation.

To generate classical Monte Carlo results in Figure 10, we find the mean average error of estimating the same expectation value as the one obtained using a range of shot numbers N. We also record the time taken for each N and extrapolate using a linear fit in log-log scale. The result is shown as the black line the figure. In terms of computational resources, we use one CPU core. More CPUs or GPUs would be beneficial to the classical solution.[14]

[14]For completeness, Appendix E shows how this problem can be implemented in PyTorch, which can be used to run this problem on multiple CPUs or GPUs, for example, within

t (s)	T_{tot} (s)	ε	n	N_q	N_c
10^{-8}	9.40×10^{-2}	6.86×10^{-3}	15	3.28×10^{4}	3.03×10^{7}
10^{-7}	1.22×10^{3}	7.27×10^{-5}	25	3.36×10^{7}	1.85×10^{13}
10^{-6}	1.58×10^{7}	7.69×10^{-7}	36	6.87×10^{10}	1.13×10^{19}
10^{-5}	2.04×10^{11}	8.15×10^{-9}	47	1.41×10^{14}	6.88×10^{24}
10^{-4}	2.65×10^{15}	8.62×10^{-11}	58	2.88×10^{17}	4.20×10^{30}

Table 2: The QMC algorithm outperforms classical Monte Carlo sampling when T_{tot} exceeds a minimum amount. This table details the minimum T_{tot} and the corresponding estimation error ε for a range of gate times t in the QMC algorithm. The number of phase estimation qubits n is also provided, as well as the number N_q of applications of $\mathcal{Q}$ in the QMC algorithm and the number N_c of samples N in the classical algorithm.

However, the underlying scaling law of the classical MC algorithm would remain unchanged. This means that quantum advantage for this problem could eventually be achieved, albeit at a higher number of estimation qubits.

Figure 10 illustrates the expected speedup provided by the QMC algorithm due to the steeper slope of the extrapolated line.[15] The choice of individual gate time t sets the offset height of each line for QMC, which has the practical implication of determining when the QMC algorithm becomes preferential to its classical counterpart. A smaller t results in the QMC line crossing the classical line at a lower value of total algorithm runtime T_{tot}. The crossover points are shown as dashed vertical lines in the figure and are summarized in Table 2.

We therefore see that the gate time t is an important indicator for when the QMC algorithm will become performant. By pushing down the gate

the settings of federated learning (Konečný et al. (2015)).

[15]The QMC algorithm performance could also be improved on the algorithmic level. For example, using the linear approximation of Egger et al. (2020) and Stamatopoulos et al. (2020b) results in a negative effect on scaling, shifting from an ideal $\alpha = -1$ to $\alpha = -2/3$, when $\varepsilon = N^{\alpha}$. This may be remedied by considering quantum arithmetic or alternative methods (Herbert (2021b)). Finally, the QMC algorithm could also be adapted to replace the standard amplitude estimation part, requiring a register of phase-estimation qubits and an inverse quantum Fourier transform. This has been suggested in Suzuki et al. (2020), Burchard (2019), Grinko et al. (2021), Giurgica-Tiron et al. (2020).

time, we are able to outperform classical Monte Carlo sampling at lower algorithm runtimes T_{tot} and also require fewer phase estimation qubits and a lower depth N_q.

It is also fruitful to compare the two algorithms for a fixed value of runtime T_{tot} or a fixed target error ε. For example, it is common to run a complex Monte Carlo estimation overnight to achieve the greatest accuracy. Consider a runtime of 10 hours ($T_{\text{tot}} = 3.6 \times 10^4$ s). Using Figure 10, it can be seen that the classical algorithm has an error of 1.43×10^{-5}. This is outperformed by the QMC algorithm only for gate times 10^{-7} s and 10^{-8} s, with corresponding errors of 8.47×10^{-6} and 1.97×10^{-6}, respectively. Conversely, consider a target accuracy of $\varepsilon = 10^{-6}$. The classical algorithm requires a runtime of roughly 21 hours, while the QMC algorithm requires a runtime of around 1 hour when $t = 10^{-8}$ s and around 8 hours when $t = 10^{-7}$ s.

7. Conclusion

This paper is the first to apply the quantum Monte Carlo algorithm to problems in economics and among the first to apply quantum computation more generally to this field. We build our paper in such a way that an economist without any knowledge of quantum computation can gradually progress to fully implementing the QMC algorithm. We begin with a simple Gaussian sampling needed to estimate the mean of a random variable to illustrate quantum decomposition. Next, we focus on more sophisticated applications: (a) stress testing of banks and (b) solving DSGE models with deep learning. We derive in detail how these problems can be converted into quantum circuits—both theoretically and in terms of code. The quantum simulations that we perform for these problems verify that quantum solutions converge to the theoretical predictions. In the absence of hardware noise, high accuracy is achieved even for a small number of qubits.

The other important contribution of this paper is a fair comparison between QMC and classical MC. Whereas QMC speedup is usually represented in terms of the number of oracle calls (number of times the unitary $\mathcal{Q}$ is applied), this paper shows how to obtain a direct time-vs-time comparison. For the DSGE deep learning problem, we decompose the quantum circuit into a set of standard elementary gates. This allows us to compute the depth of the circuit, which we convert into computational time of the QMC algorithm for various quantum gate times. The resulting graph informs when quantum advantage might be expected for this economics problem.

With regard to the QMC algorithm itself, this paper makes two minor contributions. We show how the unitary $\mathcal{A}$ can be approximated with a quantum variational circuit. Also, we show how the controlled $\mathcal{Q}$ unitary can be achieved without the controlled $\mathcal{F}$ unitary.

Although further improvements in quantum hardware are necessary before QMC can achieve an advantage in practice over classical MC, this paper shows that the QMC scales better for problems in economics. Therefore, eventually quantum Monte Carlo stands to deliver quantum advantage to economics, where Monte Carlo problems abound.

References

Alaminos, D., Salas, M.B., Fernández-Gámez, M.A., . Quantum computing and deep learning methods for GDP growth forecasting. Computational Economics .

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J.C., Barends, R., Biswas, R., Boixo, S., Brandao, F.G., Buell, D.A., et al., 2019. Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510.

Azinovic, M., Gaegauf, L., Scheidegger, S., 2022. Deep equilibrium nets. International Economic Review .

Bach, F., 2017. Breaking the curse of dimensionality with convex neural networks. The Journal of Machine Learning Research 18, 629–681.

Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H., 1995. Elementary gates for quantum computation. Physical Review A 52, 3457.

Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M.S., Ahmed, S., Arrazola, J.M., Blank, C., Delgado, A., Jahangiri, S., et al., 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv preprint arXiv:1811.04968 .

Brassard, G., Hoyer, P., Mosca, M., Tapp, A., 2000. Quantum amplitude amplification and estimation. Quantum Computation and Quantum Information: A Millennium Volume. AMS Contemporary Mathematics Series .

Burchard, P., 2019. Lower bounds for parallel quantum counting. arXiv preprint arXiv:1910.04555 .

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., et al., 2021. Variational quantum algorithms. Nature Reviews Physics , 1–20.

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.

Chakraborty, S., 2015. Generating discrete analogues of continuous probability distributions-a survey of methods and constructions. Journal of Statistical Distributions and Applications 2, 6.

Cleve, R., Ekert, A., Macchiavello, C., Mosca, M., 1998. Quantum algorithms revisited. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 454, 339–354.

Crooks, G.E., 2019. Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. arXiv preprint arXiv:1905.13311 .

Davidson, R., MacKinnon, J.G., et al., 1993. Estimation and inference in econometrics. volume 63. Oxford New York.

Duprey, T., Liu, X., MacDonald, C., van Oordt, M., Priazhkina, S., Shen, X., Slive, J., 2018. Modelling the emergence of the interbank networks. Bank of Canada Staff Analytical Note 36.

Efron, B., 1979. Bootstrap methods: Another look at the jackknife. The Annals of Statistics 7, 1–26.

Egger, D.J., Gutierrez, R.G., Mestre, J.C., Woerner, S., 2020. Credit risk analysis using quantum computers. IEEE Transactions on Computers .

Farmer, J.D., Kleinnijenhuis, A.M., Schuermann, T., Wetzer, T., 2022. Handbook of Financial Stress Testing. Cambridge University Press.

Fernández-Villaverde, J., Hurtado, S., Nuno, G., 2019. Financial frictions and the wealth distribution. Technical Report. National Bureau of Economic Research.

Giurgica-Tiron, T., Kerenidis, I., Labib, F., Prakash, A., Zeng, W., 2020. Low depth algorithms for quantum amplitude estimation. arXiv preprint arXiv:2012.03348 .

Gómez, A., Leitao, Á., Manzano, A., Musso, D., Nogueiras, M.R., Ordóñez, G., Vázquez, C., 2022. A survey on quantum computational finance for derivatives pricing and var. Archives of Computational Methods in Engineering , 1–27.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT press.

Grinko, D., Gacon, J., Zoufal, C., Woerner, S., 2021. Iterative quantum amplitude estimation. npj Quantum Information 7, 1–6.

Grover, L., 1996. A fast quantum mechanical algorithm for database search, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212–219.

Grover, L., Rudolph, T., 2002. Creating superpositions that correspond to efficiently integrable probability distributions. arXiv preprint quant-ph/0208112 .

Hastings, W.K., 1970. Monte carlo sampling methods using markov chains and their applications .

Hałaj, G., Priazhkina, S., 2021. Stressed but not helpless: Strategic behaviour of banks under adverse market conditions. Bank of Canada Staff Working Paper 35.

Hendry, D.F., 1984. Monte carlo experimentation in econometrics. Handbook of Econometrics 2, 937–976.

Herbert, S., 2021a. The problem with grover-rudolph state preparation for quantum monte-carlo. arXiv:2101.02240.

Herbert, S., 2021b. Quantum monte-carlo integration: The full advantage in minimal circuit depth. arXiv:2105.09100.

Hommes, C.H., 2006. Heterogeneous agent models in economics and finance. Handbook of Computational Economics 2, 1109–1186.

Hull, I., Sattath, O., Diamanti, E., Wendin, G., 2020. Quantum technology for economists. Available at SSRN 3745608 .

Häner, T., Roetteler, M., Svore, K.M., 2018. Optimizing quantum circuits for arithmetic. arXiv:1805.12445.

Jordan, S., 2022. Quantum algorithm zoo. URL: https://quantumalgorithmzoo.org/.

Judd, K.L., 1998. Numerical methods in economics. MIT press.

Kalos, M.H., Whitlock, P.A., 2009. Monte carlo methods. John Wiley & Sons.

Kaneko, K., Miyamoto, K., Takeda, N., Yoshino, K., 2020. Quantum pricing with a smile: Implementation of local volatility model on quantum computer. arXiv preprint arXiv:2007.01467 .

Kaye, P., Mosca, M., 2001. Quantum networks for generating arbitrary quantum states, in: International Conference on Quantum Information, Optical Society of America. p. PB28.

Kitaev, A., Webb, W.A., 2008. Wavefunction preparation and resampling using a quantum computer. arXiv preprint arXiv:0801.0342 .

Kitaev, A.Y., 1995. Quantum measurements and the abelian stabilizer problem. arXiv preprint quant-ph/9511026 .

Konečnỳ, J., McMahan, B., Ramage, D., 2015. Federated optimization: Distributed optimization beyond the datacenter. arXiv preprint arXiv:1511.03575 .

Lepetyuk, V., Maliar, L., Maliar, S., 2020. When the us catches a cold, canada sneezes: A lower-bound tale told by deep learning. Journal of Economic Dynamics and Control 117, 103926.

Lubinski, T., Johri, S., Varosy, P., Coleman, J., Zhao, L., Necaise, J., Baldwin, C.H., Mayer, K., Proctor, T., 2021. Application-oriented performance benchmarks for quantum computing. arXiv preprint arXiv:2110.03137 .

Madsen, L.S., Laudenbach, F., Askarani, M.F., Rortais, F., Vincent, T., Bulmer, J.F., Miatto, F.M., Neuhaus, L., Helt, L.G., Collins, M.J., et al.,

2022. Quantum computational advantage with a programmable photonic processor. Nature 606, 75–81.

Maliar, L., Maliar, S., 2015. Merging simulation and projection approaches to solve high-dimensional problems with an application to a new keynesian model. Quantitative Economics 6, 1–47.

Maliar, L., Maliar, S., Winant, P., 2021. Deep learning for solving dynamic economic models. Journal of Monetary Economics .

Martin, S., 2021. Here, there, everywhere: Nvidia platform accelerates quantum circuit simulation ecosystem. [Online; accessed 27-April-2022].

Metropolis, N., Ulam, S., 1949. The monte carlo method. Journal of the American Statistical Association 44, 335–341.

Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K., 2018. Quantum circuit learning. Physical Review A 98, 032309.

Montanaro, A., 2015. Quantum speedup of monte carlo methods. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 471, 20150301.

Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M., 2005. Transformation of quantum states using uniformly controlled rotations. Quantum Info. Comput. 5, 467–473.

Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M., 2004. Quantum circuits for general multiqubit gates. Physical Review Letters 93.

Nielsen, M., Chuang, I., 2010. Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press.

Orús, R., Mugel, S., Lizaso, E., 2019. Forecasting financial crashes with quantum computing. Physical Review A 99, 060301.

Pang, T., 2016. An Introduction to Quantum Monte Carlo Methods. IOP Concise Physics, Morgan & Claypool Publishers.

Plesch, M., Brukner, Č., 2011. Quantum-state preparation with universal gate decompositions. Physical Review A 83, 032302.

Rebentrost, P., Gupt, B., Bromley, T.R., 2018. Quantum computational finance: Monte carlo pricing of financial derivatives. Physical Review A 98.

Rubin, D.B., 1981. The bayesian bootstrap. The Annals of Statistics , 130–134.

Satoh, T., Oomura, S., Sugawara, M., Yamamoto, N., 2022. Pulse-engineered controlled-v gate andits applications on superconductingquantum device. IEEE Transactions on Quantum Engineering .

Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., Killoran, N., 2019. Evaluating analytic gradients on quantum hardware. Physical Review A 99, 032331.

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

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

Stokey, N.L., Lucas Jr, R.E., Prescott, E.C., 1989. Recursive Methods in Economic Dynamics. Harvard University Press.

Suzuki, Y., Uno, S., Raymond, R., Tanaka, T., Onodera, T., Yamamoto, N., 2020. Amplitude estimation without phase estimation. Quantum Information Processing 19, 1–17.

Vedral, V., Barenco, A., Ekert, A., 1996. Quantum networks for elementary arithmetic operations. Physical Review A 54, 147.

Woerner, S., Egger, D.J., 2019. Quantum risk analysis. npj Quantum Information 5, 1–8.

Xu, G., Daley, A.J., Givi, P., Somma, R.D., 2018. Turbulent mixing simulation via a quantum algorithm. AIAA Journal 56, 687–699.

Zhong, H.S., Wang, H., Deng, Y.H., Chen, M.C., Peng, L.C., Luo, Y.H., Qin, J., Wu, D., Ding, X., Hu, Y., et al., 2020. Quantum computational advantage using photons. Science 370, 1460–1463.

Zoufal, C., Lucchi, A., Woerner, S., 2019. Quantum generative adversarial networks for learning and loading random distributions. npj Quantum Information 5, 103.

Appendix A. Discretizing the simple example

The following code shows how the simple example in Appendix A can be discretized using Python code.

```
from pennylane import numpy as np
from scipy.stats import norm

m = 5
M = 2 ** m

x_max = np.pi
x = np.linspace(-x_max, x_max, M)

p = np.array([norm().pdf(x_i) for x_i in x])
p /= np.sum(p)

f = lambda i: np.sin(x[i]) ** 2
```

```
import pennylane as qml
import numpy as np
from scipy.stats import norm

m = 5
n = 6
M = 2 ** m

x_max = np.pi
x = np.linspace(-x_max, x_max, M)

p = np.array([norm().pdf(x_i) for x_i in x])
p /= np.sum(p)

f = lambda i: np.sin(x[i]) ** 2
```

```
qubits = range(m + 1)
est_qubits = range(m + 1, n + m + 1)
n_qubits = n + m + 1

dev = qml.device("default.qubit", wires=n_qubits)

@qml.qnode(dev)
def qmc():
    qml.templates.QuantumMonteCarlo(
        p,
        f,
        qubits,
        est_qubits,
    )
    return qml.probs(est_qubits)

p_est = qmc()
```

Appendix B. Optimizing a variational circuit to prepare the probability distribution

Appendix C discusses how a variational quantum circuit can be optimized to output a target probability distribution when sampled in the computational basis. The code below shows how this can be achieved using the probability distribution and $m = 5$-qubit setting detailed in Appendix A.

```
reps = 1000
layers = 5
a_qubits = range(m)
dv = qml.device("default.qubit", wires=a_qubits)
theta = qml.init.strong_ent_layers_uniform(
    n_layers=layers, n_wires=m
)


def U(theta):
    qml.templates.StronglyEntanglingLayers(
        theta, wires=a_qubits
    )


@qml.qnode(dv)
def p_circ(theta):
    U(theta)
    return qml.probs(wires=a_qubits)


def C(theta):
    return np.sum(np.abs(p_circ(theta) - p))


opt = qml.GradientDescentOptimizer()

for i in range(reps):
    theta, c = opt.step_and_cost(C, theta)
    if i % 50 == 0:
        print(f"Cost at step {i}:", c)

print("Final cost:", C(theta))
```

The variational circuit is composed of 5 layers. Each layer applies a single-qubit rotation gate to each qubit, with each gate having 3 parameters allowing for a general rotation in the Bloch sphere. The rotation gates are followed by an entangling block of CNOT gates.

Appendix C. Implementation of quantum Monte Carlo algorithm for Gaussian example

With the QMC algorithm defined, it is natural to ask how the algorithm may be practically implemented. To do so, we must decompose the probability-encoding unitary $\mathcal{A}$, the random-variable encoding unitary $\mathcal{R}$, and the phase-encoding unitary $\mathcal{Q}$ into elementary gates that are compatible with quantum hardware.

Practical applications of Monte Carlo estimation typically require evaluating expectation values in a multidimensional space, that is, as shown in equation (5) with $d > 1$. Here, it is often useful to split up the register of discretization qubits into d subregisters of m qubits so that each dimension in $\boldsymbol{x}$ is discretized into 2^m points. This approach is particularly fruitful when the joint probability distribution $p(\boldsymbol{x})$ is a product distribution so that each dimension of $\boldsymbol{x}$ is independent. A product distribution allows $\mathcal{A}$ to be written as

$$\mathcal{A} = \bigotimes_{j=1}^{d} \mathcal{A}_j, \tag{C.1}$$

with each $\mathcal{A}_j$ applied independently to subregister j and preparing the corresponding marginal distribution. We focus on this use-case in the following.

Decomposing $\mathcal{A}$

Encoding a probability distribution as a quantum state is in general nontrivial and is a special case of state preparation. This can be performed using exponentially-scaling circuits (Möttönen et al. (2005), Plesch and Brukner (2011)), but such approaches are insufficient for the QMC algorithm because the quadratic speedup will be lost. We hence aim to identify efficient circuits that scale polynomially with the number of qubits in the discretization register.

An efficient decomposition for $\mathcal{A}$, known as the Grover-Rudolph method (Grover and Rudolph (2002)), exists for probability distributions that are efficiently integrable on a classical computer, such as log-concave distributions. However, there can be a significant overhead associated with calculating integrals in a pre-processing step. This is especially so when the integrals themselves need to be approximated with Monte Carlo estimation. As a result, it has been argued that the Grover-Rudolph method is insufficient for the QMC algorithm (Chakrabarti et al. (2021), Herbert (2021a)).

Although alternative algorithms have been proposed by such authors as Kaye and Mosca (2001), Kitaev and Webb (2008), Kaneko et al. (2020), a major focus has been on optimizing variational quantum circuits to approximate a target distribution. The variational approach provides the flexibility to construct low-depth circuits or increase the depth depending on the required accuracy. The work of Zoufal et al. (2019) proposes a hybrid quantum-classical generative adversarial network and demonstrates its ability to prepare canonical distributions, such as the log normal distribution. Here, we show how the probability distribution of a variational circuit can be optimized directly using gradient-based optimization.

Consider an m-qubit variational quantum circuit $U(\boldsymbol{\theta})$ composed of gates with trainable parameters $\boldsymbol{\theta}$ and applied to an input state $|0\rangle^{\otimes m}$. When sampling in the computational basis, the circuit has a probability distribution

$$p(i, \boldsymbol{\theta}) = |\langle i|U(\boldsymbol{\theta})|0\rangle|^2 . \tag{C.2}$$

The objective is to minimize the distance between this distribution and a target distribution $p_{\text{targ}}(i)$. This can be achieved using the cost function

$$C(\boldsymbol{\theta}) = \sum_i |p(i, \boldsymbol{\theta}) - p_{\text{targ}}(i)| \tag{C.3}$$

and finding

$$\boldsymbol{\theta}_{\text{opt}} = \text{argmin}_{\boldsymbol{\theta}} \;\; C(\boldsymbol{\theta}). \tag{C.4}$$

Minimization of the cost function can be achieved using gradient-based

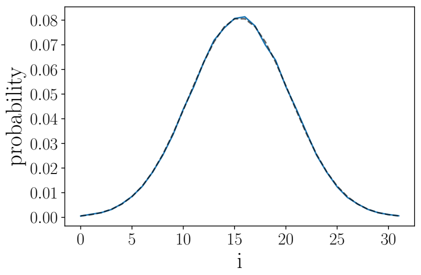

Figure C.11: Optimizing a variational quantum circuit so that its output probability distribution in the computational basis is the discretized normal distribution. The solid blue line shows the output distribution from the optimized circuit, while the dashed grey line shows the target distribution.

optimizers by calculating the gradient $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$, which is accessible on quantum hardware using the finite-difference approximation or the parameter-shift rule for certain types of circuit (Mitarai et al. (2018), Schuld et al. (2019)). The resulting circuit $U(\boldsymbol{\theta}_{\text{opt}})$ can then be used as an approximation to $\mathcal{A}$ (or $\mathcal{A}_j$). Appendix B shows how this method can be used to train a variational circuit to reproduce the discretized normal distribution discussed in section 4.4. The resulting probability distribution $p(i, \boldsymbol{\theta}_{\text{opt}})$ is shown in Figure C.11.

Decomposing $\mathcal{R}$

Similarly to the decomposition for $\mathcal{A}$, although approaches that scale exponentially with the number of qubits in the discretization register exist for applying $\mathcal{R}$ (Möttönen et al. (2004)), doing so is insufficient to preserve the quadratic speedup of the QMC algorithm. One commonly-used approach is to discretize so that each dimension of $\boldsymbol{x}$ is represented as an m-bit fixed-point number (Rebentrost et al. (2018), Chakrabarti et al. (2021)), for example, so that a positive x can be written as

$$x = \sum_{i=1}^{m} 2^{m-i+k} b_i, \tag{C.5}$$

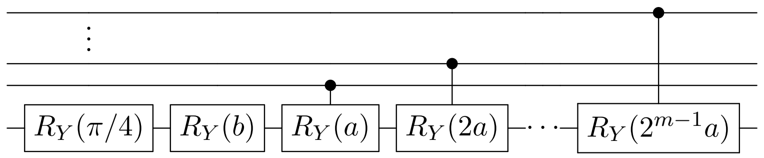

Figure C.12: Encoding a linear function $f(x)$ onto an ancilla qubit from an m-qubit register can be achieved by discretizing and rescaling to $f(i) = ai + b$ and applying the above circuit Egger et al. (2020). Here, $R_Y(\phi) = e^{-i\phi\sigma_Y}$ with σ_Y the Pauli-Y operator.

with a corresponding bitstring $\{b_1, b_2, \ldots, b_m\}$ and a fixed point $k \in \mathbb{Z}$. An additional bit may be used to encode the sign of x. We can then associate the m-qubit basis state $|b_1 b_2 \ldots b_m\rangle$ with x. With the random variable $f(\boldsymbol{x})$ written as a composition of such operations on $\boldsymbol{x}$, $\mathcal{R}$ can then be performed by applying those operations onto the corresponding registers.

Elementary arithmetic on a quantum circuit is a well-established topic, and efficient decompositions are available for common operations like addition and multiplication.[16] Additional registers of calculation qubits may be required to compute $f(\boldsymbol{x})$. The result must then be imprinted from a result register onto the ancilla qubit. This can be achieved by first applying square-root and arcsine operations and then performing a cascade of controlled-Y rotations.

Despite the existence of efficient decompositions, the elementary arithmetic approach to performing $\mathcal{R}$ is challenging. Each operation can require a non-negligible number of gates to enact as well as an additional register of qubits to store the output, making the approach costly for implementation on hardware and simulators. For example, adding two registers of $m = 5$ qubits using the approach of Vedral et al. (1996) requires 18 Toffoli gates, 20 CNOT gates, and an additional register of qubits. This is an overhead that quickly adds up with multiple operations. There are also often multiple decompositions provided in the literature with differing advantages and disadvantages, and choosing which to adopt can be difficult.

[16]See Häner et al. (2018), Chakrabarti et al. (2021) for details.

Subsequent works have proposed alternative implementations of $\mathcal{R}$ with the objective of avoiding quantum arithmetic as in Stamatopoulos et al. (2020a), Herbert (2021b). In this paper, we consider a simple approach outlined in Egger et al. (2020), Stamatopoulos et al. (2020b) for a one-dimensional linear function $f(x)$. Although this setting is trivial to solve on a classical computer, it provides an entry point into exploring the application of QMC to a relevant problem in economics and for benchmarking the resource requirements, as we see in the following sections.

In this approach, the function is first discretized and rescaled to the range $[-c_s, c_s]$ for some small constant $c_s > 0$. The result can be written as

$$f(i) = ai + b, \tag{C.6}$$

with two constants a and b that depend on c_s. The cascade of controlled-Y rotations shown in Figure C.12 can then be shown to approximate $\mathcal{R}$, that is, so that the probability of measuring the ancilla qubit in the $|1\rangle$ state can be rescaled according to c_s to provide an estimate of $\mathrm{E}(f(x))$. This holds provided that c_s is sufficiently small, such that $\sin^2(f(i) + \pi/4) \approx f(i) + 1/2$. Indeed, c_s must be set so that the dominant error is from the Monte Carlo estimation, and it can be shown that the optimal c_s scales with the number N of applications of $\mathcal{Q}$ as $N^{-1/3}$ (Woerner and Egger (2019)). However, this approximation results in a less preferential scaling with $N = \mathcal{O}(\varepsilon^{-3/2})$.

Decomposing $\mathcal{Q}$

Recall that $\mathcal{Q} = (\mathcal{F}\mathcal{Z}\mathcal{F}^\dagger\mathcal{V})^2$ as given in equation (21). However, the QMC algorithm requires the ability to apply $\mathcal{Q}$ with control from one of the phase estimation qubits. Due to the nature of $\mathcal{Q}$, this can be achieved without the need for a controlled version of $\mathcal{F}$, as shown in Figure C.13.

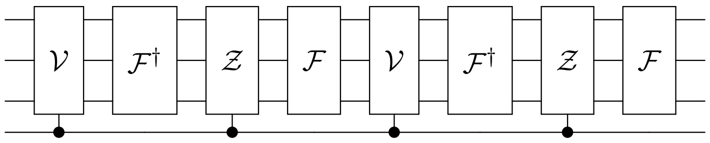

Figure C.13: A controlled version of $\mathcal{Q}$ can be achieved without using a controlled-$\mathcal{F}$.

Let us now consider the unitary $\mathcal{V}$. It can be seen that

$$\begin{aligned}\mathcal{V} &= \mathbb{1}_{m+1} - 2\mathbb{1}_m \otimes |1\rangle\langle 1| \\ &= \mathbb{1}_m \otimes \sigma_z,\end{aligned} \tag{C.7}$$

with σ_z the Pauli-Z matrix. Hence, controlled-$\mathcal{V}$ can be implemented simply as a CZ gate. Furthermore, the controlled-$\mathcal{Z}$ gate can be implemented as shown in Figure C.14. This requires the ability to apply a multi-controlled-X gate, for which an efficient decomposition exists that scales linearly with m (Barenco et al. (1995)).

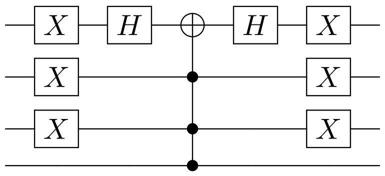

Figure C.14: Applying a controlled version of $\mathcal{V}$ can be achieved using Pauli-X rotations, Hadamard gates, and a multi-controlled-X gate. Note that the target for the multi-controlled-X gate can be any of the wires in the m-qubit register, provided the Hadamard gates are also applied to that wire.

Appendix D. Details of the QMC implementation for the neoclassical model

We discretize our target function (47) into $M = 32$ points using $m = 5$ qubits and rescale the function to $f(i) = ai + b \in [-c_s, c_s]$ according to a constant $c_s > 0$, such that $a = 2c_s/31$ and $b = -c_s$. When using n qubits and $N = 2^n$ applications of $\mathcal{Q}$ in the QMC algorithm, we set $c_s = \sqrt[3]{3\pi/N}$ as suggested in Woerner and Egger (2019).

Using this, $\mathcal{R}$ can be applied according to the cascade of controlled-Y rotations in Figure C.12. For the probability-encoding unitary $\mathcal{A}$, we set x_i and $p(i)$ using equation ((28)) with $x_{\text{max}} = 1.06$ and $x_{\text{min}} = 0.94$ (which corresponds to three standard deviations away from the mean), and carry out the optimization procedure in Appendix C. With 10 trainable layers, 150 parameters are varied to minimize the cost function $C(\boldsymbol{\theta})$ in equation ((C.3)). The training was carried out with the learning rate equal to 0.01 for the first 1000 epochs, 0.001 for the next 1000 epochs, and 0.0001 for the final 1000 epochs (see Appendix B for details). The final cost reached this way is 1.54×10^{-4}. The code block below shows how the resulting $\mathcal{F}$ can be constructed in PennyLane.

```
m = 5
c_s = np.power(3*np.pi/N, 1 / 3)
a = 2 * c_s / 31
b = -c_s
ancilla_qubit = m
a_qubits = range(m)

def F():
    A(theta)  # Apply A trained previously

    # Apply R
    qml.RY(np.pi / 2, wires=ancilla_qubit)
    qml.RY(2 * b, wires=ancilla_qubit)
    for qubit in a_qubits:
        phi = 2 ** (m - qubit) * a
        qml.CRY(phi,wires=[qubit,ancilla_qubit])
```

With the components of $\mathcal{F}$ fixed, we can then proceed to apply the QMC algorithm. This can be achieved as follows (where "dev" can refer to a quantum device or a classical simulator):

```
#n is number of estimation qubits chosen
est_qubits = range(m + 1, n + m + 1)
@qml.qnode(dev)
def qmc():
    qml.quantum_monte_carlo(
        F, qubits, ancilla_qubit, est_qubits)
    return qml.probs(est_qubits)
```

The quantum resource requirements of this circuit can be obtained as a Python dictionary using the following code:

```
expanded_tape = qmc.qtape.expand(depth=10)
specs = expanded_tape.specs
```

Appendix E. Classical MC solution using PyTorch for benchmarking in section 6

```
import torch
from torch.utils.data import WeightedRandomSampler
from torch.utils.data import DataLoader
import numpy as np
from scipy.stats import norm
from time import time

device=torch.device('cuda' if torch.cuda.is_available()\
                    else 'cpu')

nepochs=10
nsample=100000

N=nepochs*nsamples #total samples drawn
```

```
mu=1.0
sigma=0.02
xmax=1.06
xmin=0.94
m=5
M=2**m

xs=np.linspace(xmin,xmax,M)
probs=np.array([norm(loc=mu,scale=sigma).pdf(x) for x in xs])
probs/=np.sum(probs)

tprobs=torch.tensor(probs).to(device)
txs=torch.tensor(xs).to(device)

batch_size=1000

sampler=WeightedRandomSampler(probs,nsample,replacement=True)
data=DataLoader(txs,batch_size=batch_size,sampler=sampler)

totsum=torch.tensor(0.0,device=device).to(torch.float64)

t0=time()

for epoch in range(nepochs):
    epochsum=torch.tensor(0.0,device=device).to(torch.float64)
    for x in data:
        epochsum+=x.sum()
    totsum+=epochsum/nsample

t1=time()

print('Samples drawn:',N)
print('Time to draw samples:',t1-t0)
print('Computed MC estimate (true value = 1.0):',\
            totsum.item()/nepochs)
```