Explaining quantum phase estimation using data structures

(Dated: October 29, 2025)

Abstract

The quantum phase estimation algorithm is an fundamental technique for performing physical simulation tasks on quantum computers. Most existing presentations of quantum phase estimation proceed by first stating the quantum circuit for phase estimation and then using an algebraic manipulation to prove its correctness. Such a presentation can be unsatisfactory as it does not explain quantum phase estimation by appeal to any algorithmic ideas. This paper explains the phase estimation algorithm in a new way, using elementary polynomial data structures from computer science as helpful high-level concepts. Once the polynomial data structure concepts have been understood, the explanation of quantum phase estimation can be stated in just a few lines. We hope this alternative approach to explaining phase estimation will help instructors as quantum computing grows in popularity and is taught in more diverse contexts.

I. INTRODUCTION

Quantum computers promise new algorithms for solving problems that are intractable on classical computers[1]. Their power comes from quantum bits (qubits), which, unlike classical bits, can exploit superposition, interference, and entanglement to store and process information more efficiently. Among the most important algorithms that harness this power is quantum phase estimation[2]. It is a subroutine in algorithms for the simulation of physical systems[3] and in Shor's factoring algorithm[4].

Most existing presentations of phase estimation use detailed algebraic derivations[1,5]. They generally present the quantum circuit for the algorithm and present an analysis of the correctness of the algorithm by means of algebraic manipulation. While rigorous, they often obscure the underlying ideas and can be tedious especially for settings where a quick, high-level explanation would be sufficient. The usefulness of this paper is in providing an alternative explanation for quantum phase estimation. We show how elementary data structures for representing and multiplying polynomials efficiently and a divide-and-conquer approach can help explain the quantum phase estimation algorithm at a high level. Instructors can consider whether the usual algebraic details are required in their course, and they may opt to leave them to students in the form of homework exercises to be done only after the students are comfortable with the main algorithmic idea.

An ever-larger community of scientists will need at least a working understanding of phase estimation because quantum computing is a growing scientific area and industry in its own right, and it is expected that it will be applied in various branches of science. With the rapid growth of university courses and programs in quantum computing, many instructors across physics will find themselves teaching or applying it. We hope our conceptual explanation will be helpful to these instructors.

II. DATA STRUCTURES FOR PHASE ESTIMATION

This section introduces the concepts that feature in our explanation of quantum phase estimation. We first review classical data structures for representing polynomials, then show how these motivate the Discrete and Fast Fourier transforms, and finally explain the quantum analogues of these ideas. Table I condenses these concepts and how they are

related.

A. Polynomial data structures

The material in this section can be motivated by asking students to consider how polynomials might be represented in a classical computer. The straightforward approach, the **coefficient representation** of a polynomial[6] involves storing a vector (or list) of all its coefficients. For example the polynomial $p(x) = a_{N-1}x^{N-1} + ... + a_0$ is stored as the vector $a = [a_0...a_{N-1}]$.

There are other data structures for polynomials[6]. Consider the **point-value representation** of a polynomial[6], which uses the fact that a polynomial, p, with degree less than N is uniquely determined by a set of N distinct points $\{(x_0, p(x_0)), ..., (x_{N-1}, p(x_{N-1}))\}$. For example, if we store two points, we represent the unique line that passes through those points. If we fix the inputs $\{x_0, ..., x_{N-1}\}$ at which we will evaluate every polynomial, then we can represent the any polynomial of degree up to N by only storing the values $[p(x_0)...p(x_{N-1})]$ as a vector. We are free to choose any set of N distinct input values for this purpose.

The point-value representation is significant in classical algorithms because multiplying two polynomials that are represented in this way is efficient. For two polynomials p and q, their product $r = p \cdot q$ can be obtained simply by multiplying their stored values coordinate by coordinate:

$$[r(x_0), \ldots, r(x_{N-1})] = [p(x_0)q(x_0), \ldots, p(x_{N-1})q(x_{N-1})]. \tag{1}$$

One requirement for this multiplication procedure is that polynomials p and q each have degree less than $N/2$ so that the resulting product r still has degree less than N and can be uniquely determined by N evaluation points. Multiplication of polynomials in the point-value representation only requires $\mathcal{O}(N)$ multiplications of numbers. In contrast, multiplication of polynomials in the coefficient representation requires combining and summing cross-terms, which requires $O(N^2)$ additions and multiplications of numbers.

We now motivate the discrete Fourier transform from this perspective. The coefficient representation is the more common and natural choice, so one would hope to store their polynomials in the coefficient representation but use a method to convert between the two representations efficiently in order to take advantage of the faster multiplication algorithm in

the point-value representation. The naive approach to converting from coefficients to point values requires multiplying by a Vandermonde matrix. For example, given $p(x) = a_0 + a_1 x + \cdots + a_{N-1}x^{N-1}$ and fixed evaluation points $\{x_0, \ldots, x_{N-1}\}$, the point value representation is obtained by:

$$\begin{bmatrix} p(x_0) \\ p(x_1) \\ \vdots \\ p(x_{N-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N-1} & x_{N-1}^2 & \cdots & x_{N-1}^{N-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix}. \tag{2}$$

In general multiplying a matrix onto a vector requires $O(N^2)$ elementary operations. Thus we cannot hope to benefit from the $O(N)$ multiplication of the point-value representation if we convert representations in the naive way.

However it is possible to convert representations in $O(N \log N)$ time if we choose the a special set of evaluation points. Specifically, if we choose the N-th roots of unity (i.e. powers of $\omega_N = e^{i\frac{2\pi}{N}}$) then the Vandermonde matrix becomes

$$F_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^{2\cdot 2} & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}. \tag{3}$$

This matrix F_N is called the discrete Fourier transform (DFT) matrix. Applying it to the coefficient vector $[a_0, \ldots, a_{N-1}]$ computes the values $p(1), p(\omega_N), \ldots, p(\omega_N^{N-1})$.

The Fast Fourier Transform (FFT) is a divide-and-conquer method to multiply F_N to a vector in only $O(N \log N)$ time, making the conversion between coefficient and point-value representations efficient.

The key idea is to split the N coefficients $(a_0, \ldots, a_{N-1})$ into their even- and odd-indexed parts, forming two smaller polynomials of size $N/2$. These smaller polynomials can be evaluated at all the points more efficiently because some values repeat, and only half as many polynomial evaluations are needed. For instance, since

$$\omega_N^{2(i+N/2)} = \omega_N^{2i}\,\omega_N^N = \omega_N^{2i},$$

we have

$$p_{\text{even}}(\omega_N^{i+N/2}) = p_{\text{even}}(\omega_N^i),$$

i.e. the evaluations at ω_N^i and $\omega_N^{i+N/2}$ are identical for all i. This is also true for the odd-indexed part. Then combining the results using simple additions and multiplications by powers of ω_N:

$$p(\omega_N^k) = p_{\text{even}}(\omega_N^{2k}) + \omega_N^k \, p_{\text{odd}}(\omega_N^{2k}), \qquad k = 0, \ldots, N/2 - 1. \tag{4}$$

Recursively applying this decomposition halves the problem size at each step, reducing the total work from $O(N^2)$ to $O(N \log N)$.

B. Quantum data structures

After students have been introduced to the background concepts in the previous section, the material in this section can be motivated as the natural quantum analogues of the concepts in the previous section.

Quantum computers are capable of performing some computations on large vectors efficiently by encoding the data into the amplitudes of a quantum state. Since an n qubit quantum state is described using a vector of 2^n amplitudes, computations that encode vectors as quantum states and use them are capable of achieving large advantages in space and time complexity. The standard encoding, known as the **amplitude encoding** of a vector, is the quantum state obtained by normalising the components of a vector and putting them into amplitudes of a quantum state, for example as: $v = [v_0 ... v_{N-1}] \rightarrow \frac{1}{\|v\|} \sum_{i=0}^{N-1} v_i \, |i\rangle$. Our explanation of QPE will feature the *amplitude encodings* of the coefficient and point-value representations of polynomials.

The primary way that quantum computers manipulate these amplitude encoded vectors is by the multiplication of unitary matrices. When normalised by a $\frac{1}{\sqrt{N}}$ factor, the Discrete Fourier Transform is a unitary matrix. If we implement this matrix as a quantum circuit, it converts from the *amplitude encoding* of the coefficient representation to the *amplitude encoding* of the point-value representation. This circuit is called the Quantum Fourier Transform and it can be implemented efficiently due to a divide-and-conquer algorithm which follows a similar logic to the divide-and-conquer approach of the Fast Fourier Transform[7].

	Coefficient Representation	**Point-value Representation**
Classical	$[\, a_0 \quad a_1 \quad \dots \quad a_{N-1} \,]$	$[\, p(1) \quad p(\omega_N) \quad \dots \quad p(\omega_N^{N-1}) \,]$
Quantum (Amplitude encoding)	$\frac{1}{\|a\|} \sum_{j=0}^{N-1} a_j \, \|j\rangle$	$\frac{1}{\sqrt{N}\|a\|} \sum_{j=0}^{N-1} p(\omega_N^j) \, \|j\rangle$

TABLE I: Data structures for storing polynomial $p(x) = a_0 + ... + a_{N-1}x^{N-1}$. The two classical data structures are a list of values or a vector. The two quantum data structures are the amplitude encodings of the same vectors. We can convert from the coefficient representation to the point-value representation by applying the Discrete Fourier Transform in the classical case, and the Quantum Fourier Transform in the quantum case.

III. QUANTUM PHASE ESTIMATION

A. Problem

We now apply the above concepts to explain quantum phase estimation (QPE). The QPE problem formalizes the task of extracting an eigenvalue of a unitary operator given an eigenstate.

Problem. *Let U denote a unitary matrix, and let $|\psi\rangle \in \mathbb{P}(\mathbb{C}^N)$ be an eigenvector of U with corresponding eigenvalue $\omega_N^k = e^{i2\pi k/N}$, where $k \in \mathbb{Z}$ is an unknown.*

Given:

1. *$|\psi\rangle$ as one copy of the quantum state, and*

2. *access to U via black-box calls to a qudit-controlled-U^d gate,*

the objective is to determine k.

We define the qudit-controlled-U^d gate, as the gate that acts in the following manner: $(C_d U^d) \, |j\rangle \, |\phi\rangle = |j\rangle \, U^j \, |\phi\rangle$. This is a generalisation of the standard controlled-U gate, which either applies the U gate 0 or 1 times, depending on the controlling register. The qudit-controlled-U^d gate applies the U gate j times, if the number in the controlling register is j. This allows us to apply U different numbers of times in a controlled way.

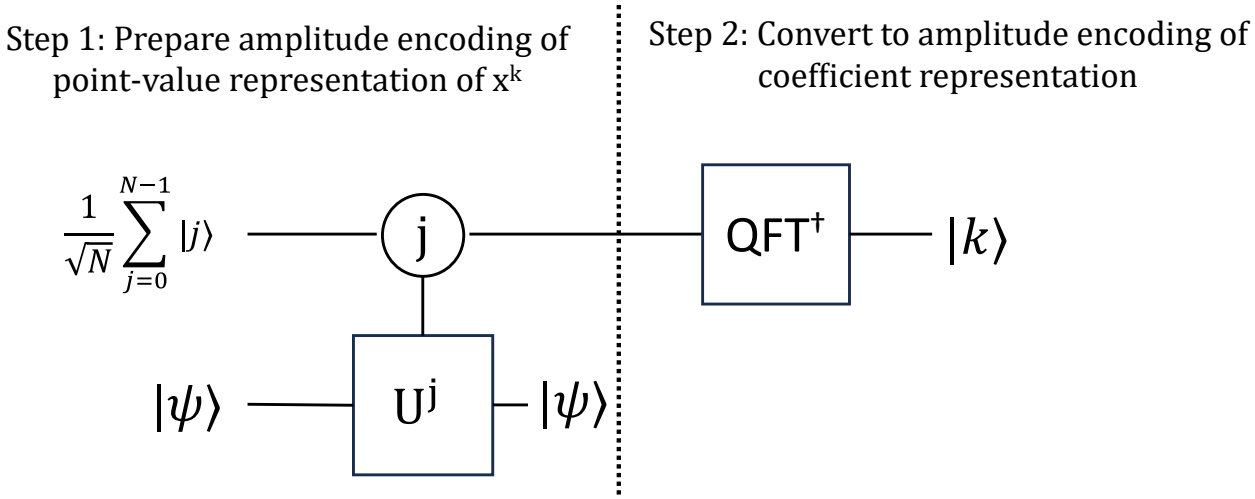

FIG. 1: High level circuit explaining the Quantum Phase Estimation Algorithm. The algorithm has two steps. Step 1 prepares $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^k)^j|j\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^j)^k|j\rangle$, which is the amplitude encoding of the point-value representation of the polynomial x^k. Step 2 converts to the amplitude encoding of the coefficient representation, yielding k.

B. Algorithm

The Quantum Phase Estimation algorithm is a quantum circuit that uses the qudit-controlled-U^d gate and the quantum state $|\psi\rangle$ to determine the eigenvalue. This circuit has two parts. The first part **prepares an amplitude encoding of the point-value representation of the polynomial $\mathbf{x^k}$**. This can be shown from a simple derivation, notice that the resultant state has amplitudes equal to taking each root of unity to the k-th power:

$$\begin{aligned} & C_d U^d[\tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}|j\rangle|\psi\rangle] \\ &= \tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}(\omega_N^k)^j|j\rangle|\psi\rangle \;\; (\text{Since } U|\psi\rangle = \omega_N^k|\psi\rangle) \\ &= \tfrac{1}{\sqrt{N}}\textstyle\sum_{j=0}^{N-1}(\omega_N^j)^k|j\rangle|\psi\rangle \end{aligned} \tag{5}$$

The second part **converts to the amplitude encoding of the coefficient representation of the polynomial** using the inverse-QFT, giving us the outcome $|k\rangle$ since the

polynomial was x^k.

---

[1] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

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

[3] D. S. Abrams and S. Lloyd, "Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors," *Physical Review Letters*, vol. 83, no. 24, p. 5162, 1999.

[4] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[5] T. G. Wong, *Introduction to classical and quantum computing*. Rooted Grove Omaha, NE, USA, 2022.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Polynomials and the FFT*, ch. 30, pp. 898–925. MIT Press, 3rd ed., 2009.

[7] G. Paradisi and H. Randriam, "A presentation of the quantum fourier transform from a recursive viewpoint," *arXiv preprint quant-ph/0411069*, 2004.