Explaining the quantum algorithm for phase estimation using data structures

Shashvat Shukla

September 23, 2025

Abstract

The phase estimation algorithm is an algorithm for quantum computers that will be useful in physical simulation tasks and as a subroutine in Shor's quantum algorithm for prime factorisation. Most textbook presentations of the phase estimation algorithm are driven by algebraic manipulation and prioritise details of how the quantum circuit acts while obscuring the 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 QPE can be stated in just a few lines.

1 Quantum Phase Estimation

1.1 Setup

The Quantum Phase Estimation algorithm solves the following problem: Given a quantum state $|\psi\rangle$, and unitary U and such that $|\psi\rangle$ is an eigenvector of U, find the corresponding eigenvalue. $|\psi\rangle$ is given as one copy of a quantum state, and the access to U is given as access to black-box calls to a generalised controlled-U gate.

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.

The Quantum Phase Estimation algorithm is a quantum circuit that uses the qudit-controlled-U^d gates and the quantum state $|\psi\rangle$ to determine the eigenvalue.

Approximating eigenvalues by roots of unity

Figure 1: The goal of QPE is to find the eigenvalue of U corresponding to $|\psi\rangle$. For a fixed precision N, QPE identifies the closest N-th root of unity to the true eigenvalue ($e^{i2\pi r}$). Thus we can see the goal of QPE as identifying the value k such that r is closest to $\frac{k}{N}$. Notice how increasing N will make the algorithm more accurate as there will be a finer discretisation of the circle.

Say $U|\psi\rangle = e^{i2\pi r}|\psi\rangle$. We can specify the bits of precision used in our algorithm as a parameter n (and define $N = 2^n$) such that $r \approx \frac{k}{N}$ where $k \in \mathbb{N}$. Let $\omega_N = e^{i2\pi/N}$, the principal N-th root of unity. Thus our goal is to determine k s.t. $U|\psi\rangle \approx \omega_N^k|\psi\rangle$ (See Figure 1).

1.2 Polynomial Data Structures

This subsection introduces the key data structure concepts that we use in our explanation of QPE. Figure 2 shows these concepts and how they are related in one graphic. Once the concepts in this subsection have been understood, then QPE can be explained in just a few lines as we do in the next subsection.

The amplitude encoding of a vector is the quantum state obtained from normalising the components of a vector and putting them into amplitudes of a quantum state, for example as: $v = [v_0...v_{N-1}] \rightarrow \frac{1}{\|v\|}\sum_{i=0}^{N-1} v_i|i\rangle$.

The coefficient representation of a polynomial is a vector containing all its coefficients. For example $p(x) = a_{N-1}x^{N-1} + ... + a_0$ is represented as $a = [a_0...a_{N-1}]$. The final step of QPE involves the *amplitude encoding* of the coefficient representation of a polynomial so $p(x) = a_{N-1}x^{N-1} + ... + a_0$ is stored as $\frac{1}{\|a\|}\sum_{i=0}^{N-1} a_i|i\rangle$. Since the polynomial in question turns out to be a single term polynomial x^k, the amplitude encoding of it is just $|k\rangle$.

The point-value representation of a polynomial is motivated by the fact that a polynomial (p with degree less than N) is represented by a series of points $\{(x_0, p(x_0)), ..., (x_{N-1}, p(x_{N-1}))\}$. For example, with 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 the polynomial, then we can just store the values $[p(x_0)...p(x_{N-1})]$ as a vector. We can choose any set of N input values. For our purposes we will choose the N-th roots of unity $\{1, \omega_N, ..., \omega_N^{N-1}\}$ as our set of inputs and store $[p(1)...p(\omega_N^{N-1})]$. This choice makes it efficient to convert between the coefficient and point-value representations.

Consider how we can convert from the coefficient representation to the point value representation. We have to evaluate the polynomial at each of the N inputs. Converting $[1, \omega_N, ..., \omega_N^{N-1}]$ to $[p(1), ..., p(\omega_N^{N-1})]$ requires the

Polynomial Data Structures

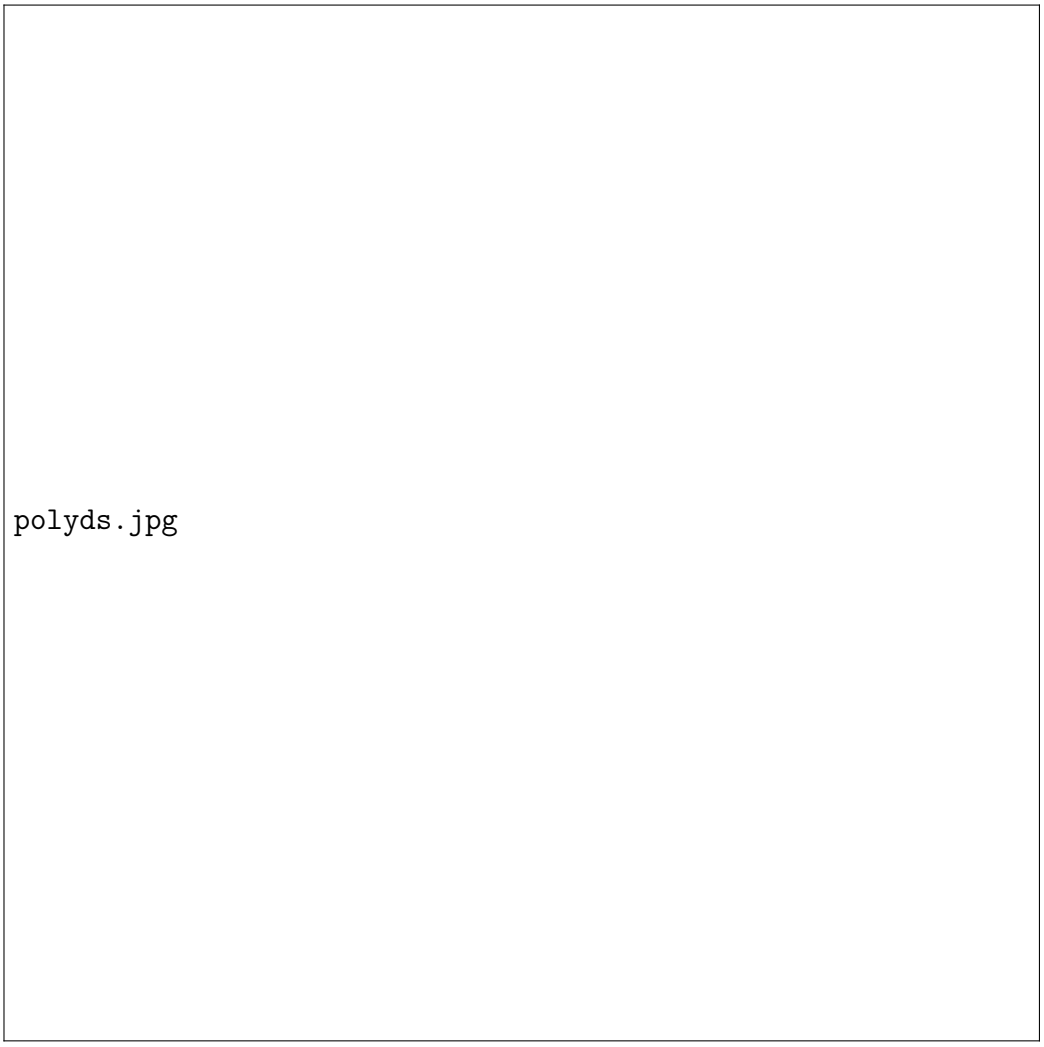

Figure 2: This figure depicts the two polynomial data structures and their quantum analogues whic are used in our explanation of QPE.

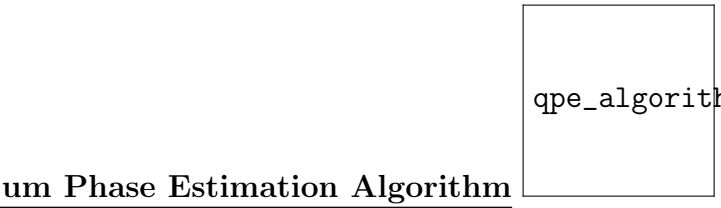

Figure 3: High level circuit explaining the Quantum Phase Estimation Algorithm. The algorithm has two steps. Step 1 prepares $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^k)^j\,|j\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(\omega_N^j)^k\,|j\rangle$, which is the amplitude encoding of the point-value representation of the polynomial x^k. Step 2 converts to the amplitude encoding of the coefficient representation, yielding k.

following transformation:

$$p(\omega_N^l) = \sum_{j=0}^{N-1} a_j \omega_N^{jl} = \sum_{j=0}^{N-1} a_j e^{i2\pi jl/N},$$

which is exactly the Discrete Fourier Transform.

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. The details of this are interesting but not essential for a first understanding of QPE, so we discuss it in the next section.

The second step of the QPE algorithm performs an inverse-QFT to convert from the amplitude encoding of the point-value representation to the amplitude encoding of the coefficient representation.

1.3 Algorithm

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:

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

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.

2 Notes

This explanation was motivated by reading Ref. [1].

References

[1] 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.