Title
Efficient phase-factor evaluation in quantum signal processing

Journal
Physical Review A, 103(4)

ISSN
2469-9926

Authors
Dong, Yulong
Meng, Xiang
Whaley, K Birgitta
et al.

Publication Date
2021-04-01

DOI
10.1103/physreva.103.042419

Peer reviewed

Efficient phase-factor evaluation in quantum signal processing

Yulong Dong[1,2], Xiang Meng[3], K. Birgitta Whaley[1,2], and Lin Lin[3,4]
[1] *Berkeley Center for Quantum Information and Computation, Berkeley, California 94720 USA*
[2] *Department of Chemistry, University of California, Berkeley, California 94720 USA*
[3] *Department of Mathematics, University of California, Berkeley, California 94720 USA and*
[4] *Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*
(Dated: July 13, 2021)

Quantum signal processing (QSP) is a powerful quantum algorithm to exactly implement matrix polynomials on quantum computers. Asymptotic analysis of quantum algorithms based on QSP has shown that asymptotically optimal results can in principle be obtained for a range of tasks, such as Hamiltonian simulation and the quantum linear system problem. A further benefit of QSP is that it uses a minimal number of ancilla qubits, which facilitates its implementation on near-to-intermediate term quantum architectures. However, there is so far no classically stable algorithm allowing computation of the phase factors that are needed to build QSP circuits. Existing methods require the usage of variable precision arithmetic and can only be applied to polynomials of relatively low degree. We present here an optimization based method that can accurately compute the phase factors using standard double precision arithmetic operations. We demonstrate the performance of this approach with applications to Hamiltonian simulation, eigenvalue filtering, and the quantum linear system problems. Our numerical results show that the optimization algorithm can find phase factors to accurately approximate polynomials of degree larger than $10,000$ with error below 10^{-12}.

I. INTRODUCTION

Recent progress in quantum algorithms has enabled construction of efficient quantum circuit representations for a large class of non-unitary matrices, which significantly expands the potential range of applications of quantum computers beyond the original goal of efficient simulation of unitary dynamics envisaged by Benioff [2] and Feynman [10]. The basic tool for representation of non-unitary matrices and hence of non-unitary quantum operators is called block-encoding [12]. It describes the process in which one embeds a non-unitary matrix A into the upper-left block of a larger unitary matrix U_A, and then expresses the quantum circuit in terms of U_A.

Computation of matrix functions, *i.e.*, evaluation of $F(A)$, where $F(x)$ is a smooth (real-valued or complex-valued) function, is a central task in numerical linear algebra [17]. Numerous computational tasks can be performed by generating approximations to matrix functions. These include application of a broad range of operators to quantum states: *e.g.*, $e^{-\mathrm{i}tA}$ for the Hamiltonian simulation problem; $e^{-\beta A}$ for the thermal state preparation problem; A^{-1} for the matrix inverse (also called the quantum linear system problem, QLSP); and the spectral projector of A for the principal component analysis, to name a few.

Several routes to construct a quantum circuit for $f(A)$ have been developed. These include methods using phase estimation (*e.g.*, the HHL algorithm [16] for the matrix inverse), the method of linear combination of unitaries (LCU) [3, 7], and the method of quantum signal processing (QSP) [12, 21, 23]. Among these methods, QSP stands out as so far the most general approach capable of representing a broad class of matrix functions via the eigenvalue or singular value transformations of A, while using a minimal number of ancilla qubits. The basic idea of QSP is to approximate the desired function $F(x)$ by a polynomial function $f(x)$, and then find a circuit to encode $f(A)$ *exactly* (assuming an exact block-encoding U_A). Treating the block-encoding U_A as an oracle, the application of QSP has given rise to asymptotically optimal Hamiltonian simulation algorithms [8, 14]. Applications have also been made to solving QLSP [12, 13], and to eigenvalue filtering [20]. In particular, the eigenvalue filtering approach of Ref. [20] does not directly approximate A^{-1}, but approximates a spectral projection operator, leading also to a quantum algorithm for solving QLSP with near-optimal complexity without the need of involving complex procedures such as variable time amplitude amplification [1].

Despite these fast growing successes, practical application of QSP on quantum computers, whether these are near- or long-term machines, still faces a significant challenge. A QSP circuit is defined using a series of adjustable phase factors. Once these phase factors are known, the QSP circuit can be directly implemented using U_A together with a set of multi-qubit control gates and single qubit phase rotation gates. However, the inverse problem, *i.e.*, finding the phase factors associated with a given polynomial function $f(x)$ is extremely difficult, to the extent that in practice very few applications of QSP have been made to date. The original work of Low and Chuang [21] demonstrated the existence of the phase factors but was not constructive. Initial efforts to find constructive procedures were not encouraging. Thus it was reported in [8] that it was prohibitive to obtain a QSP circuit of length that is larger than 30 for the Jacobi-Anger expansion [21] of the Hamiltonian simulation problem, and concluded "the difficulty of computing the angles needed to perform the QSP algorithm prevents us from taking full advantage of the algorithm in practice, so it would be useful to develop a more efficient classical procedure for specifying these angles".

The first constructive procedure to find phase factors was given in [12], with a procedure which requires a recur-

sive solution of roots of high degree polynomials to high precision, counting multiplicities of the roots. Therefore this procedure is not stable for representing high degree polynomials using QSP. Significant improvement has recently been made by Haah [13], who proposed a numerical algorithm to compute phase factors up to order ~ 1000, provided that all arithmetic operations can be computed with sufficiently high precision. Specifically, the number of classical bits needed for this scales as $\mathcal{O}(d\log(d/\epsilon))$, where d is the degree of the polynomial f, and ϵ is the target accuracy. Therefore the algorithm is still not *classically* numerically stable (a numerically stable algorithm should use no more than $\mathcal{O}(\text{poly}\log(d/\epsilon))$ classical bits) [18]. Haah's algorithm was implemented in Ref. [13] using Mathematica and employing the variable precision arithmetic capability of this. The running time is observed to be $\mathcal{O}(d^3)$.

In this paper, we demonstrate that the phase factors can be accurately determined with standard double precision arithmetic operations, even when the degree of the polynomial $f(x)$ is very high ($\gtrsim 10,000$) and when a very high precision (L^∞ error of function approximation $\lesssim 10^{-12}$) is required. We achieve this with a standard optimization approach that only minimizes a loss function, rather than recursively determining the phase terms. This minimization involves the multiplication of matrices in SU(2) and is thus numerically stable. We iteratively refine the phase factors to minimize the loss functions. However, since the optimization of the phase factors is a very nonlinear procedure, the initial guess must be carefully chosen. Indeed, if we randomly select the initial guess, the accuracy of the resulting phase factors is usually very low. We also find that under proper conditions, the QSP phase factors exhibit an inversion symmetry structure with respect to the center. This should be respected in the initial guess and preserved throughout the optimization procedure. We combine these two features to provide a simple, and yet highly effective choice of the initial guess.

We demonstrate here the performance of our optimization based approach to determine the phases for QSP algorithms with examples for Hamiltonian simulation, eigenstate filtering, and matrix inversion. We show that our algorithm can significantly outperform existing approaches using variable precision arithmetic operations [11, 13]. Numerical observation indicates that the computational cost of our method scales only quadratically as $\mathcal{O}(d^2)$, while the number of classical bits used remains constant (using the standard double precision, *i.e.*, 64 bits, arithmetic operations) as d increases.

We note that the previous algorithms for finding the phase factors require an analytic expansion of the smooth function $F(x)$ into polynomials. For instance, the Jacobi-Anger expansion is used for Hamiltonian simulation [13, 21]. When $F(x)$ is defined only on a sub-interval of $[-1,1]$, as for, *e.g.*, matrix inversion, where $F(x)=1/x$ is not well defined at $x=0$, one must first find an approximate smooth function and then perform expansion with respect to this approximate smooth function. Both steps introduce additional approximations and lead to inefficiencies in implementation. As an alternative, we propose here to use the Remez exchange algorithm [25] to directly find the minimax approximation to $F(x)$ on $[-1,1]$ or a given sub-interval. Our numerical evidence shows that this not only streamlines the process of finding QSP factors, but that the use of the Remez algorithm can also lead to polynomials of significantly lower degree.

Besides the inversion symmetry, we also find that the phase factors used for approximating smooth functions can decay rapidly away from the center. We find that the decay of the phase factors is directly linked to the decay of the coefficients in the Chebyshev expansion of the target function. This enables us to design a "phase padding" procedure, which identifies an initial guess of the QSP phase factors for a high degree polynomial, given the corresponding phase factors for a relatively low degree polynomial.

Throughout this paper we shall use the following notation: $N=2^n, M=2^m$, and $[N]=\{0,1,\ldots,N-1\}$, with n the number of logical qubits (also called system qubits), and m the number of qubits added to construct the unitary U_A. We shall refer to the latter as the "ancilla qubits for block-encoding", which is to be distinguished with additional ancilla qubits needed for quantum signal processing. T_d and R_d are Chebyshev polynomials of degree d of the first and second kind respectively. For a matrix A, the transpose, Hermitian conjugate and complex conjugate are denoted by $A^\top$, $A^\dagger$, A^*, respectively.

II. REVIEW OF QUANTUM SIGNAL PROCESSING

II.1. Block-encoding and qubitization

Block-encoding is a general technique to encode a non-unitary matrix on a quantum computer. Let $A\in\mathbb{C}^{N\times N}$ be an n-qubit Hermitian matrix. If we can find an $(m+n)$-qubit unitary matrix $U\in\mathbb{C}^{MN\times MN}$ such that

$$U_A=\begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix} \tag{1}$$

holds, *i.e.*, A is the upper-left matrix block of U_A, then we may get access to A via the unitary matrix U_A. In particular,

$$A=\left(\langle 0^m|\otimes I_n\right)U_A\left(|0^m\rangle\otimes I_n\right). \tag{2}$$

In general, the representation (2) may not exist, *e.g.*, when the operator norm $\|A\|_2$ is larger than 1. So the definition of block-encoding should be relaxed as follows [12, 21]: if we can find $\alpha,\epsilon\in\mathbb{R}_+$, a state $|G\rangle\in\mathbb{C}^M$, and an $(m+n)$-qubit matrix U_A such that

$$\|A-\alpha\left(\langle G|\otimes I_n\right)U_A\left(|G\rangle\otimes I_n\right)\|\le\epsilon, \tag{3}$$

then U_A is called an (α,m,ϵ)-block-encoding of A. Here $|G\rangle$ is referred to as the signal state (for block-encoding). Then Eq. (2) gives a $(1,m,0)$-block-encoding of A with

$|G\rangle = |0^m\rangle$. If U_A is Hermitian, it is called a Hermitian block-encoding. In particular, all the eigenvalues of a Hermitian block-encoding U_A are ± 1. For simplicity of presentation, in the following we present the explicit construction of block-encoding and qubitization for Hermitian U_A. We shall then briefly discuss the generalization to non-Hermitian U_A and refer the reader to Appendix C for full details of this.

As an example, assume that A is written as the linear combination of Pauli operators [3, 7] with real coefficients, as

$$A = \sum_{l\in[M]} c_l U_l, \quad c_l \geq 0. \tag{4}$$

Here U_l is a multi-qubit Pauli operator, which is unitary and Hermitian. We assume the availability of two oracles. The first one is the $(m+n)$-qubit select oracle:

$$U_{\mathrm{SEL}} = \sum_{l\in[M]} |l\rangle\langle l| \otimes U_l. \tag{5}$$

U_{SEL} implements the selection of the unitary U_l conditioned on the state of the m-qubit signal register. The second is the m-qubit prepare oracle that generates a specific superposition of the m-qubit signal states (note that $|l=0\rangle \equiv |0^m\rangle$):

$$U_{\mathrm{PREP}} |0^m\rangle = \frac{1}{\sqrt{\|c\|_1}} \sum_{l\in[M]} \sqrt{c_l} |l\rangle, \tag{6}$$

where the 1-norm is $\|c\|_1 = \sum_{l\in[M]} |c_l|$. Then defining

$$U_A = (U_{\mathrm{PREP}}^\dagger \otimes I_n) U_{\mathrm{SEL}} (U_{\mathrm{PREP}} \otimes I_n), \tag{7}$$

we may verify that U_A is a $(\|c\|_1, m, 0)$-Hermitian block encoding of A.

We also define

$$U_\Pi = 2|0^m\rangle\langle 0^m| \otimes I_n - I_m \otimes I_n. \tag{8}$$

Both U_Π and U_A are unitary and Hermitian. Then Jordan's lemma [19] states that the entire Hilbert space $\mathcal{H} = \mathbb{C}^{MN}$ can be decomposed into orthogonal subspaces $\mathcal{H}_j$ invariant under U_Π and U_A, where each $\mathcal{H}_j$ has dimension 1 or 2. Restricted to each irreducible two-dimensional subspace $\mathcal{H}_j$, with a properly chosen basis denoted by $\mathcal{B}_j$, the matrix representations of U_Π and U_A are

$$[U_\Pi]_{\mathcal{B}_j} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, [U_A]_{\mathcal{B}_j} = \begin{pmatrix} \lambda_j & -\sqrt{1-\lambda_j^2} \\ -\sqrt{1-\lambda_j^2} & -\lambda_j \end{pmatrix}. \tag{9}$$

Here $\lambda \in [-1, 1]$, and a potential phase factor in the off diagonal elements of $[U_A]_{\mathcal{B}_j}$ can be absorbed into the choice of the basis. It is worth noting that we can always choose $[U_\Pi]_{\mathcal{B}_j}$ to be a σ_z matrix. Given the eigendecomposition $\alpha^{-1}A = \sum_{j\in[N]} \lambda_j |\psi_j\rangle\langle\psi_j|$, there are exactly N such two-dimensional subspaces $\mathcal{H}_j$ of the full Hilbert space $\mathcal{H}$. Each subspace is associated with a vector $|0^m\rangle|\psi_j\rangle$ in the $(m+n)$-qubit space and Eq. (9) gives

$$(\langle 0^m| \otimes I_n) U_A |0^m\rangle |\psi_j\rangle = \lambda_j |\psi_j\rangle = \alpha^{-1} A |\psi_j\rangle. \tag{10}$$

Each subspace $\mathcal{H}_j$ is also the invariant subspace of the operator $\Omega := U_\Pi U_A$, which is referred to as the *iterate* [22]. Furthermore, when restricted to $\mathcal{H}_j$, the iterate Ω is a rotation matrix with eigenvalues $e^{\pm \mathrm{i} \arccos(\lambda_j)}$. Then the combined space $\oplus_{j\in[N]} \mathcal{H}_j$ forms a $2N$-dimensional subspace of $\mathcal{H}$. This introduces an additional ancillary qubit, so that the total number of qubits is now $n+m+1$. Each eigenvalue λ_j is associated with two branches and hence with an SU(2) matrix via the mapping $\lambda_j = \cos\theta_j \mapsto e^{\pm \mathrm{i}\theta_j}$. This technique is called qubitization [22].

Although the decomposition in Eq. (9) formally involves the eigenvalue λ_j of A and the proper basis $\mathcal{B}_j$, it is important that we do not necessarily need the eigendecomposition of A explicitly. In fact, the key advantage of qubitization is that one can perform the eigenvalue transformations for all eigenvalues simultaneously by means of the quantum signal processing approach.

II.2. Quantum signal processing

Given the above constructions of block-encoding and qubitization, quantum signal processing (QSP) then considers the following parameterized circuit consisting of d iterates and $d+1$ rotations that are interleaved in alternating sequence:

$$U_{\widetilde{\Phi}} = \left[\prod_{i=0}^{d-1} (e^{\mathrm{i}\tilde{\phi}_i U_\Pi} U_\Pi U_A)\right] e^{\mathrm{i}\tilde{\phi}_d U_\Pi}. \tag{11}$$

Here $\tilde{\phi}_i \in \mathbb{R}$, and $\widetilde{\Phi} = (\tilde{\phi}_0, \ldots, \tilde{\phi}_d)$ is the vector of phase factors that will specify the polynomial $f(x)$ approximating the desired function $F(x)$. The use of the notation $\tilde{\phi}$ here is due to the fact that there are multiple sets of phase factors, which can be deduced from each other. In this section we use different notations such as $\tilde{\phi}, \phi, \varphi$ to distinguish these phase factors, and record their relation explicitly.

We now summarize the construction of these phase factors for a non-unitary but Hermitian operator A, according to the approach of Ref. [12]. For any $\tilde{\phi} \in \mathbb{R}$ and n-qubit state $|\psi\rangle$, we have

$$e^{\mathrm{i}\tilde{\phi} U_\Pi} |0^m\rangle |\psi\rangle = e^{\mathrm{i}\tilde{\phi}} |0^m\rangle |\psi\rangle.$$

For any m-qubit state $|\perp^m\rangle$ satisfying $\langle 0^m| \perp^m\rangle = 0$, we have

$$e^{\mathrm{i}\tilde{\phi} U_\Pi} |\perp^m\rangle |\psi\rangle = e^{-\mathrm{i}\tilde{\phi}} |\perp^m\rangle |\psi\rangle.$$

Therefore

$$U_\Pi = -\mathrm{i} e^{\mathrm{i}\frac{\pi}{2} U_\Pi}.$$

We may then absorb U_Π into the rotation matrix as

$$U_{\tilde{\Phi}} = (-\mathrm{i})^d \left[\prod_{i=0}^{d-1} (e^{\mathrm{i}\varphi_i U_\Pi} U_A)\right] e^{\mathrm{i}\varphi_d U_\Pi}. \tag{12}$$

Here we have redefined the phase factors as $\varphi_i = \tilde{\phi}_i + \frac{\pi}{2}$ for $i = 0, \ldots, d-1$, and $\varphi_d = \tilde{\phi}_d$. The global phase factor $(-\mathrm{i})^d$ can be optionally discarded and we shall do so below.

Then we may readily check that the matrix $e^{\mathrm{i}\varphi U_\Pi}$ has a $(1,1,0)$-block-encoding as illustrated in Fig. 1. Here the control gate represents an $(m+1)$-qubit Toffoli gate (with the usual convention that open circles represent the target qubit being flipped when the control bits are zero).

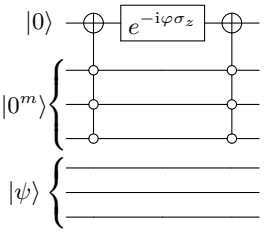


FIG. 1: Quantum circuit for block-encoding $e^{\mathrm{i}\varphi U_\Pi}$. The three distinct groups of lines represent 1,m,n qubits, respectively.

Using the circuit in Fig. 1, we may then implement the $(n+m)$-qubit unitary operator $U_{\tilde{\Phi}}$ of Eq. (12) using only one additional ancilla qubit and the circuit in Fig. 2 [11].

Ref. [11] investigated the general question as to which class of functions can be block-encoded by $U_{\tilde{\Phi}}$ for some choice of phase factors. First, each $\mathcal{H}_j$ is an invariant subspace of $U_{\tilde{\Phi}}$. So the upper-left element of $U_{\tilde{\Phi}}$ acting on $\mathcal{H}_j$ is a function of the eigenvalue λ_j. Thus we see that qubitization reduces the problem of representing a matrix function on an n-qubit system to a representation problem in SU(2), which can be carried out on classical computers. We now state main theorem of QSP from Ref. [11] below in Theorem 1.

Theorem 1. (**Quantum Signal Processing in SU(2)** [11, Theorem 3]) For any $P, Q \in \mathbb{C}[x]$ and a positive integer d such that *(1)* $\deg(P) \leq d, \deg(Q) \leq d-1$, *(2)* P has parity $(d \mod 2)$ and Q has parity $(d-1 \mod 2)$, *(3)* $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1, \forall x \in [-1,1]$. Then, there exists a set of phase factors $\Phi := (\phi_0, \cdots, \phi_d) \in [-\pi, \pi)^{d+1}$ such that

$$\begin{aligned} U_\Phi(x) &= e^{\mathrm{i}\phi_0\sigma_z} \prod_{j=1}^{d} \left[W(x) e^{\mathrm{i}\phi_j\sigma_z}\right] \\ &= \begin{pmatrix} P(x) & \mathrm{i}Q(x)\sqrt{1-x^2} \\ \mathrm{i}Q^*(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix} \end{aligned} \tag{13}$$

where

$$W(x) = e^{\mathrm{i}\arccos(x)\sigma_x} = \begin{pmatrix} x & \mathrm{i}\sqrt{1-x^2} \\ \mathrm{i}\sqrt{1-x^2} & x \end{pmatrix}.$$

The proof of Theorem 1 is constructive and, as shown explicitly in Ref. [11], it yields an algorithm to compute the phase factor vector Φ once the polynomials $P, Q \in \mathbb{C}[x]$ are given. The algorithm of Ref. [11] is summarized in Appendix G (Algorithm 5, with modifications to enhance the numerical stability). We note that these phase factors are unique, modulo certain trivial equivalence relations (Appendix A).

In order to connect Theorem 1 with the representation of $U_{\tilde{\Phi}}$ in Eq. (11), we consider the matrix representation of $U_{\tilde{\Phi}}$ restricted to $\mathcal{H}_j$, let $x = \lambda_j$, and use the following identity

$$e^{\mathrm{i}\arccos(x)\sigma_x} = e^{-\mathrm{i}\frac{\pi}{4}\sigma_z} \begin{pmatrix} x & -\sqrt{1-x^2} \\ \sqrt{1-x^2} & x \end{pmatrix} e^{\mathrm{i}\frac{\pi}{4}\sigma_z}. \tag{14}$$

Hence to connect Eq. (13) with Eq. (11), we have $\tilde{\phi}_0 = \phi_0 - \frac{\pi}{4}$, $\tilde{\phi}_d = \phi_d + \frac{\pi}{4}$, and $\tilde{\phi}_i = \phi_i$ for $1 \leq i \leq d-1$. Therefore, the relation between the phase factors $\{\phi_i\}_{i=0}^d$ in Theorem 1 (Eq. (13)) and the phase factors $\{\varphi_i\}_{i=0}^d$ appearing in $U_{\tilde{\Phi}}$ of Eq. (11) and in the implementation of the QSP circuit in Fig. 2, is given by

$$\varphi_i = \begin{cases} \phi_0 + \frac{\pi}{4}, & i = 0, \\ \phi_i + \frac{\pi}{2}, & 1 \leq i \leq d-1, \\ \phi_n + \frac{\pi}{4}, & i = d. \end{cases} \tag{15}$$

II.3. Representing general matrix polynomials

Now given a degree d polynomial $P(x) \in \mathbb{C}[x]$ satisfying the requirement of Theorem 1, for any $(\alpha, m, 0)$ Hermitian-block-encoding of A, the circuit in Fig. 2 yields a $(1, m+1, 0)$-block-encoding of $P(A/\alpha)$. With some abuse of notation, we shall denote both this block-encoding of the polynomial function of A and the associated QSP circuit by U_Φ. The QSP circuit uses d queries of U_A and $\mathcal{O}((m+1)d)$ other primitive quantum gates.

We should remark that the condition (3) in Theorem 1 imposes very strong constraints on P, Q that are nontrivial to satisfy. Therefore we consider the following cases separately on how to construct QSP circuits in practice.

Case 1. In many applications, we are interested in computing $f(A/\alpha)$, where $f(x)$ is a real polynomial. It is stated in [11, Theorem 5] that for $f \in \mathbb{R}[x]$ satisfying *(1), (2)* and $|f(x)| \leq 1, \forall x \in [-1,1]$, there exists $P \in \mathbb{C}[x], Q \in \mathbb{R}[x]$ such that $\mathrm{Re}[P(x)] = f(x)$. The choice of P, Q may not be unique. This only gives the block-encoding of $P(A/\alpha)$. In order to obtain the block-encoding of $f(A/\alpha)$, we can use the linear combination of unitaries (LCU) technique to separate the real and imaginary parts of $P(x)$ as follows. Note that

$$f(x) = \frac{1}{2}(P(x) + P^*(x)). \tag{16}$$

If the upper-left entry of $U_\Phi(x)$ is $P(x)$ as in Eq. (13),

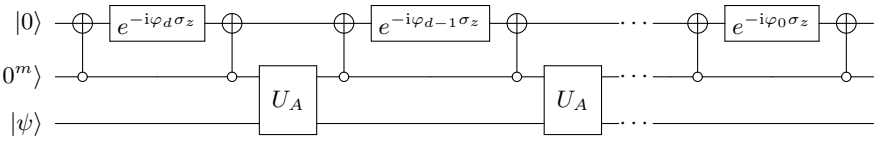


FIG. 2: Quantum circuit for quantum signal processing of a general matrix polynomial with a Hermitian block-encoding U_A.

then

$$U_\Phi^*(x) = e^{-\mathrm{i}\phi_0\sigma_z}\prod_{j=1}^{d}\left[W^*(x)e^{-\mathrm{i}\phi_j\sigma_z}\right]$$
$$= \begin{pmatrix} P^*(x) & -\mathrm{i}Q^*(x)\sqrt{1-x^2} \\ -\mathrm{i}Q(x)\sqrt{1-x^2} & P(x) \end{pmatrix}$$

Here $U_\Phi^*(x)$ is the complex conjugation of $U_\Phi(x)$, and hence its upper-left entry of is $P^*(x)$. From

$$W^*(x) = e^{\mathrm{i}\frac{\pi}{2}\sigma_z}W(x)e^{-\mathrm{i}\frac{\pi}{2}\sigma_z},$$

we find that $U_\Phi^*(x) = U_{-\Phi}(x)$, where the negative phase factors are defined by

$$-\Phi := \left(-\phi_0 + \frac{\pi}{2}, -\phi_1, \cdots, -\phi_{d-1}, -\phi_d - \frac{\pi}{2}\right), \quad (17)$$

which simply negates each phase factor except for ϕ_0 and ϕ_d. In order to find a block-encoding of $\frac{1}{2}(U_\Phi + U_{-\Phi})$, we can introduce one additional ancilla qubit to the signal register. The prepare oracle U_{PREP} is simply the Hadamard gate H. Fig. 3 gives the circuit for the $(1, m+2, 0)$-block-encoding of $f(A/\alpha)$. This technique is also called the addition of block-encodings [12]. Note that according to Eq. (15), the negative phase factors $-\Phi$ should be implemented using the circuit in Fig. 2 with

$$\varphi_i = \begin{cases} -\phi_0 + \frac{3\pi}{4}, & i = 0, \\ -\phi_i + \frac{\pi}{2}, & 1 \le i \le d-1, \\ -\phi_d - \frac{\pi}{4}, & i = d. \end{cases} \quad (18)$$

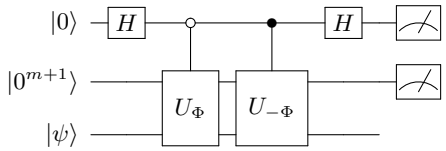


FIG. 3: Quantum circuit for block-encoding of $f(A/\alpha)$ using LCU to separate real and imaginary parts of $f(x)$. The three horizontal lines represent 1,$m+1$,n qubits, respectively. The circuits U_Φ, $U_{-\Phi}$, are shown, after proper transformation of the phase factors, in Fig. 2.

Case 2. The real polynomial $f(x)$ in case 1 is assumed to have definite parity. For a general real polynomial without parity constraints, we may use the decomposition

$$f(x) = f_{\mathrm{even}}(x) + f_{\mathrm{odd}}(x), \quad (19)$$

where $f_{\mathrm{even}}(x) = \frac{1}{2}(f(x) + f(-x)), f_{\mathrm{odd}}(x) = \frac{1}{2}(f(x) - f(-x))$. If $|f(x)| \le 1$ on $[-1,1]$, then $|f_{\mathrm{even}}(x)|, |f_{\mathrm{odd}}(x)| \le 1$ on $[-1,1]$, and $f_{\mathrm{even}}(x)$, $f_{\mathrm{odd}}(x)$ can be each constructed using the circuit in Fig. 3. Introducing another ancilla qubit and using the same form of the LCU circuit in Fig. 3 (the $U_\Phi, U_{-\Phi}$ circuits should be replaced by the QSP circuits for even and odd parts, respectively), we find a $(2, m+3, 0)$-block-encoding of $f(A/\alpha)$. Equivalently, we have a $(1, m+3, 0)$-block-encoding of $\frac{1}{2}f(A/\alpha)$.

Case 3. The most general case is that $f(x) \in \mathbb{C}[x]$ is a complex polynomial. Let $f(x) = g(x) + \mathrm{i}h(x)$ where $g, h \in \mathbb{R}[x]$ are the real and imaginary parts of $f(x)$, respectively. We remark that even when $h = 0$ (*i.e.*, $f(x)$ is a real polynomial), the associated polynomial $P(x)$ might have a non-vanishing imaginary component. Therefore in general we cannot expect to find phase factors that simultaneously encode $g(x) + \mathrm{i}h(x)$, even if $f(x)$ has definite parity. Hence we need to use LCU once again to find the block-encoding of f through the linear combination of block-encodings of g and $\mathrm{i}h$, respectively. Assuming $|g(x)|, |h(x)| \le 1$ on $[-1,1]$, following case 2, we have a $(2, m+3, 0)$-block-encoding of $g(A/\alpha)$ denoted by U_g. Similarly a circuit of the form in Fig. 4 gives the $(2, m+3, 0)$-block-encoding of $\mathrm{i}h(A/\alpha)$ denoted by $U_{\mathrm{i}h}$.

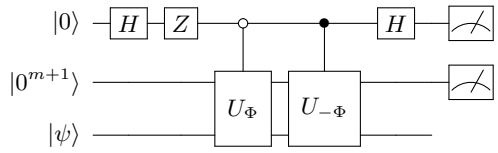


FIG. 4: Quantum circuit for block-encoding of $\mathrm{i}h(A/\alpha)$ using linear combination of unitaries. The three lines represent 1,$m+1$,n qubits, respectively. The circuit U_Φ, $U_{-\Phi}$, after proper transformation of the phase factors, is given in Fig. 2.

We can use the LCU circuit of the form in Fig. 3, with the $U_\Phi, U_{-\Phi}$ circuits now replaced by U_g and $U_{\mathrm{i}h}$, respectively, to ensure that the prepare oracle is still the Hadamard gate. This gives a $(4, m+4, 0)$-block-encoding of $f(A/\alpha)$.

We now make some general remarks on the block-encoding of matrix polynomials. First, while LCU is a general technique for implementing addition of block-encodings, when block-encoding a real polynomial as in case 1 above, one can actually save an ancilla qubit by taking advantage of the special structure of QSP circuits (see Appendix B). A similar implementation ex-

ists for an imaginary polynomial, using a Z gate as in Fig. 4. This reduces the number of additional ancilla qubits by 1 for all cases discussed above, and the number of ancilla qubits then matches the results in [12]. Second, although the concept of qubitization and QSP were introduced here for Hermitian block-encodings in order to make use of Jordan's lemma, all the constructions shown above can be generalized to non-Hermitian block-encodings. One possible procedure to achieve this is described in Appendix C, which requires only use of one additional ancilla qubit. We note here that an alternative procedure is to use the quantum singular value transformation, which removes the need of this ancilla qubit and leads to a slightly simpler circuit, as well as allowing treatment of the case when A is not a Hermitian matrix [12]. For simplicity all further discussion in this paper assumes that an $(\alpha, m, 0)$-block-encoding U_A is available. When the block-encoding itself is not error-free, *i.e.*, U_A is an (α, m, ϵ)-block-encoding of A, the cumulative error in the QSP circuit can also be analyzed. We refer readers to [11, 12] for more details.

II.4. Direct methods for finding phase factors

According to Section II.3, case 1 is the most important step, since cases 2 and 3 can simply be obtained from applying case 1 repeatedly and using the LCU technique. In fact, the proof of Theorem 5 in [11] also provides a constructive method for finding the phase factors, as follows. Given a properly normalized real polynomial with definite parity $f(x)$, one may first reconstruct complementing polynomials $B(x), C(x) \in \mathbb{R}[x]$ to form $P = f + \mathrm{i}B, Q = C$ satisfying the requirement in Theorem 1. This can be done by solving all the roots (including multiplicities) of the polynomial $1 - f(x)^2$ [11, Lemma 6]. Then one can use a reduction method to find the phase factors. This procedure will be referred to as the GSLW method. This procedure is exact if all floating point arithmetic operations can be performed with infinite precision, but is numerically unstable with standard double precision arithmetic operations. One disadvantage of the GSLW method is that it is based on the Taylor expansion of high order polynomials, which can be numerically highly unstable when the degree of polynomials becomes large.

To improve the numerical stability of the GSLW method, another algorithm was proposed in [13], which we will refer to as the Haah method. In the Haah method, the polynomials defined on $[-1, 1]$ are mapped to the unit circle via the transformation $x \mapsto e^{\pm \mathrm{i} \arccos(x)}$, and then extended to the complex plane. Such treatment is equivalent to a Chebyshev polynomial expansion, which improves the numerical stability over the GSLW method which uses the standard basis $\{1, x, x^2, \ldots\}$. Then, a similar reduction procedure is used to deduce the phase factors. However, one still needs to find the roots of a polynomial of high degree, and the number of classical bits required for this is $\mathcal{O}(d \log d)$, where d is the degree of polynomial.

In both the GSLW method and the Haah method, the phase factors are obtained from a single shot calculation. Therefore we refer to them as the direct methods for finding phase factors. This is in contrast to the optimization based method to be introduced below, which finds the phase factors via an iterative procedure.

The performance of the GSLW method has also been improved by a more recent work [5] after this paper was posted. The improved method of [5] is still based on direct factorization of polynomials. However, it is found that the numerical stability can be empirically improved using a method called "capitalization", which adds a small perturbation to the leading order term of the target polynomial. Together with another technique called "halving", the method of [5] can find a sequence with more than 3000 phase factors with double precision arithmetic operations. This result indicates that the sensitivity of the phase factors with respect to perturbation of the target polynomials is still not well understood. Our optimization-based algorithm below presents a very different approach to determining the phase factors, which can achieve machine precision directly without perturbing the target polynomials and which is thus not limited by stability of such procedures. We show that with the optimization approach up to 10,000 phase factors can be determined with error less than 10^{-12}.

III. OPTIMIZATION BASED METHOD FOR FINDING PHASE FACTORS

Both the GSLW and the Haah methods are limited by the usage of root-finding and matrix reduction procedure, which result in the numerical instability when the degree of polynomials becomes large. Here we consider an alternative strategy to find the phase factors, by direct minimization with respect to a certain distance function,

$$L(\Phi) := \mathrm{dist}\left\{\mathrm{Re}\left[\langle 0 | U_\Phi(x) | 0 \rangle\right], f(x)\right\}. \tag{20}$$

In practice, the distance function will be characterized by the mean squared loss over discrete sample points. When $L(\Phi^*)$ is zero, we obtain the desired phase factors through the minimizer Φ^*. This strategy bypasses the difficulty of constructing the complementing polynomials that relies on the high-precision root-finding procedure. Because the computation of the gradient and the Hessian matrix of the objective function only involve the matrix multiplications in SU(2), which is a numerically stable procedure, the optimization scheme is expected to significantly improve the robustness of the algorithm. This will be verified by our numerical tests. It also ensures an efficient optimization.

In the following discussion, we use P, Q as the polynomials involved in the QSP unitary matrix in Eq. (13). Let $\mathcal{C}_{d+1} \subset [-\pi, \pi)^{d+1}$ be the irreducible set of phase factors with $d + 1$ entries. The pair of polynomials $P(x), Q(x) \in \mathbb{C}[x]$ satisfying conditions in Theorem 1

determines a unique set of phase factors $\Phi \in \mathcal{C}_{d+1}$ (see Appendix A).

We again only consider a properly normalized real polynomial with definite parity $f(x)$ as in case 1 of Section II.3. Because the form of $Q(x)$ is not of interest, we may restrict $Q(x) \in \mathbb{R}[x]$.

III.1. Symmetry property of the phase factors

Given a set of QSP factors Φ, let the inverse phase factors be defined as

$$\Phi^- = (\phi_d, \phi_{d-1}, \cdots, \phi_0). \tag{21}$$

The inverse phase factors should not be confused with the negative phase factors $-\Phi$ in Eq. (17).

Theorem 2 states that when we choose $Q(x)$ to be a real polynomial, the phase factors are symmetric under inversion.

Theorem 2 (Inversion Symmetry). 1) If $\Phi = \Phi^-$, then $Q \in \mathbb{R}[x]$. 2) If $Q \in \mathbb{R}[x]$, then we may choose $\Phi \in \mathcal{C}_{d+1}$ such that $\Phi = \Phi^-$.

Proof. 1): Obviously,

$$\begin{aligned} U_{\Phi^-}(x) &= e^{\mathrm{i}\phi_d\sigma_z}\prod_{j=1}^{d}\left[W(x)e^{\mathrm{i}\phi_{d-j}\sigma_z}\right] = U_\Phi(x)^\top \\ &= \begin{pmatrix} P(x) & \mathrm{i}Q^*(x)\sqrt{1-x^2} \\ \mathrm{i}Q(x)\sqrt{1-x^2} & P^*(x) \end{pmatrix}. \end{aligned} \tag{22}$$

Then, the statement that Φ is invariant under inversion implies that $Q(x) = Q^*(x) \in \mathbb{R}[x]$.
2): If $Q \in \mathbb{R}[x]$, then $U_\Phi(x) = U_\Phi(x)^\top = U_{\Phi^-}(x)$. Expand P, Q in terms of Chebyshev polynomials, *i.e.*, $P(x) = \sum_j p_j T_j(x)$, $\sqrt{1-x^2}Q(x) = \sum_j q_j\sqrt{1-x^2}R_{j-1}(x)$. After a change of variable $x = \cos\theta$, P, Q are transformed to Fourier series in terms of $\cos(j\theta)$ and $\sin(j\theta)$ respectively. The continuation $\theta \mapsto 2\pi - \theta$ extends the QSP unitary consisting of P, Q to a $\mathrm{U}(1) \to \mathrm{SU}(2)$ function, after identifying θ with $e^{\mathrm{i}\theta} \in U(1)$. Moreover, the parity constraint implies that this function only has non-zero coefficients $j = -d, -d+2, \cdots, d-2, d$ with respect to $e^{\mathrm{i}j\theta}$. Appendix A shows that the set of phase factors is unique, up to the equivalence relation for the irreducible set $\mathcal{C}_{d+1}$. So $\Phi = \Phi^-$ up to equivalence relations. In particular, we may choose the phase factors such that $\Phi = \Phi^-$. □

As an example, let $P(x) = T_d(x)$, $Q(x) = R_{d-1}(x)$, the corresponding QSP phase factors are $\Phi = (\underbrace{0, 0, \cdots, 0}_{d+1})$. For $P(x) = \mathrm{i}T_d(x)$, $Q(x) = R_{d-1}(x)$, the phase factors are $\Phi = (\frac{\pi}{4}, \underbrace{0, \cdots, 0}_{d-1}, \frac{\pi}{4})$. In both cases, the polynomial Q is real. Thus, it is evident that the phase factors satisfy the inversion symmetry in Theorem 2.

The symmetry property allows us to reduce the number of degrees of freedom by a factor of 2, and also motivates the symmetric construction of phase factors in the optimization procedure later. The appearance of two $\pi/4$ factors in the example above can be justified by Lemma 3, which shows that the action of these phase factors interchanges the real and imaginary parts of the polynomial P up to a sign.

Lemma 3. Given a set of QSP phase factors Φ, the following relations hold point-wise for $x \in [-1, 1]$,

$$\begin{aligned} \mathrm{Re}\left[\langle 0|U_\Phi(x)|0\rangle\right] &= -\mathrm{Im}\left[\langle 0|e^{-\mathrm{i}\frac{\pi}{4}\sigma_z}U_\Phi(x)e^{-\mathrm{i}\frac{\pi}{4}\sigma_z}|0\rangle\right], \\ \mathrm{Im}\left[\langle 0|U_\Phi(x)|0\rangle\right] &= \mathrm{Re}\left[\langle 0|e^{-\mathrm{i}\frac{\pi}{4}\sigma_z}U_\Phi(x)e^{-\mathrm{i}\frac{\pi}{4}\sigma_z}|0\rangle\right]. \end{aligned}$$

Proof. Factorize the QSP unitary as $U_\Phi = a_0 I + a_1\sigma_z + a_2\sigma_x + a_3\sigma_y$. The algebra of Pauli matrices implies that $e^{-\mathrm{i}\frac{\pi}{4}\sigma_z}U_\Phi e^{-\mathrm{i}\frac{\pi}{4}\sigma_z} = a_0 e^{-\mathrm{i}\frac{\pi}{2}\sigma_z} + a_1 e^{-\mathrm{i}\frac{\pi}{2}\sigma_z}\sigma_z + a_2\sigma_x + a_3\sigma_y = -\mathrm{i}a_0 I - \mathrm{i}a_1\sigma_z + a_2\sigma_x + a_3\sigma_y$. Then, the conclusion follows. □

III.2. Choice of objective function

If the target smooth function $f(x)$ is not a polynomial, we first approximate $f(x)$ using a polynomial, and then feed the polynomial into the QSP solver. We would stress that this preprocessing step of polynomial approximation is necessary for the success of the optimization method. If we directly feed a non-polynomial function $f(x)$ into the objective function, then generally the equation $L(\Phi) = 0$ does not have a solution. Numerical evidence indicates that the landscape of the objective function is very complex and the optimization procedure can easily get stuck in one of the many local minima. On the other hand, for any polynomial satisfying conditions in Theorem 1, there always exists a set of QSP factors Φ^* so that $L(\Phi^*) = 0$. Our numerical results indicate that starting from a proper initial guess, the optimization procedure can be very robust.

Since $Q(x)$ is not involved in the distance function, we may require $Q(x) \in \mathbb{R}[x]$ and impose the inversion symmetry constraint (Theorem 2) on the phase factors. Under this constraint, the phase factors $\Phi = (\phi_0, \ldots, \phi_d)$ have $\lceil \frac{d+1}{2} \rceil$ degrees of freedom for optimization. As a result, it is reasonable to choose the approximation as a polynomial f of degree d with parity $(d \mod 2)$, which has the same number of adjustable coefficients. Theorem 1 and Theorem 2 together guarantee the existence of symmetric phase factors Φ such that $\mathrm{Re}\left[\langle 0|U_\Phi(\cdot)|0\rangle\right] = f$. In this case, the optimization over Φ towards the minimum value of the distance function can be viewed as a polynomial interpolation taking the QSP parameterization. These features suggest that the mean squared loss in terms of $\tilde{d} := \lceil \frac{d+1}{2} \rceil$ sample points on $(0, 1]$ provides an accurate enough characterization of distance function. Therefore, we can write objective function for optimiza-

tion as

$$L(\hat{\Phi}) = \frac{1}{\tilde{d}} \sum_{j=1}^{\tilde{d}} \left| \mathrm{Re}\left[\langle 0|U_\Phi(x_j)|0\rangle\right] - f(x_j) \right|^2, \tag{23}$$

where for $\hat{\Phi} = (\phi_0, \ldots, \phi_{\tilde{d}-1}) \in [-\pi, \pi)^{\tilde{d}}$,

$$\Phi = \begin{cases} (\phi_0, \cdots, \phi_{\tilde{d}-1}, \phi_{\tilde{d}-1}, \cdots, \phi_0) & d \text{ is odd,} \\ (\phi_0, \cdots, \phi_{\tilde{d}-2}, \phi_{\tilde{d}-1}, \phi_{\tilde{d}-2}, \cdots, \phi_0) & d \text{ is even.} \end{cases} \tag{24}$$

We choose $x_j = \cos\left(\frac{(2j-1)\pi}{4\tilde{d}}\right)$, $j = 1, \ldots, \tilde{d}$ as the positive roots of the Chebyshev polynomial $T_{2\tilde{d}}(x)$. Theorem 4 shows that using the Chebyshev nodes, the accuracy of the polynomial approximation can be directly measured in terms of the objective function (the proof is given in Appendix D).

Theorem 4. Suppose we have the following expansions:

$$f(x) = \sum_{j=0}^{d} \alpha_j T_j(x), \;\; f_\Phi(x) = \sum_{j=0}^{d} \beta_j T_j(x),$$

where $f_\Phi(x) = \mathrm{Re}\left[\langle 0\,|U_\Phi\,(x)|\,0\rangle\right]$. If the discrete samples are chosen to be positive roots of $T_{2\lceil \frac{d+1}{2} \rceil}(x)$ and $L(\hat{\Phi}) \leq \epsilon$, then we have

$$\max_{j=1,\ldots,d} |\alpha_j - \beta_j| \leq 2\sqrt{\epsilon}.$$

Note that the optimal phase factors are not necessarily unique. This is because the real part of P does not uniquely determine P, Q, even when assuming Q is real. Nonetheless, we only need to find one set of phase factors Φ^* to accurately encode $f(x)$.

Our optimization problem can be viewed as variational quantum circuit (more specifically, similar to the quantum approximate optimization algorithm (QAOA) [9]), in which one set of quantum gates (those associated with σ_x) are fixed. Due to the complex energy landscape, a good initial guess is necessary for the performance of the optimizer.

III.3. Generating approximation polynomials

In order to generate a polynomial to approximate f to a given degree, we consider in this work two efficient approaches: the Fourier-Chebyshev expansion method and the Remez method.

For a real smooth function F on the interval $[-1, 1]$, we find its polynomial approximation in terms of Chebyshev polynomial of the first kind, *i.e.*, $F(x) \approx f(x) = \sum_{j=0}^{d} c_j T_j(x)$. The Fourier approach uses the fast Fourier transformation (FFT) to efficiently evaluate the coefficients via a quadrature

$$c_j \approx \frac{(2-\delta_{j0})}{2K} (-1)^j \sum_{l=0}^{2K-1} F\left(-\cos\theta_l\right) e^{\mathrm{i}j\theta_l} \tag{25}$$

where $\theta_l = \pi l/K, 0 \leq l \leq 2K-1$, and K is the number of quadrature points.

We may alternatively consider optimization with respect to the L^∞ norm. In fact, we may even restrict the interval of approximation to be a subset $[a, b] \in [-1, 1]$. In this case, an approximation polynomial can be obtained by solving the optimal approximation problem in terms of the L^∞ norm

$$f = \operatorname*{argmin}_{f \in \mathbb{R}[x], \deg(f) \leq d} \max_{x \in [a,b]} |F(x) - f(x)|. \tag{26}$$

The Remez algorithm [6, 25] allows efficient solution of Eq. (26). This is an iterative method consisting of two steps. In the first step, we find the coefficients of f from $d+2$ points sampled from the interval by solving a set of linear equations. The second step involves adjusting $d+2$ samples from coefficients solved in the first step. We can also use the Remez algorithm to solve for f using parity constraint. Full details are given in Appendix E.

III.4. Choice of initial point

The objective function in the optimization model of Eq. (23) is highly non-convex, rendering the global minimum hard-to-find. Numerical tests given in Section IV.4 illustrate that the solver can easily get stuck in a local minimum if we initiate it randomly, confirming the complexity of the landscape. Another possible choice of the initial phase factors is $\Phi = (0, 0, \ldots, 0, 0)$. Then the components of QSP matrix are Chebyshev polynomials $P(x) = T_d(x)$ and $Q(x) = R_{d-1}(x)$. However, straightforward computation shows that in this case we have $\nabla L(\hat{\Phi}) = 0$, *i.e.*, $\hat{\Phi}$ is a stationary point, and obviously $L(\hat{\Phi}) \neq 0$.

Our main observation is that if we slightly modify the initial point as

$$\Phi = \left(\frac{\pi}{4}, 0 \ldots, 0, \frac{\pi}{4}\right) \in \mathbb{R}^{d+1}, \tag{27}$$

or correspondingly, the symmetrized version

$$\hat{\Phi}^0 = \left(\frac{\pi}{4}, 0 \ldots, 0\right) \in \mathbb{R}^{\tilde{d}}, \tag{28}$$

then a gradient-based algorithm can reach a global minimum in all cases shown in Section IV. According to the discussion in Section III.1, this corresponds to the initial guess with $P(x) = \mathrm{i}T_d(x)$ and $Q(x) = R_{d-1}(x)$. The intuitive reason for choosing such an initial point is that we are interested in the real part of $P(x)$. The choice in Eq. (27) ensures that $\mathrm{Re}[P(x)] = 0$, which is unbiased with respect to the function to be approximated. On the other hand, the seemingly natural choice $\Phi = (0, 0, \ldots, 0, 0)$ gives $P(x) = T_d(x)$, which is a heavily biased initial guess of the real component. The theoretical study of the landscape around such an initial guess justifying the effectiveness of such a choice of the initial guess will be the focus of future work.

III.5. Algorithm

We use a quasi-Newton method to perform numerical optimization of the phase factors. Compared to the Newton type method, we find that a quasi-Newton method such as the L-BFGS method [26, Chapter 5] leads to fast convergence without any need to evaluate the Hessian matrix, for which the computational cost would scale as $\mathcal{O}(d^3)$. Appendix F describes the L-BFGS algorithm, which is applied to the symmetry-reduced phase factors according to Eq. (23). Using the initial phase factors in Eq. (28), the Hessian matrix $\mathrm{Hess}\, L(\hat{\Phi}^0)$ is a constant matrix regardless of approximation polynomial f. More specifically, we have

$$\mathrm{Hess}\, L(\hat{\Phi}^0) = \begin{cases} 2I & d \text{ is odd}, \\ \mathrm{diag}(2,\ldots,2,1) & d \text{ is even}. \end{cases} \tag{29}$$

The inverse of this Hessian matrix will be fed into the L-BFGS algorithm. In Algorithm 1 below we describe how to compute optimal phase factors corresponding to a given polynomial. The complete procedure to approximate a generic complex-valued function as polynomial components is presented in Algorithm 2.

ALGORITHM 1: **Function:** $\hat{\Phi} = \mathrm{QSPBFGS}(\hat{\Phi}^0, f, \epsilon)$

Input: An initial vector $\hat{\Phi}^0$, a real polynomial f of degree d and error tolerance ϵ.

Choose $\tilde{d} = \lceil \frac{d+1}{2} \rceil$ points $x_j = \cos(\frac{(2j-1)\pi}{4\tilde{d}})$ as the positive roots of Chebyshev polynomial $T_{2\tilde{d}}$.
Construct objective function $L(\hat{\Phi})$ using Eq. (23).
Choose the initial approximation of inverse Hessian B_0 using Eq. (29).
Set $t = 0$
while $L(\hat{\Phi}) > \epsilon$ **do**
  Obtain $\hat{\Phi}^{t+1}$ by updating $\hat{\Phi}^t$ via L-BFGS algorithm.
  Set $t = t + 1$.
end while
Return: $\hat{\Phi}^t$

ALGORITHM 2: Finding phase factors for the polynomial approximation of a smooth function f over interval $[a, b]$

Input: A complex-valued function $F \in C^\infty[a, b]$, a non-negative integer d and error tolerance ϵ.

Find polynomial $f \in \mathbb{C}[x]$ of degree at most n which approximates f over the interval $[a, b]$. One can obtain such polynomial via the Fourier-Chebyshev expansion approach or the Remez algorithm [6, 25].
Scale f by a constant factor α.
Denote $f_j, j = 1, 2, 3, 4$ as real/imaginary and even/odd part of f/α.
Set $\hat{\Phi}^0 = (\frac{\pi}{4}, 0, \ldots, 0) \in \mathbb{R}^{\tilde{d}}$.
Solve $\hat{\Phi}_j = \mathrm{QSPBFGS}(\hat{\Phi}^0, f_j, \epsilon)$ for each component.
Return: $\hat{\Phi}_j, j = 1, 2, 3, 4$ and factor α.

IV. NUMERICAL RESULTS

We present a number of tests to examine the effectiveness of the optimization based method compared to the previous direct methods. We implement the direct algorithms designed in [12] and [13] (denoted here as the GSLW and Haah methods, respectively). All numerical tests are performed on an Intel Core 4 Quad CPU at 2.30 GHZ with 8 GB of RAM. Our method is implemented in MATLAB R2018b, while the GSLW and the Haah method are written in Julia 1.2 for its better support for high-precision arithmetic. Our implementation (optimization, GSLW, Haah) can be downloaded from the Github repository[1].

We utilize the BigFloat type to achieve variable precision arithmetic and internal routines in Julia for the root-finding procedures. In Appendix G, we present the details of algorithms used for comparison and state some modifications to enhance the numerical stability. The stopping criterion is

$$\max_{j=1,\ldots,\tilde{d}} |\mathrm{Re}\left[\langle 0 | U_\Phi(x_j) | 0 \rangle\right] - f(x_j)| < \epsilon \tag{30}$$

for both the GSLW method and our optimization method. The Haah method is terminated when the resulting factors are ϵ-close to the target polynomial of degree d for values on the d-th roots of unity. We set ϵ to be 10^{-12}. We highlight the critical feature that all of the arithmetic in our optimization algorithm is performed using only double-precision floating-point numbers. This is a remarkable advantage in terms of computation cost and numerical stability compared to the direct algorithms, which have to make use of variable precision arithmetic operations. In fact, our numerical results indicate that even with variable precision arithmetic operations, both the GSLW and the Haah method still struggle to find the phase factors accurately when the degree of polynomial becomes large ($\gtrsim 500$).

IV.1. Hamiltonian Simulation

A Hermitian matrix H with bounded norm $\|H\|_2 \leq 1$ has the spectral decomposition $H = \sum_j \lambda_j |j\rangle\langle j|$. The Hamiltonian simulation with duration τ through H is then given by $f(H) = e^{-\mathrm{i}\tau H} = \sum_j e^{-\mathrm{i}\tau\lambda_j} |j\rangle\langle j|$. Implementation of Hamiltonian simulation is thus determined by the phase factors that approximate the smooth complex-valued function $f(x) = e^{-\mathrm{i}\tau x}$. Since this is smooth on the interval $[-1, 1]$, its polynomial approximation can be generated from the Jacobi-Anger expansion[3]:

$$\begin{aligned} e^{-\mathrm{i}\tau x} = J_0(\tau) + 2 \sum_{k \text{ even}} (-1)^{k/2} J_k(\tau) T_k(x) \\ + 2\mathrm{i} \sum_{k \text{ odd}} (-1)^{(k-1)/2} J_k(\tau) T_k(x). \end{aligned} \tag{31}$$

[1] https://github.com/qsppack/QSPPACK

Here J_k's are the Bessel functions of the first kind. The L^∞ error to truncate the series up to order d is bounded by

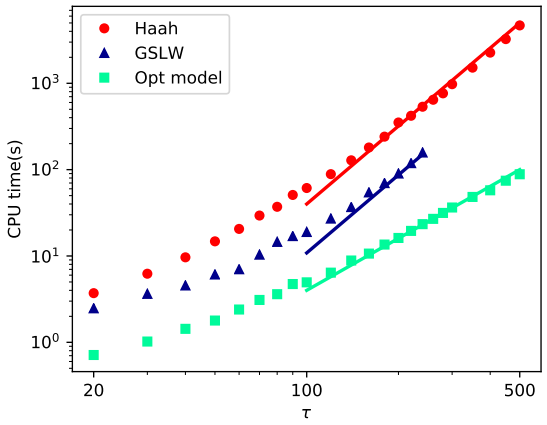
$$\begin{aligned} 2\sum_{k=d+1}^{\infty}|J_k(\tau)| &\le 2\sum_{k=d+1}^{\infty}\left(\frac{e|\tau|}{2}\right)^k k^{-k} \\ &\lesssim e^{-d}\sum_{k=d+1}^{\infty}\frac{1}{k!}\left(\frac{e|\tau|}{2}\right)^k < e^{e|\tau|/2-d}. \end{aligned} \tag{32}$$

Thus, the truncated series up to $d \approx e|\tau|/2 + \log(1/\epsilon_0)$ leads to an approximation whose truncation error is bounded by ϵ_0. In our simulation, we simply choose $d = 1.4|\tau| + \log(1/\epsilon_0)$, where $\epsilon_0 = 10^{-14}$, to make the truncation error negligible compared to the error caused by other factors. We denote such an approximation for Hamiltonian simulation with duration τ by f_τ.

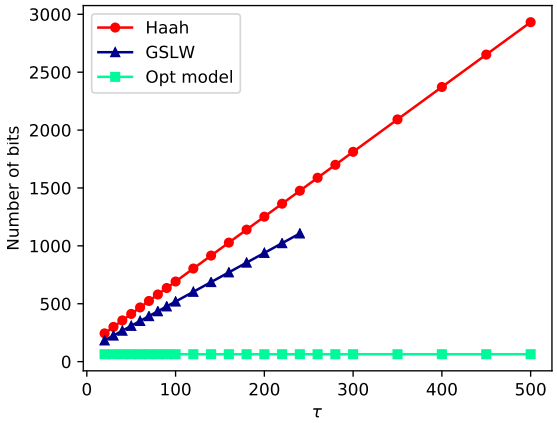
We compare our method with the GSLW and Haah methods on the polynomial given by Eq. (31). For each τ, we divide f_τ into real and imaginary parts, and perform algorithms separately according to case 3 in Section II.3. Then, we sum up the CPU time and the error together of each part as final results. We divide the coefficients of f_τ by a constant factor 2 to ensure $|f_\tau| \le 1$ for $x \in [-1, 1]$. The CPU time and the number of bits utilized to perform arithmetic are displayed in Fig. 5(a) and Fig. 5(b), respectively, together with polynomial fits to the data for large τ values in Fig. 5(a) (the points for small τ values are in the pre-asymptotic regime and are excluded in the fits).

We display results for τ up to 500 since the direct methods become very inefficient for larger values of τ. In particular, the GSLW method fails to yield phase factors with required accuracy $\epsilon = 10^{-12}$ when the degree d of f_τ is larger than 369. We contribute the failure to the instability of Julia's internal root-finding procedure. We observe that the CPU time of our proposed method scales as τ^2, while it scales as τ^3 for the Haah method. Moreover, for both the GSLW and the Haah method the number of bits required is linear in τ, while our optimization method is seen to be numerically stable in all calculations with use of only standard double precision arithmetic operations, *i.e.*, the number of bits is independent of τ.

To further demonstrate the capability of our method, we test our algorithm with τ up to 5000. When $\tau = 5000$, the polynomial degree d is 7033. The computational cost for evaluating the real and imaginary parts of f_τ is given in Fig. 6. We also display in Table I the L^∞ error (*i.e.* the maximum error) between the polynomial given by QSP phase factors and $e^{-i\tau x}$, to verify the robustness of our method and the effectiveness of our choice of the stopping criterion. The CPU time still scales asymptotically as τ^2, in agreement with our expectations since the per-iteration cost of the optimization procedure is $\mathcal{O}(d^2)$.



(a)



(b)

FIG. 5: Resource costs in determining QSP phase factors for the Hamiltonian simulation problem. Red dots, blue triangles and green squares correspond to the results by using Haah, GSLW and our optimization method, respectively. (a) CPU time(s) spent by each algorithm as a function of duration τ, together with polynomial fits in the large τ region. The degree of polynomial is $d = 1.4|\tau| + \log(1/\epsilon_0)$, with $\epsilon_0 = 10^{-14}$. The slope of the red (gray) and the blue (dark gray) lines is 3, representing CPU time = const $\cdot\, \tau^3$. The slope of the green (light gray) line is 2, representing CPU time = const $\cdot\, \tau^2$. (b) Number of bits used to store floating-point numbers and perform arithmetic. We show results for the GSLW method only up to $\tau = 240$ since it fails to generate accurate phase factors for larger τ.

τ	100	150	200	300	500	800
real	6.1e-13	7.9e-13	1.1e-12	2.4e-13	4.7e-13	3.6e-13
imaginary	1.1e-12	2.3e-13	3.3e-13	3.2e-13	2.8e-13	5.9e-13
τ	1000	1500	2000	3000	4000	5000
real	5.6e-13	5.5e-13	5.5e-13	7.2e-13	1.2e-12	9.4e-13
imaginary	4.2e-13	5.9e-13	9.0e-13	7.3e-13	9.0e-13	1.5e-12

TABLE I: L^∞ error of the optimization algorithm for determining QSP phase factors for Hamiltonian simulation as a function of τ. The degree of truncated polynomial is $d = 1.4|\tau| + \log(1/\epsilon_0)$ with $\epsilon_0 = 10^{-14}$.

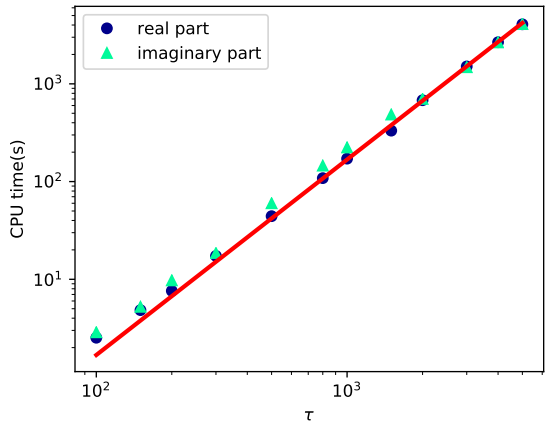


FIG. 6: CPU time(s) required using the optimization algorithm for determining QSP phase factors for Hamiltonian simulation, shown as a function of τ. Blue dots (green triangles) correspond to the real (imaginary) part of f_τ of degree $d = 1.4|\tau| + \log(1/\epsilon_0)$ with $\epsilon_0 = 10^{-14}$. The slope of the red line is 2, representing CPU time = const $\cdot\, \tau^2$.

IV.2. Eigenstate filtering function

In order to prepare an eigenstate corresponding to a known eigenvalue, we consider the following $2k$-degree polynomial

$$f_k(x, \Delta) = \frac{T_k(-1 + 2\frac{x^2-\Delta^2}{1-\Delta^2})}{T_k(-1 + 2\frac{-\Delta^2}{1-\Delta^2})}. \tag{33}$$

Suppose H is a Hermitian matrix with an eigenvalue λ that is separated from other eigenvalues by a gap $\Delta > 0$. Let $\tilde{H} = (H - \lambda I)/(\alpha + |\lambda|)$ and $\tilde{\Delta} = \frac{\Delta}{2\alpha}$. It was proven in [20] that

$$\|f_k(\tilde{H}, \tilde{\Delta}) - \hat{P}_\lambda\|_2 \leq 2e^{-\sqrt{2}k\tilde{\Delta}}, \tag{34}$$

where $\hat{P}_\lambda$ is the projection operator onto the eigenspace corresponding to λ. Furthermore, f_k, which is referred to as the eigenstate filtering function, is the optimal polynomial for filtering out the unwanted information from all other eigenstates.

For this demonstration we assume $\lambda = 0$, and $\alpha = 1$. We choose $\Delta = 0.1, 0.05, 0.01, 0.005$ and test our algorithm with different target filter values k. Eq. (34) indicates that $k\Delta$ controls the accuracy of the approximation. For each Δ we choose k such that $k\Delta = 3, 5, 10, 15, 20, 25$, respectively. The largest polynomial in this example is $d = 10,000$. The coefficients of polynomials are divided by $\sqrt{2}$ to avoid instabilities during optimization (see Section IV.5 for reasons to scale the function). The results are summarized in Fig. 7 and Table II. From the figure we observe that the optimization method performs stably in all cases, with CPU time scaling as k^2. These results are compared with the corresponding results for the direct methods of GSLW and Haah in Fig. 8, for Δ ranging from 0.005 to 0.1. This comparison is made only for $k\Delta = 3$, since we observe that direct methods struggle to treat larger values of $k\Delta$. It is evident that the optimization algorithm also shows superior performance to the direct methods in this example.

In particular, the Haah method fails to solve the QSP phase factors with required accuracy $\epsilon = 10^{-12}$ when Δ is less than 0.01. The weaker performance of the Haah method compared to (our modified) GSLW method observed in Fig. 8 can be attributed to the following reasons. We note that Julia's internal root-finding routine has difficulty finding all the roots of a polynomial when its degree is high, even when variable precision arithmetic operations are used. The performance of the GSLW and Haah methods can thus depend on the dataset, since they apply the root-finding procedure to different polynomials. We observe that sometimes the GSLW method can reach a polynomial of higher degree than the Haah method, and sometimes it is the other way around. We remark that the degree of polynomial fed into the Haah method is twice as large as that fed into the GSLW method, since the variable x is replaced by $(z + 1/z)/2$ in the Haah method. This increases the difficulty for the Haah method to solve phase factors successfully. By contrast, our modified implementation of the GSLW method (Appendix G) expands the polynomial in the Chebyshev basis, which significantly increases its stability, making its performance comparable to Haah's.

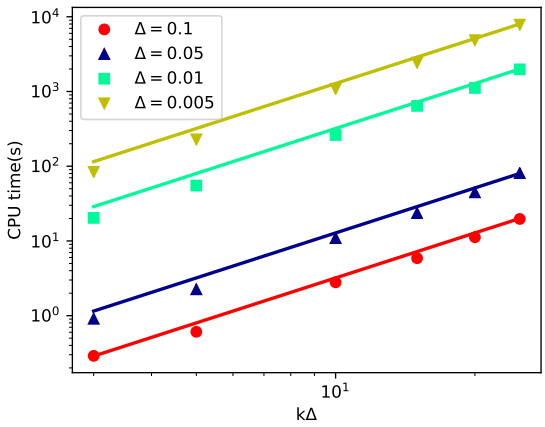


FIG. 7: CPU time(s) using the optimization algorithm for determining the set of phase factors for the eigenstate filter, shown as a function of $k\Delta$. The degree of each polynomial is $2k$. The slope of each line is 2, reflecting the quadratic cost CPU time = const $\cdot\, k^2$.

IV.3. Matrix inversion

Consider the quantum linear problem $A|x\rangle = |b\rangle$ where A is a Hermitian matrix whose condition number is κ. Then the eigenvalues of A are distributed within the interval $D_\kappa := [-1, -1/\kappa] \cup [1/\kappa, 1]$. The solution $|x\rangle$ can be constructed via matrix inversion, using QSP to gen-

Δ \ $k\Delta$	3	5	10	15	20	25
0.1	3.4e-14	5.2e-13	1.1e-13	1.1e-12	8.9e-14	8.5e-13
0.05	3.2e-14	4.9e-13	1.1e-13	1.1e-12	1.0e-13	8.4e-13
0.01	4.7e-14	4.9e-13	1.7e-13	1.1e-12	2.2e-13	8.1e-13
0.005	2.1e-13	5.6e-13	2.1e-13	1.2e-12	4.7e-13	8.8e-13

TABLE II: L^∞ error of the optimization algorithm for determining the QSP phases for the eigenstate filter defined in Eq. (33).

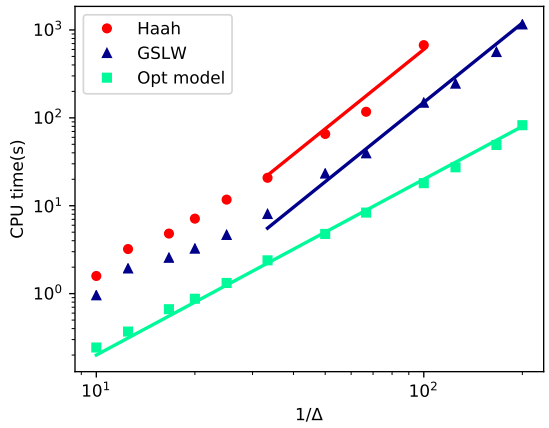


FIG. 8: CPU time(s) of the optimization algorithm (green squares) compared with direct (GSLW and Haah) algorithms (blue triangles and red dots, respectively) for determining QSP phases for the eigenstate filter, shown as a function of $1/\Delta$. The degree of each polynomial is $2k = 6/\Delta$. The slopes of the red (gray) and blue (dark gray) lines are 3, corresponding to a cubic cost in τ. The slope of the green (light gray) line is 2, corresponding to a quadratic cost in τ.

erate the action of A^{-1}. For this we need a polynomial approximation of $1/x$ on the interval D_κ. We consider two options here. The first is to generate a polynomial approximation of $1/x$ on D_κ by extending the function to the interval $[-1, 1]$ via an approximate function, as outlined in Section III.2 above. The second is to apply the Remez algorithm [6, 25] directly to the interval D_κ. The first approach was pursued in [7], where the following odd extension was proposed

$$g(x) := \frac{1-(1-x^2)^b}{x}. \tag{35}$$

Then, the truncated sum of Chebyshev polynomials

$$f(x) = 4\sum_{j=0}^{d}(-1)^j \frac{\sum_{i=j+1}^{b}\binom{2b}{b+i}}{2^{2b}} T_{2j+1}(x) \tag{36}$$

is ϵ_0-close to $1/x$ on D_κ by choosing $b = \left\lceil \kappa^2 \log(\frac{\kappa}{\epsilon_0}) \right\rceil$ and $d = \left\lceil \sqrt{b \log(\frac{4b}{\epsilon_0})} \right\rceil$. In the test made here ϵ_0 is set to be 10^{-14}.

In the second approach using the Remez algorithm, our goal for the matrix inversion problem is to directly construct an odd polynomial that approximates $f(x) = 1/x$ on D_κ. More generally, we note that if A is positive definite and $D_\kappa = [1/\kappa, 1]$, then we may approximate f by extending it to a function that is either even or odd. Since this paper focuses on the problem of finding the phase factors for approximating a smooth function in general, we will consider both the even and odd extensions below. For the current instance $f(x) = 1/x$, we gradually increase the degree d until the value of $f(x)$ obtained by the Remez algorithm approximates $1/x$ over D_κ with L^∞ error below ϵ_0. Fig.9 compares the polynomial given by the Fourier-Chebyshev method, Eq. (36), with that generated by the Remez method, for $\kappa = 20$ and $\epsilon_0 = 10^{-3}$.

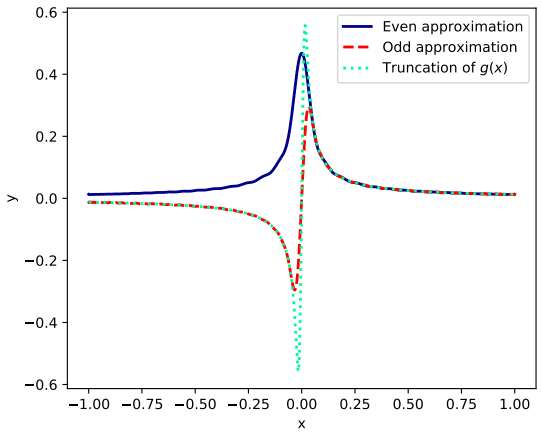


FIG. 9: Comparison of the form of polynomials given by the Fourier-Chebyshev method, Eq. (36), with those generated by the Remez method, for odd and even parities. The degree of the truncated polynomial here is 611 and degrees of the even (odd) approximation polynomials generated by the Remez method are 76 (111). The approximation polynomials are divided by 80 for this plot.

In this example we choose $\kappa = 10, 20, \ldots, 50$. We test our algorithm with $\epsilon_0 = 10^{-14}$ on polynomials given by Eq. (36) and generated by the Remez algorithm with odd and even parity, respectively. The CPU time associated with each polynomial approximation is presented in Fig. 11. We also compare the optimization method with the GSLW and the Haah method on the polynomials with lower degrees. We choose $\epsilon_0 = 10^{-6}$ and generate polynomials by the Remez algorithm with odd and even parity. The results of the comparison are demonstrated in Fig. 10, while the degrees of the polynomials given by each method are shown in Table III. Similar to the case of eigenstate filtering polynomials, we find that the GSLW and Haah methods cannot reach the target accuracy when the degree of the polynomial becomes large. Hence we reduce the accuracy in order to decrease the polynomial degrees here.

Table III indicates that use of the Remez method can significantly reduce the degree of polynomials needed to approximate $1/x$, with a reduction of to a factor of

$2 \sim 3$. We find that the even polynomial approximation is slightly less expensive than the odd expansion. This is due to the fact that an even extension has smaller gradient near the origin, compared with that of the odd extension, as shown in Fig. 9. Our proposed optimization method performs well on these examples, yielding phase factors robustly, with computational cost scaling quadratically with respect to κ. The largest polynomial degree $d = 4035$.

κ	10	20	30	40	50
Truncation of $g(x)$ ($\epsilon_0 = 10^{-14}$)	759	1559	2375	3201	4035
Odd Remez ($\epsilon_0 = 10^{-14}$)	303	607	911	1215	1519
Even Remez ($\epsilon_0 = 10^{-14}$)	280	560	840	1020	1400
Odd Remez ($\epsilon_0 = 10^{-6}$)	125	249	373	499	623
Even Remez ($\epsilon_0 = 10^{-6}$)	104	206	310	412	516

TABLE III: Degrees of approximation polynomials with accuracy ϵ_0 given by an odd smearing function in Eq. (36) and the Remez method with odd and even parity, respectively.

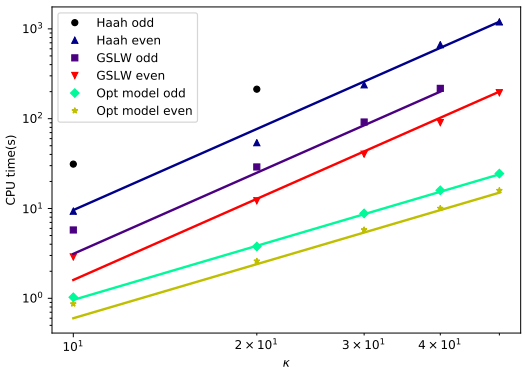


FIG. 10: CPU times for approximating $1/x$ over D_κ via QSP as a function of κ for the optimization method, compared with the corresponding times for the GSLW and Haah methods. Lines labelled "even" ("odd") represent the results of approximating polynomials given by the Remez method with even (odd) parity. The slopes of the two lowest lines are 2, corresponding to quadratic cost in κ, while the slopes of all other lines are 3, corresponding to cubic cost in κ. The line corresponding to the result by using Haah method to solve odd polynomials is not shown in the figure because only two data points are generated due to the numerical instability.

IV.4. Impact of the initial point

To demonstrate the complexity of the optimization landscape, we report the final value of the objective function starting from randomly generated points for the Hamiltonian simulation problem. For $\tau \in$

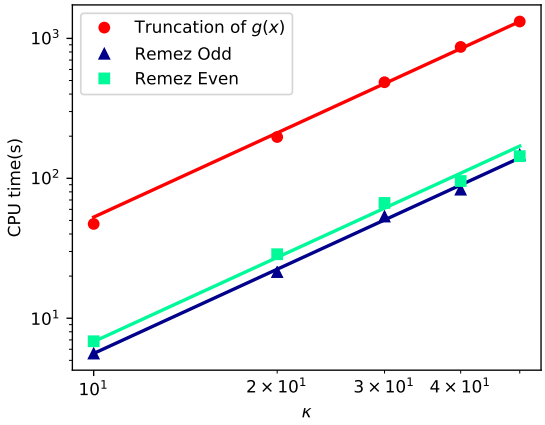


FIG. 11: Comparison of CPU time(s) of optimization algorithm for approximating $1/x$ over D_κ via QSP as a function of κ for the two different methods of finding the optimal polynomial. The red (gray) line represents the result of approximating the polynomial with the Fourier method, Eq. (36), the blue (dark gray) and the green (light gray) lines represent the CPU time of results of approximating the polynomial using the Remez method with odd and even parity, respectively. All lines have slope 2, corresponding to quadratic cost in κ.

$\{100, 200, 300, 400, 500\}$, we choose the target polynomial

$$f(x) = J_0(\tau)/2 + \sum_{\text{k even}}^{d} (-1)^{k/2} J_k(\tau) T_k(x) \qquad (37)$$

as an approximation to $\cos(\tau x)/2$. The initial points are uniformly distributed in $[-\pi, \pi)^{d+1}$. We run the L-BFGS algorithm until it converges or the number of iteration reaches 200. Fig. 12 summarizes the performance of the algorithm under random initialization. We see that most of the calculations get stuck in local minima with a relatively large objective value, confirming the complexity of the landscape. Furthermore, the difficulty of finding a good solution increases with the degree of the polynomial. By comparison, if we start from $\Phi = (\frac{\pi}{4}, 0, \ldots, 0, \frac{\pi}{4})$, the algorithm will converge within dozens of iterations to the global minimum with the objective function very close to 0.

IV.5. Sensitivity analysis

We further analyze the robustness of the method by reporting the condition number of the Hessian matrix $\text{Hess}\, L(\tilde{\Phi}^*)$ at the optimal point. The condition number of the Hessian matrix is an indicator reflecting the sensitivity of the optimizer with respect to small perturbations of the target function.

We compute here the Hessian condition number for the three optimization problems presented above in Sections IV.1 - IV.3. Interestingly, we observe that the condition

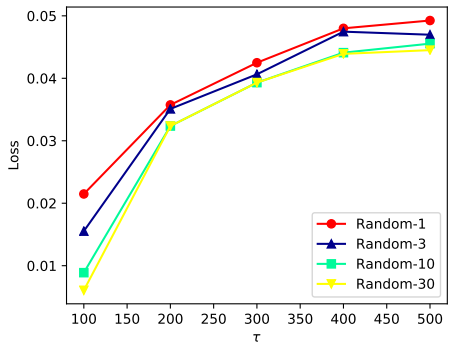


FIG. 12: Loss of optimization method initiated with randomly generated points. The target polynomial is defined as the truncated polynomial of Jacobi-Anger expansion of degree $d = 1.4|\tau| + \log(1/\epsilon_0)$ with $\epsilon_0 = 10^{-14}$. "Random-k" represents that we start from k different initial points and select best result.

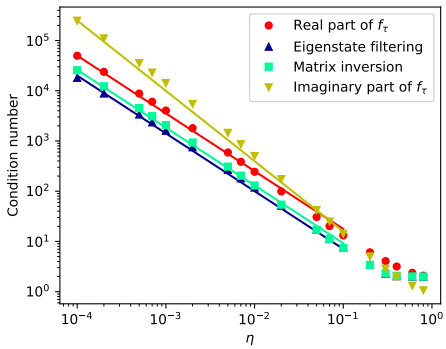


FIG. 13: Condition number of the Hessian matrix at the optimum of the objective function $L(\tilde{\Phi})$ defined in Eq. (23), shown for three different target polynomials studied in this work. (a) Real (Imaginary) part of truncated Jacobi-Anger expansion in Eq. (31), where $\tau = 200$ and $d = 312$ (represented by red dots and yellow downward triangles, respectively). (b) Eigenstate filter defined in Eq. (33) with $k = 300$, $\Delta = 0.05$ (represented by blue triangles). (c) Even polynomial approximation of $1/x$ on $D_{\kappa=20}$ generated by the Remez method (represented by green squares). Polynomials are scaled by a constant factor such that $\|f\|_\infty = 1 - \eta$, where η is the x-axis. The slope is 1.4 for the yellow (top) line and 1.15 for all others.

number is mostly affected by L^∞ norm of the target polynomial, rather than by its degree or by its parameters. Thus, each problem can be exemplified by one polynomial with a given degree and parameters. To investigate how the norm affects Hessian condition number, we scale the L^∞ norm of the given polynomial to $1 - \eta$. Fig. 13 shows the scaled Hessian condition numbers as a function of η. As $\eta \to 0^+$, we find that the condition number increases as $\eta^{-\gamma}$ with $\gamma > 1$ in all three cases. This indicates that when $\|f\|_\infty$ is close to 1, the optimizer can be very sensitive to perturbations in f. When $\|f\|_\infty$ is below 1, the enhanced stability implies that these phase factors can be used as an initial guess for a slightly perturbed target polynomial, which will be discussed in detail in Section V. Furthermore, scaling the target polynomial f to ensure that $\|f\|_\infty \leq 1 - \eta$ for some given threshold η is also preferable. Such scaling of the target polynomial was also suggested in the root-finding procedures of the direct algorithms in order to ensure numerical stability [13].

V. DECAY OF PHASE FACTORS FROM THE CENTER AND PHASE FACTOR PADDING

In addition to the symmetry structure discussed in Section III.1, for smooth target functions, we observe that the QSP phase factors decay rapidly away from the center. To illustrate the decay and also the symmetry, we plot several examples in Fig. 14. After subtracting the $\pi/4$ factor on both ends of the phase factors, we observe that the decay of the phase factors closely follows the decay of the Chebyshev coefficients (defined only on the positive axis in Fig. 14).

Theorem 5 states that for phase factors with relatively small magnitudes, the optimal phase factors can be expressed approximately analytically in terms of the coefficients of the Chebyshev polynomial expansion. The proof is given in Appendix H.

Theorem 5. Let $\Phi \in \mathcal{C}_{d'}$ be a set of symmetric QSP phase factors. Define $\widetilde{\phi}_j := \phi_j - \frac{\pi}{4}(\delta_{j,0} + \delta_{j,d'-1})$ and $\widetilde{\Phi} := (\widetilde{\phi}_0, \cdots, \widetilde{\phi}_{d'-1})$. Define a polynomial

$$g_\Phi(x) := -\left(\prod_{j=0}^{d'-1} \cos\widetilde{\phi}_j\right) \times \begin{cases} \sum_{j=0}^{d} 2\tan\left(\widetilde{\phi}_j\right) T_{2d+1-2j}(x) & , d' = 2d+2 \\ \tan\left(\widetilde{\phi}_d\right) + \sum_{j=0}^{d-1} 2\tan\left(\widetilde{\phi}_j\right) T_{2d-2j}(x) & , d' = 2d+1. \end{cases} \tag{38}$$

Then for sufficiently small $\left\|\widetilde{\Phi}\right\|_1$, there exists a constant $C > 0$ such that the desired QSP component $f_\Phi(x) := \mathrm{Re}\left[\langle 0|U_\Phi(x)|0\rangle\right]$ satisfies

$$\|f_\Phi(x) - g_\Phi(x)\|_\infty \leq C\left\|\widetilde{\Phi}\right\|_1^3. \tag{39}$$

According to Theorem 5, one can directly deduce approximate values of the phase factors from the coefficients of the Chebyshev expansion. For example, when d' is even, $\widetilde{\phi}_j \approx -\arctan(c_{2d+1-2j}/2) \approx -c_{2d+1-2j}/2$ holds up to $\mathcal{O}(\|\widetilde{\Phi}\|_1^3)$. For smooth functions, the Chebyshev coefficients decay at least super-algebraically (*i.e.*, faster than any polynomial decay) [4]. So the phase factors also

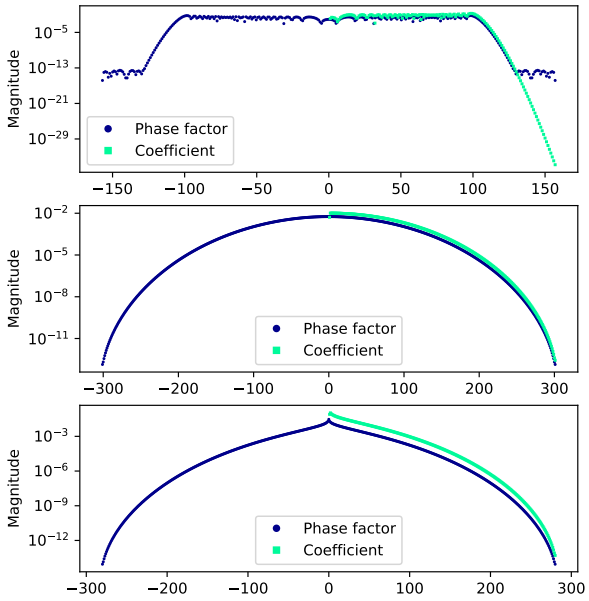


FIG. 14: Magnitude of coefficients of the polynomial f in the Chebyshev basis (light gray) and the corresponding phase factors (after subtracting $\pi/4$ on both ends, dark gray), for the three different problems studied in this work. We shift the x-axis to more clearly illustrate the symmetry property of the phase factors. Coefficients that are zero due to parity are omitted. (a) Real part of truncation of the Jacobi-Anger expansion in Eq. (31), with $\tau = 200$ and $d = 312$. (b) Eigenstate filter defined in Eq. (33), with $k = 300$, $\Delta = 0.05$. (c) Even polynomial approximation of $1/x$ on $D_{\kappa=20}$ as generated by the Remez method.

decay super-algebraically away from the center. The uniformly small phase factors can be realized by rescaling the function f to f/β, with β being a large number. We remark that our numerical results in Fig. 14 do not rely on such a scaling factor. A more precise characterization of the decay of the phase factors will be a focus of future work.

One possible usage of the decay property of the phase factors is as follows, which we refer to as a "phase padding" procedure. Suppose we have solved the QSP phase factors corresponding to a polynomial approximation f_1 of relatively low degree to a real-valued function f with definite parity. In order to improve the accuracy of the approximation, another small term f_2 of higher polynomial degree is needed to be added to approximate f together with f_1. Therefore, a natural question is whether we can reuse the phase factors associated with f_1 to generate that of $f_1 + f_2 \approx f$.

To solve this problem, one needs to increase the dimension of Φ, since the degree of the polynomial has been increased and hence also the number of phase factors. Due to the symmetry structure, we may consider the following symmetrically padded phase factors and further show that the symmetrical padding operation preserves the desired part of the QSP.

Definition 6 (**l-padded phase factors**). Let $\Phi = (\phi_0, \cdots, \phi_d) \in \mathcal{C}_{d+1}$ be symmetric QSP phase factors. Then, the corresponding l-padded phase factors in $\mathcal{C}_{d+2l+1}$ are given by $\Phi_l := (\frac{\pi}{4}, \underbrace{0, \cdots, 0}_{l-1}, \phi_0 - \frac{\pi}{4}, \phi_1, \cdots, \phi_{d-1}, \phi_d - \frac{\pi}{4}, \underbrace{0, \cdots, 0}_{l-1}, \frac{\pi}{4})$.

Theorem 7. Given a set of symmetric phase factors Φ and a nonnegative integer l, its l-padded phase factors preserve the real part of the upper-left component of the QSP unitary matrix, *i.e.*, $\mathrm{Re}\left[\langle 0|U_\Phi(x)|0\rangle\right] = \mathrm{Re}\left[\langle 0|U_{\Phi_l}(x)|0\rangle\right], \forall x \in [-1, 1]$.

Proof. Using Lemma 3, it is equivalent to prove the equality

$$\mathrm{Im}\left[\langle 0|U_\Phi(x)|0\rangle\right] = \mathrm{Im}\left[\langle 0|W(x)^l U_\Phi(x) W(x)^l|0\rangle\right]$$

for symmetric phase factors Φ. Insert the resolution of identity,

$$\begin{aligned}\text{r.h.s.} =& \mathrm{Im}\left[T_l(x)^2 P(x) - 2(1-x^2) R_{l-1}(x) T_l(x) Q(x)\right. \\ & \left. -(1-x^2) R_{l-1}(x)^2 P^*(x)\right] \\ =& (T_l^2(x) + (1-x^2) R_{l-1}^2(x)) \mathrm{Im}[P(x)] \\ =& \mathrm{Im}[P(x)] = \text{l.h.s.}\end{aligned}$$

Here we have used $Q \in \mathbb{R}[x]$ according to Theorem 2. □

To demonstrate the usage of this phase padding procedure, we consider the approximation of $\cos(\tau x)/2$, namely, the real part of Eq. (31) scaled by a constant factor 2. First, an integer d_0 is chosen such that the truncated series up to d_0 is a rough approximation of $\cos(\tau x)/2$. Meanwhile, the corresponding phase factors are solved by optimization. Then we gradually increase the size of the problem by an even number l, *i.e.*, adding $l/2$ more terms of higher order polynomials. In order to reuse the phase factors, the initial guess in step k is lifted from the phase factors solved in the previous step, *i.e.*, the polynomial approximation of degree $d_0 + (k-1)l$. The procedure is repeated until the degree meets a maximal criterion d_1, which generates an accurate polynomial approximation of $\cos(\tau x)/2$.

The parameters in numerical implementations are set to be $\tau = 500$, $d_0 = 500$, $l = 10$, $d_1 = 600$. The L^∞ error before the optimization (*i.e.*, only using phase factor padding) and after the optimization in each step is shown in Fig. 15, while Table IV compares the computational cost between optimizations initiated with and without padding. We observe that the polynomial given by the lifted phase factors is already close to the target polynomial. This means that the lifted phase factors provide a good initial guess close to the global minimum.

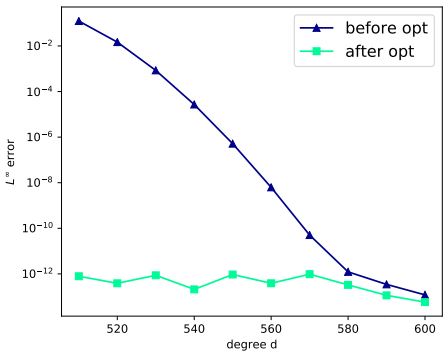


FIG. 15: L^∞ error between the polynomial obtained from the lifted phase factors, and the target polynomial, as a function of the degree d of the latter. Blue triangles represent the error before optimization, and green squares represent the error after optimization. The target polynomial here is the truncated polynomial of Jacobi-Anger expansion of degree d.

d	510	520	530	540	550
with padding	19.9	19.7	16.9	16.4	12.2
without padding	21.8	21.2	22.5	22.6	23.5
d	560	570	580	590	600
with padding	9.37	4.69	3.18	3.17	3.19
without padding	24.2	26.1	28.5	28.3	27.7

TABLE IV: CPU times for optimizations initiated with and without phase padding (see text). The target polynomial here is the truncated polynomial of Jacobi-Anger expansion of degree d.

VI. DISCUSSION

We have demonstrated that using an optimization based approach, we can efficiently and accurately evaluate the phase factors needed to build QSP circuits for generation of unitary representations of non-unitary operations. Taken together with the QSP formalism of Refs. [11, 21], this approach now provides efficient and accurate constructive procedures to implement QSP and thereby removes a crucial bottleneck for the application of QSP in quantum algorithms. We expect that our method will be useful for a wide range of matrix functions of interest to quantum algorithms, including the broad classes of Hamiltonian simulation, generation of thermal states, and linear algebra problems. The optimization approach was found to be superior to previous direct methods that rely on a reduction procedure in which numerical errors are accumulated and amplified. Instead of employing a reduction procedure, our approach is based on optimization of a distance function that quantifies the difference between the target polynomial and the QSP representation of this, with the QSP phases as variable parameters. We identified two key features for success of the optimization based method: first, the choice of the initial guess, and second, preservation of the symmetry structure of the phase factors. We found that a simple choice of the initial guess can be surprisingly effective, despite the complexity of the global landscape of the objective function. This indicates that a better understanding of the local energy landscape connecting the initial guess to the optimal phase factors is needed. Our study also reveals the connection between two seemingly unrelated objects in the QSP construction, namely, the decay of phase factors from the center, and the decay of the Chebyshev coefficients of the target function. More precise characterization of this connection will be a useful future research direction, together with further work to understand the energy landscape of the objective function.

Acknowledgment This work was partially supported by a Google Quantum Research Award (Y.D.,L.L.,B.W.), by the Quantum Algorithm Teams Program under Grant No. DE-AC02-05CH11231 (L.L. and B.W.), and by Department of Energy under Grant No. DE-SC0017867 (L.L.). X.M. thanks the Office of international relations, Peking University, Beijing, China for partial funding of an exchange studentship at the University of California, Berkeley. We thank Robert Kosut, Nathan Wiebe, and Yu Tong for discussion. Y.D. and X.M. contributed equally to this work.

[1] A. Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. In STACS'12 (29th Symposium on Theoretical Aspects of Computer Science), volume 14, pages 636–647, 2012.

[2] P. Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. Journal of statistical physics, 22(5):563–591, 1980.

[3] D. W. Berry, A. M. Childs, and R. Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. Proceedings of the 56th IEEE Symposium on Foundations of Computer Science, pages 792–809, 2015.

[4] J. P. Boyd. Chebyshev and Fourier spectral methods. Courier Corporation, 2001.

[5] R. Chao, D. Ding, A. Gilyen, C. Huang, and M. Szegedy. Finding angles for quantum signal processing with machine precision. arXiv preprint arXiv:2003.02831, 2020.

[6] E. W. Cheney. Introduction to approximation theory. McGraw-Hill, 1966.

[7] A. M. Childs, R. Kothari, and R. D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. SIAM J. Comput., 46:1920–1950, 2017.

[8] A. M. Childs, D. Maslov, Y. Nam, N. J. Ross, and Y. Su. Toward the first quantum simulation with quantum speedup. Proc. Nat. Acad. Sci., 115:9456–9461, 2018.

[9] E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028, 2014.

[10] R. P. Feynman. Simulating physics with computers. Int. J. Theor. Phys, 21(6/7), 1982.

[11] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. 2018.

[12] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, pages 193–204, 2019.

[13] J. Haah. Product decomposition of periodic functions in quantum signal processing. Quantum, 3:190, 2019.

[14] J. Haah, M. Hastings, R. Kothari, and G. H. Low. Quantum algorithm for simulating real time evolution of lattice hamiltonians. In 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, oct 2018. doi:10.1109/focs.2018.00041.

[15] A. Haar. Die minkowskische geometrie und die annäherung an stetige funktionen. Mathematische Annalen, 78(1):294–311, 1917.

[16] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. Phys. Rev. Lett., 103:150502, 2009.

[17] N. Higham. Functions of matrices: theory and computation, volume 104. SIAM, 2008.

[18] N. J. Higham. Accuracy and stability of numerical algorithms, volume 80. Siam, 2002.

[19] C. Jordan. Essai sur la géométrie à n dimensions. Bulletin de la Société mathématique de France, 3:103–174, 1875.

[20] L. Lin and Y. Tong. Solving quantum linear system problem with near-optimal complexity. arXiv:1910.14596, 2019.

[21] G. H. Low and I. L. Chuang. Optimal hamiltonian simulation by quantum signal processing. Phys. Rev. Lett., 118:010501, 2017.

[22] G. H. Low and I. L. Chuang. Hamiltonian simulation by qubitization. Quantum, 3:163, 2019.

[23] G. H. Low, T. J. Yoder, and I. L. Chuang. Methodology of resonant equiangular composite quantum gates. Phys. Rev. X, 6:041067, 2016.

[24] V. Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. Computers & Mathematics with Applications, 31(12):97–138, 1996.

[25] E. Remez. Sur le calcul effectif des polynomes d'approximation de tchebichef. CR Acad. Sci. Paris, 199: 337–340, 1934.

[26] W. Sun and Y.-X. Yuan. Optimization theory and methods: nonlinear programming, volume 1. Springer Science & Business Media, 2006.

Appendix A: Uniqueness of phase factors

We refer to the representation in Eq. (13) as appeared in [12, Theorem 3] as GSLW's representation. There is another equivalent form proposed in [13], which we call it Haah's representation. Under Haah's representation, the QSP unitary is

$$U_{\hat{\Phi}}(x) = e^{\mathrm{i}\sigma_z\hat{\phi}_0}\prod_{j=1}^{d}\left[e^{\mathrm{i}\sigma_z\hat{\phi}_j/2}W(x)e^{-\mathrm{i}\sigma_z\hat{\phi}_j/2}\right] = e^{\mathrm{i}\sigma_z(\hat{\phi}_0+\hat{\phi}_1/2)}\left(\prod_{j=1}^{d-1}W(x)e^{\mathrm{i}\sigma_z(\hat{\phi}_{j+1}-\hat{\phi}_j)/2}\right)W(x)e^{-\mathrm{i}\sigma_z\hat{\phi}_d/2} \tag{A1}$$

where $\hat{\phi}_{d+1} := 0$. Compared to Eq. (13), the transformation between two representations is evident, *i.e.*, $\mathcal{T} : [-\pi,\pi)^{d+1} \to \mathcal{C}_{d+1}$, $\hat{\Phi} \mapsto \Phi$ such that $\phi_0 = \hat{\phi}_0 + \frac{\hat{\phi}_1}{2}$, $\phi_j = \frac{\hat{\phi}_{j+1}-\hat{\phi}_j}{2}$, $\forall j = 1,\cdots,d-1$ and $\phi_d = -\hat{\phi}_d/2$. The irreducible set $\mathcal{C}_{d+1}$ is defined as the image of this linear transformation. The uniqueness of Haah's phase factors in $[-\pi,\pi)^{d+1}$ was proved in [13, Theorem 2], which considers a formally more general class of polynomial functions $\mathrm{U}(1) \to \mathrm{SU}(2)$. The bijection $\mathcal{T}$ implies the uniqueness of GSLW's phase factors in $\mathcal{C}_{d+1}$. It is evident that the 2π-periodicity of Haah phase factors lead to a pair of $\pm\pi$ shifts in the corresponding GSLW phase factors. Then, if we define the equivalence relation $\Phi \sim \Psi$ when $\phi_k = \psi_k, \forall k \neq i,j$ and $\phi_i = \psi_i + \pi, \phi_j = \psi_j - \pi$, the irreducible set is the quotient space $\mathcal{C}_{d+1} \equiv [-\pi,\pi)^{d+1}/\sim$.

Appendix B: Reducing one ancilla qubit for representation of real polynomials

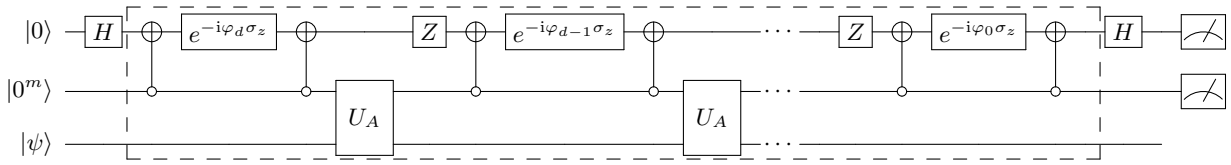


FIG. 16: Quantum circuit for quantum signal processing of real matrix polynomials with a Hermitian block-encoding U_A.

We explain here why the additional ancilla qubit needed for representing real polynomials in Section II.3, case 1 as a result of the linear combination of two QSP circuits, is in fact not needed and can be avoided. Specifically, this ancilla qubit can be combined with the first ancilla qubit in Fig. 2. To see why this is the case, note that the phase factors for $U_{-\Phi}$ in Eq. (18) can be obtained by taking the phase factors for U_Φ in Eq. (15), and perform the mapping $\varphi_i \mapsto -\varphi_i + \pi(1-\delta_{id}), i = 0,\ldots,d$. In other words, we negate φ_i and add π to all but the d-th entry. Negating the phase can be implemented by feeding $|1\rangle$ instead of $|0\rangle$ to the signal state, and adding π to the phase can be implemented via a σ_z gate associated with $\phi_0,\ldots,\phi_{d-1}$.

We may verify that by slightly modifying Fig. 2, the circuit in the box with dashed line in Fig. 16 in fact implements

$$|0\rangle\langle 0| \otimes U_\Phi + |1\rangle\langle 1| \otimes U_{-\Phi},$$

which is the select oracle, Eq. (6). Therefore, using the Hadamard gate as the prepare oracle (Eq. (5)) as before, the circuit Fig. 16 provides a $(1, m+1, 0)$-block-encoding of $f(A/\alpha)$, which saves one ancilla qubit.

Appendix C: Quantum signal processing with a non-Hermitian block-encoding matrix

Let A be an n-qubit Hermitian matrix, but its $(\alpha, m, 0)$-block-encoding U_A is not Hermitian. We can still perform QSP by introducing an additional ancilla qubit. To this end, we first generate an $(\alpha, m+1, 0)$-block-encoding of A that is Hermitian. Define an $(m+n+1)$-qubit controlled block-encoding as

$$V_A' := |0\rangle\langle 0| \otimes U_A + |1\rangle\langle 1| \otimes U_A^\dagger, \tag{C1}$$

which uses both U_A and $U_A^\dagger$. We also introduce the swap operation $S := \sigma_x \otimes I_m$. Then

$$U_A' := (S \otimes I_n)V_A' = |1\rangle\langle 0| \otimes U_A + |0\rangle\langle 1| \otimes U_A^\dagger \tag{C2}$$

is Hermitian. Define an $(m+1)$-qubit signal state for block-encoding

$$|G\rangle := |+\rangle\,|0^m\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0^m\rangle, \tag{C3}$$

then

$$\begin{aligned}(\langle G|\otimes I_n)U'_A(|G\rangle\otimes I_n) =&(\langle G|\otimes I_n)V'_A(|G\rangle\otimes I_n)\\ =&\frac{1}{2}(\langle 0^m|\otimes I_n)U_A(|0^m\rangle\otimes I_n)+\frac{1}{2}(\langle 0^m|\otimes I_n)U_A^\dagger(|0^m\rangle\otimes I_n)\\ =&\frac{1}{2}A+\frac{1}{2}A^\dagger = A.\end{aligned}$$

In the last equality, we used that A is a Hermitian matrix. This proves that U'_A is indeed an $(\alpha, m+1, 0)$-block-encoding of A. Define

$$U'_\Pi = (2|G\rangle\langle G| - I_{m+1})\otimes I_n, \tag{C4}$$

we may use Jordan's lemma to simultaneously block-diagonalize U'_Π, U'_A. In particular, the matrix representation in Eq. (9) still holds, which provides the qubitization of A.

Then QSP representation in Eqs. (11) and (12) can be directly obtained by substituting $U_\Pi \to U'_\Pi, U_A \to U'_A$. The circuit is given in Fig. 17. In the second line, the Hadamard gate converts the $|+\rangle$ state in the signal state into $|0\rangle$ and back in order to apply the $(m+2)$-qubit Toffoli gate. The swap operation can be implemented via a single σ_x gate. The last Hadamard gate in the second line is not present, in order to measure in the $|\pm\rangle$ basis set according to the signal state $|G\rangle$.

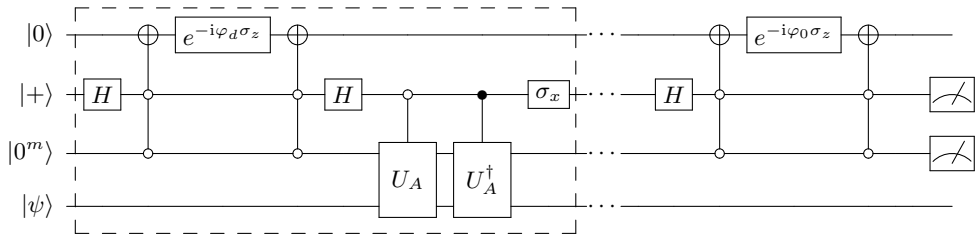


FIG. 17: Quantum circuit for quantum signal processing with a non-Hermitian block-encoding matrix. The circuit in the box enclosed by the dashed line should be repeated d times, each time with a different phase factor. The last Hadamard gate in the second line is removed if measurements are to be made in the $|\pm\rangle$ basis set.

Appendix D: Proof of Theorem 4

We first review some basic facts of the Chebyshev polynomial. The Chebyshev polynomials are two sequences of polynomials which can be defined by trigonometric functions. For each $d \in \mathbb{N}$ and $x \in [-1,1]$, the Chebyshev polynomial of the first kind is defined as $T_d(x) = \cos(d\arccos(x))$ and that of the second kind is $R_d(x) = \sin((d+1)\arccos(x))/\sin(\arccos(x))$. Both T_d and R_d are polynomials of degree d. We will focus on the properties of Chebyshev polynomials of the first kind in the following context and call T_d's Chebyshev polynomials for simplicity. Define the weighted inner product as $(f,g)_w := \int_{-1}^{1} f(x)g(x)\frac{\mathrm{d}x}{\sqrt{1-x^2}}$ on the space $L^2_w([-1,1])$. Then Chebyshev polynomials are orthogonal polynomials on $[-1,1]$ with respect to the inner product $(\cdot,\cdot)_w$, and form a complete basis on the space $L^2_w([-1,1])$.

Lemma 8. Any function $g \in L^2_w([-1,1])$ can be uniquely expressed as a series of Chebyshev polynomials,

$$g(x) = \sum_{j\in\mathbb{N}} c_j T_j(x), \quad \text{where } c_j = \frac{2-\delta_{j0}}{\pi}(g, T_j)_w.$$

By substituting $x \to \cos\theta$, the series in terms of Chebyshev polynomial becomes the Fourier series of periodic function $g(\cos\theta)$. The roots of Chebyshev polynomials are called Chebyshev nodes, *e.g.*, $\left\{\cos\left(\frac{2j-1}{2d}\pi\right) : j = 1, \cdots, d\right\}$

are Chebyshev nodes of T_d. Chebyshev polynomials satisfy the discrete orthogonality

$$\sum_{j=1}^{d} T_m(x_j)T_n(x_j) = d\frac{1+\delta_{m,0}}{2}\delta_{m,n} \tag{D1}$$

where $d > \lfloor (m+n)/2 \rfloor$ is an integer and x_j's are Chebyshev nodes of T_d.

Proof. (Theorem 4) Let $\tilde{d} = \lceil \frac{d+1}{2} \rceil$, then $2\tilde{d} > d$. Apply the Cauchy–Schwarz inequality, we have

$$\sum_{j=1}^{\tilde{d}} |f(x_j) - f_\Phi(x_j)| \leq \sqrt{\tilde{d}\sum_{j=1}^{\tilde{d}} |f(x_j) - f_\Phi(x_j)|^2} = \sqrt{\tilde{d}^2 L(\phi)} \leq \tilde{d}\sqrt{\epsilon}, \tag{D2}$$

where $x_j = \cos\left(\frac{(2j-1)\pi}{4\tilde{d}}\right)$, $j = 1, \ldots, \tilde{d}$ are positive roots of Chebyshev polynomial $T_{2\tilde{d}}(x)$. For a fixed integer $t \leq d$,

$$\sum_{j=1}^{\tilde{d}} (f(x_j) - f_\Phi(x_j))T_t(x_j) = \sum_{j=1}^{\tilde{d}}\sum_{m=0}^{d} (\alpha_m - \beta_m)T_m(x_j)T_t(x_j) = \sum_{m=0}^{d} (\alpha_m - \beta_m)\sum_{j=1}^{\tilde{d}} T_m(x_j)T_t(x_j) = \sum_{m=0}^{d} (\alpha_m - \beta_m)\eta_{mt}, \tag{D3}$$

where by discrete orthogonality in Eq. (D1) and symmetry, $\eta_{mt} = \tilde{d}\frac{1+\delta_{m,0}}{2}\delta_{m,t}$. Thus we have

$$|\alpha_m - \beta_m| \leq \frac{2}{\tilde{d}}\sum_{j=1}^{\tilde{d}} |(f(x_j) - f_\Phi(x_j))T_m(x_j)| \leq 2\sqrt{\epsilon} \tag{D4}$$

for any $m = 0, \ldots, d$. □

Appendix E: Remez Method

We would like to solve for the best approximation polynomial in terms of the L^∞ norm

$$f^* = \underset{f\in\mathbb{R}[x], \deg(f)\leq d}{\operatorname{argmin}} \max_{x\in[a,b]} |F(x) - f(x)|. \tag{E1}$$

In addition, the approximation problem encountered in this work requires that the approximation polynomial has a definite parity. Hence, we need to focus on the best approximation problem, over the linear combination of a general basis of functions $\{g_1(x), \ldots, g_N(x)\}$ other than $\{1, x, \ldots, x^d\}$. In this paper we choose $N = \lceil \frac{d+1}{2} \rceil$, where d is the degree of the approximation polynomial we would like to generate. A series of functions $\{g_1(x), \ldots, g_N(x)\}$ is said to satisfy the Haar condition on a set X, if each $g_j(x)$ is continuous and for every N points $x_1, \ldots, x_N \in X$, the N vectors $v_j := (g_1(x_j), \ldots, g_N(x_j))$, $1 \leq j \leq N$ are linearly independent [15]. As an example, the Haar condition holds if we choose $g_j(x) = T_{2j-1}(x)$ (or $T_{2j-2}(x)$) and $X \subset (0, 1]$. Solution of the best approximation problem over such a basis will yield the best odd (even) approximation polynomial. Imposing the Haar condition simplifies the solution of the generalized approximation problem.

The optimal approximate polynomial f^* over the linear combination of functions $\{g_1(x), \ldots, g_N(x)\}$ can be found via the Remez exchange method summarized in Algorithm 3, which computes a series of approximation polynomials on discrete sets. The polynomials f_t generated by the Remez algorithm converge uniformly to the optimal polynomial f^* with linear convergence rate. For a large range of functions F, the convergence rate can be improved to be quadratic. We refer the reader to [6, Chapter 3] for more details related to the Remez method.

Appendix F: L-BFGS Algorithm

In numerical optimization, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is a quasi-Newton method for solving unconstrained optimization problems [26, Chapter 5]. The BFGS method stores a dense $n \times n$ matrix to approximate the inverse of Hessian matrix. It updates this approximation by performing a rank two update using gradient information along its trajectory. Limited-memory BFGS (L-BFGS) approximates the BFGS method by using a limited amount of computer memory [26, Chapter 5]. In particular, it represents the inverse of Hessian matrix implicitly by only a few vectors. For completeness, we summarize the procedure for the L-BFGS method in Algorithm 4.

ALGORITHM 3: Remez method for solving the best approximation polynomial

Input: An interval $[a,b] \subset \mathbb{R}$, target function F, a basis $\{g_1,\ldots,g_N\}$ satisfying the Haar condition, $N+1$ initial points $a \le x_0 \le \cdots \le x_N \le b$.

Set $t=0$.
while stopping criterion is not satisfied **do**
Set $t=t+1$.
Solve the linear equation for $a_1,\ldots,a_N$ and Δ

$$\sum_{j=1}^{N} a_j g_j(x_k) - F(x_k) = (-1)^k \Delta,\ k=0,\ldots,N.$$

Denote $f_t(x)=\sum_{j=1}^{N} a_j g_j(x)$ and residual $r(x)=F(x)-f_t(x)$.
$r(x)$ has a root $z_j \in (x_{j-1},x_j)$ for $j=1,\ldots,N$. Set $z_0=a$ and $z_{N+1}=b$.
Let $\sigma_j=\mathrm{sgn}(r(x_j))$. Find $y_j=\mathrm{argmax}_{y\in[z_j,z_{j+1}]}\,\sigma_j r(y)$ for each $j=0,\ldots,N$.
if $\|r(x)\|_\infty > \max_j |r(y_j)|$ **then**
Choose

$$y=\operatorname*{argmax}_{y\in[a,b]} |r(y)|.$$

Replace a $y_k \in \{y_0,\ldots,y_N\}$ by y in such a way that the values of $r(y)$ on the resulting ordered set still satisfies

$$r(y_j)r(y_{j+1})<0,\ j=0,\ldots,N-1.$$

end if
Replace $\{x_0,\ldots,x_N\}$ by $\{y_0,\ldots,y_N\}$.
end while
Output: an approximation to the best approximation polynomial $f_t(x)$

Appendix G: Implementation details of the direct methods for finding phase factors

For completeness we provide here our implementation of the direct methods for computing phase factors, *i.e.*, the GSLW method and the Haah method. The codes are written in Julia v1.2.0. Although advanced root-finding algorithm with guaranteed performance [24] is suggested in the Haah method [13], this is a theoretical result and hard to implement. We utilize instead the function roots in the PolynomialRoots package in Julia to find the roots of polynomials. For both GSLW and Haah methods, we perform calculations with variable precision arithmetic (VPA) using the BigFloat data type. The numbers of bits R used in our numerical tests are empirical parameters whose values are chosen to minimize CPU time while maintaining accuracy. We first take R to be a large number and then gradually decrease it, until the algorithm fails to yield phase factors with sufficient accuracy. The algorithm is considered as a failure on an example if it cannot generate accurate enough phase factors, *i.e.*, within the specified tolerance, despite the arithmetic being performed under increasingly high precision. Specifically, we choose $R=3d$ for the GSLW method and $R=4d$ for the Haah method in the Hamiltonian simulation, $R=2d$ for both methods for the eigenstate filtering function, and $R=50\kappa$ for both methods in the matrix inversion problem. Here d is the degree of the polynomial. Note that the polynomials encountered in the matrix inversion subsection approximate $1/x$ on $D_\kappa=[1/\kappa,1]$.

Our implementation of the GSLW algorithm proposed in [12] is summarized in Algorithm 5. To avoid stability issues caused by inaccurate roots, a root s is regarded as a real (pure imaginary) number if the magnitude of its imaginary (real) part is smaller than machine precision ($\epsilon=10^{-16}$ in our implementation). Similarly, s is rounded to 1 if $|1-s|<\epsilon$. We evaluate the coefficients of $B(x)$ and $C(x)$ with respect to the Chebyshev basis by discrete fast Fourier transform (FFT) to enhance numerical stability. The reduction procedure in the loop is also performed based in the Chebyshev basis. We observe that compared to the original implementation of the GSLW method in [11], the use of the Chebyshev basis significantly improves the numerical stability of the algorithm. Since in the examples in this work we primarily consider situations where only P is required, we employ a zero polynomial as the input for the second polynomial $\tilde{Q}$.

ALGORITHM 4: **Function:** $\phi = \text{L-BFGS}(\phi^0, L, m, B^0)$

Input: Initial point ϕ^0, objective function $L(\phi)$, a nonnegative integer m and initial approximation of inverse Hessian B^0.

Set $t = 0$
while stopping criteria does not meet **do**
  Compute $g_t = \nabla L(\phi^t)$, set $q = g_t$
  for $i = t-1, \ldots, t-m$ **do**
    Set $\alpha_i = \rho_i s_i^\top q$
    $q = q - \alpha_i y_i$
  end for
  $r = B_0 q$
  for $i = t-m, \ldots, t-1$ **do**
    $\beta = \rho_i y_i^\top r$
    $r = r + s_i(\alpha_i - \beta)$
  end for
  Set search direction $d_t = -r$.
  Find a step size γ_t using backtracking line search.
  Set

$$\phi^{t+1} = \phi^t + \gamma_t d_t, s_k = \phi^{t+1} - \phi^t,$$
$$y_t = g_{t+1} - g_t, \rho_t = \frac{1}{s_t^\top y_t}.$$

  Set $t = t + 1$.
end while
Return: ϕ^t

ALGORITHM 5: GSLW method

Input: A nonnegative integer d, real polynomials $\tilde{P}$ and $\tilde{Q}$ satisfying condition (1) – (2) of Theorem 1 and $\tilde{P}^2(x) + (1-x^2)\tilde{Q}^2(x) \leq 1, \forall x \in [-1,1]$. A nonnegative integer R indicates the number of bits on which high-precision arithmetic is performed.

Step 1: Find the complementary polynomials

Solve all roots of $1 - P^2(x) - (1-x^2)Q^2(x)$. Denote S as the multiset that contains roots of $1 - P^2(x) - (1-x^2)Q^2(x)$ with their algebraic multiplicity. Find the following subsets of S

$$S_0 = \{s \in S | s = 0\}, \qquad S_{(0,1)} = \{s \in S | s \in (0,1)\},$$
$$S_{[1,\infty)} = \{s \in S | s \in [1,\infty)\}, \qquad S_I = \{s \in S | \mathrm{Re}(s) = 0, \mathrm{Im}(s) > 0\},$$
$$S_C = \{s \in S | \mathrm{Re}(s) > 0, \mathrm{Im}(s) > 0\}.$$

Define

$$\begin{aligned} Z(x) = K x^{|S_0|/2} \prod_{s \in S_{(0,1)}} \sqrt{x^2 - s^2} \prod_{s \in S_{[1,\infty)}} (\sqrt{s^2-1}x + \mathrm{i}s\sqrt{1-x^2}) \\ \prod_{s \in S_I} (\sqrt{|s|^2+1}x + \mathrm{i}|s|\sqrt{1-x^2}) \prod_{(a+b\mathrm{i}) \in S_C} (cx^2 - (a^2+b^2) + \mathrm{i}\sqrt{c^2-1}x\sqrt{1-x^2}), \end{aligned} \tag{G1}$$

where K is the absolute value of the coefficient of the highest order of polynomial $1 - P^2(x) - (1-x^2)Q^2(x)$, $c = a^2 + b^2 + \sqrt{2(a^2+1)b^2 + (a^2-1)^2 + b^4}$.
$Z(x)$ can be written in the form $Z(x) = B(x) + \mathrm{i}\sqrt{1-x^2}C(x)$ for $B, C \in \mathbb{R}[x]$. B and C are required complementing polynomials if B has same parity as $\tilde{P}$ while C has opposite parity, otherwise we replace $Z(x)$ by $Z(x)(x + \mathrm{i}\sqrt{1-x^2})$.
Calculate coefficients of B and C and define $P(x) := \tilde{P}(x) + \mathrm{i}B(x)$, $Q(x) := \tilde{Q}(x) + \mathrm{i}C(x)$. Then $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1, \forall x \in [-1,1]$.

Step 2: Matrix reduction

Set $t = d$.
while $\deg(P) > 0$ **do**
Denote coefficients of highest order of P and Q as p_t and q_{t-1}, respectively. We have $|p_t| = |q_{t-1}|$. Choose $\phi_t \in \mathbb{R}$ such that $e^{2\mathrm{i}\phi_t} = p_t/q_{t-1}$.
Replace P and Q by

$$P_{\mathrm{new}}(x) = e^{-\mathrm{i}\phi_t}\left(xP(x) + \frac{p_t}{q_{t-1}}(1-x^2)Q(x)\right) \tag{G2}$$

and

$$Q_{\mathrm{new}}(x) = e^{-\mathrm{i}\phi_t}\left(\frac{p_t}{q_{t-1}}xQ(x) - P(x)\right). \tag{G3}$$

Set $t = t - 1$.
end while
Choose $\phi_0 \in \mathbb{R}$ such that $e^{\mathrm{i}\phi_0} = P(1)$. Set $\phi_j = \frac{\pi}{2}$ for $j = 1, 3, \ldots, t-1$, $\phi_{j'} = -\frac{\pi}{2}$ for $j' = 2, 4, \ldots, t$.
Output: QSP phase factors $\Phi = (\phi_0, \ldots, \phi_d)$ satisfying

$$U_\Phi(x) = e^{\mathrm{i}\phi_0\sigma_z}\prod_{j=1}^{d}\left[W(x)e^{\mathrm{i}\phi_j\sigma_z}\right] = \begin{pmatrix} \tilde{P}(x) + \mathrm{i}B(x) & (\mathrm{i}\tilde{Q}(x) - C(x))\sqrt{1-x^2} \\ (\mathrm{i}\tilde{Q}(x) + C(x))\sqrt{1-x^2} & \tilde{P}(x) - \mathrm{i}B(x) \end{pmatrix} \tag{G4}$$

The Haah method proposed in [13] is summarized in Algorithm 6. Here a Laurent polynomial of degree d represents polynomials having the form $P(z) = \sum_{j=-d}^{d} p_j z^j$, $p_j \in \mathbb{C}, |p_d| + |p_{-d}| \neq 0$. A complex-valued function P is said to be real-on-circle if $P(z) \in \mathbb{R}$, $\forall |z| = 1$.

Suppose two real polynomials $\tilde{P}(x)$ and $\tilde{Q}(x)$ satisfy the requirements of Algorithm 5, they can be converted to desired input of Algorithm 6 through the formula

$$A(z) = \tilde{P}\left(\frac{z+z^{-1}}{2}\right), \quad B(z) = \frac{z-z^{-1}}{2\mathrm{i}}\tilde{Q}\left(\frac{z+z^{-1}}{2}\right). \tag{G5}$$

If $A(z)$ and $B(z)$ are generated by this formula, we may only compute $d+1$ terms $E_0, E_1(t), \ldots, E_d(t)$ from coefficients $C_{2k}^{2d}, k = -d, -d+2, \ldots, d$ such that

$$A(z) + \mathrm{i}B(z) \approx \langle + | E_0 E_1(z) \cdots E_d(z) | + \rangle, \quad \forall z \in U(1). \tag{G6}$$

[13] proved that in this case matrix P_j computed in the algorithm are of form

$$P_j = e^{\mathrm{i}\sigma_z \hat{\phi}_j/2} |+\rangle \langle +| e^{-\mathrm{i}\sigma_z \hat{\phi}_j/2}, \; j = 1, \ldots, 2d, \tag{G7}$$

and there exists $\hat{\phi}_0$ such that $E_0 = e^{\mathrm{i}\sigma_z \hat{\phi}_0}$. The transformation formula between $\hat{\Phi} = (\hat{\phi}_0, \ldots, \hat{\phi}_d)$ and QSP phase factors Φ are given in Appendix A. In practice we take $B(z) = 0$ since we are not interested in the second polynomial $\tilde{Q}$. As the rational approximation procedure in Step 1 is designed to bound the error theoretically and hard to implement, in practice we round the coefficients of $(1 - \epsilon/3)A(z)$ and $(1 - \epsilon/3)B(z)$ with small magnitude to zero instead of taking rational approximation.

ALGORITHM 6: Haah method

Input: A real parameter $\epsilon \in (0, 0.1)$, a nonnegative integer R indicates the number of bits on which high-precision arithmetic is performed and a complex-valued Laurent polynomial $A(e^{\mathrm{i}\theta}) + \mathrm{i}B(e^{\mathrm{i}\theta}) = \sum_{k=-d}^{d} \zeta_k e^{\mathrm{i}k\theta}$ such that
(1) A and B are real-on-circle polynomials,
(2) $|A(e^{\mathrm{i}\theta})|^2 + |B(e^{\mathrm{i}\theta})|^2 \leq 1$, $\forall \theta \in \mathbb{R}$,
(3)$A(e^{\mathrm{i}\theta})$ and $B(e^{\mathrm{i}\theta})$ have definite parity as a function of θ.

Step 1: Denote $d = \deg(A)$. Taking rational approximations of each coefficient of $(1-\epsilon/3)A(z)$ and $(1-\epsilon/3)B(z)$ up to error $\frac{\epsilon}{30d}$. Coefficients with magnitude smaller than $\frac{\epsilon}{30d}$ should be rounded to zero. Parity properties of A and B should be kept during rounding procedure. Denote resulting rational real-on-circle polynomials as $a(z)$ and $b(z)$, respectively. Coefficients of a and b should be store as rational numbers. Denote $n = \deg(a)$ and $n' = \deg(1 - a(z)^2 - b(z)^2)$.
Step 2: Find all roots of $1 - a(z)^2 - b(z)^2$. Denote S as the multiset that contains roots of $1 - a(z)^2 - b(z)^2$ with their algebraic multiplicity.
Step 3: Define $e(z) = z^{-\lfloor \frac{n'}{2} \rfloor} \prod_{\substack{s \in S \\ |s|<1}} (z - s)$ and constant $\alpha = \frac{1-a(z)^2-b(z)^2}{e(z)e(1/z)}$. Define complementary polynomials $c(z)$ and $d(z)$ as

$$c(z) = \sqrt{\alpha}\frac{e(z) - e(1/z)}{2\mathrm{i}}, \quad d(z) = \sqrt{\alpha}\frac{e(z) + e(1/z)}{2}. \tag{G8}$$

Evaluate $c(z)$ and $d(z)$ on $D = 2^{\lceil \log_2(2n+1) \rceil}$ points

$$\{e^{2\pi \mathrm{i}k/D} | k = 0, \ldots, D-1\} \tag{G9}$$

by computing $e(z)$ and $e(1/z)$ via factorized form rather than direct expansion.
Step 4: Compute 2-by-2 complex matrices $C_{2k}^{2n}, -n \leq k \leq n$ such that

$$\sum_{k=-n}^{n} C_{2k}^{2n} z^k = a(z)I + b(z)\mathrm{i}\sigma_x + c(z)\mathrm{i}\sigma_y + d(z)\mathrm{i}\sigma_z$$

via discrete fast Fourier transform.
Step 5:
for $m = 2n, 2n-1, \ldots, 1$ **do**
Compute

$$P_m = \frac{(C_m^m)^\dagger C_m^m}{\mathrm{Tr}((C_m^m)^\dagger C_m^m)}, \; Q_m = \frac{(C_{-m}^m)^\dagger C_{-m}^m}{\mathrm{Tr}((C_{-m}^m)^\dagger C_{-m}^m)}. \tag{G10}$$

Define $E_m(z) = zP_m + z^{-1}(I - P_m)$. Compute coefficients

$$C_k^{m-1} = C_{k-1}^m Q_m + C_{k+1}^m P_m, \; k = -m+1, -m+3, \ldots, m-3, m-1. \tag{G11}$$

end for
Define $E_0 = C_0^0$.
Output: $E_0, E_1(z), \ldots, E_{2n}(z)$ satisfying

$$|A(z^2) + \mathrm{i}B(z^2) - \langle + | E_0 E_1(z) \cdots E_{2n}(z) | + \rangle| \leq \epsilon, \quad \forall z \in U(1). \tag{G12}$$

Appendix H: Proof of Theorem 5

First consider $d' = 2d + 2$. According to Lemma 3, it is equivalent to prove

$$\left\| \mathrm{Im}\left[\langle 0|U_{\widetilde{\Phi}}(x)|0\rangle\right] + \prod_{j=0}^{2d+1} \cos\left(\widetilde{\phi}_j\right) \cdot \sum_{j=0}^{d} \left(-2\tan\left(\widetilde{\phi}_j\right)\right) T_{2d+1-2j}(x) \right\|_\infty \leq \frac{1}{6}\left\|\widetilde{\Phi}\right\|_1^3 + \mathcal{O}\left(\left\|\widetilde{\Phi}\right\|_1^5\right) \tag{H1}$$

For simplicity, we drop the tilde in phase factors. Divide the QSP phase factors into two groups symmetrically, $\Phi_l = (\phi_0, \cdots, \phi_d)$, $\Phi_r = \Phi_l^-$, Then, $U_\Phi(x)$ can be expressed in terms of the product of two QSP matrices,

$$U_\Phi(x) = U_{\Phi_l}(x) W(x) U_{\Phi_r}(x) = e^{\mathrm{i}\phi_0\sigma_z} \prod_{j=1}^{d} \left[W(x) e^{\mathrm{i}\phi_j\sigma_z}\right] W(x) \prod_{j=0}^{d-1} \left[e^{\mathrm{i}\phi_{d-j}\sigma_z} W(x)\right] e^{\mathrm{i}\phi_0\sigma_z}. \tag{H2}$$

Each QSP unitary can be equivalently written as

$$U_{\Phi_l}(x) = \left[\prod_{j=0}^{d}\cos(\phi_j)\right](1+\mathrm{i}t_0\sigma_z)\prod_{j=1}^{d}\left[W(x)\left(1+\mathrm{i}t_j\sigma_z\right)\right], \tag{H3}$$

where $t_j := \tan(\phi_j) \sim \mathcal{O}(\phi_j)$. Then, the contributions up to $\mathcal{O}(\|\Phi\|_1^4)$ come from selecting up to three σ_z's in the expansion,

$$\begin{aligned}\frac{U_{\Phi_l}(x)}{\prod_{j=0}^{d}\cos(\phi_j)} =& W(x)^d + \sum_{j=0}^{d}\mathrm{i}t_j\sigma_z W(x)^{d-2j} - \sum_{j_1<j_2} t_{j_1}t_{j_2}W(x)^{d-2(j_2-j_1)} \\ &- \sum_{j_1<j_2<j_3}\mathrm{i}t_{j_1}t_{j_2}t_{j_3}\sigma_z W(x)^{d-2(j_1+j_3-j_2)} + \mathcal{O}(\|\Phi\|_1^4).\end{aligned} \tag{H4}$$

Here we have used the following relation repeatedly

$$W(x)\sigma_z = \sigma_z W(x)^{-1}.$$

After taking imaginary part of the upper-left component in Eq. (H2), it is evident that only odd orders in ϕ_j's have nonvanishing contributions according to Eq. (H4). Furthermore, using that $U_{\Phi_r} = U_{\Phi_l}^\top$, we have

$$\begin{aligned}\frac{\mathrm{Im}\left[\langle 0|U_\Phi(x)|0\rangle\right]}{\prod_{j=0}^{2d+1}\cos(\phi_j)} =& \sum_{j=0}^{d} 2t_j T_{2d+1-2j}(x) - \sum_{j=0}^{d}\sum_{j_1<j_2} 2t_j t_{j_1}t_{j_2}T_{2d+1-2(j+j_2-j_1)}(x) \\ &- \sum_{j_1<j_2<j_3} 2t_{j_1}t_{j_2}t_{j_3}T_{2d+1-2(j_1+j_3-j_2)}(x) + \mathcal{O}(\|\Phi\|_1^5).\end{aligned} \tag{H5}$$

Let $s_j := \sin(\phi_j)$. It implies the expected bound,

$$\begin{aligned}\left\|\mathrm{Im}\left[\langle 0|U_\Phi(x)|0\rangle\right] - \prod_{j=0}^{2d+1}\cos\left(\phi_j\right)\cdot\sum_{j=0}^{d}2\tan\left(\phi_j\right)T_{2d+1-2j}(x)\right\|_\infty &\le 2\left(\frac{1}{2}+\frac{1}{6}\right)\sum_{j_1,j_2,j_3=0}^{d}|s_{j_1}s_{j_2}s_{j_3}| + \mathcal{O}(\|\Phi\|_1^5) \\ &\le \frac{4}{3}\left(\|\Phi\|_1/2\right)^3 + \mathcal{O}\left(\|\Phi\|_1^5\right) = \frac{1}{6}\|\Phi\|_1^3 + \mathcal{O}\left(\|\Phi\|_1^5\right).\end{aligned} \tag{H6}$$

This proves Eq. (39) for even d'.

Then prove the case $d' = 2d+1$. The QSP unitary is again divided symmetrically and we drop the tilde in phase factors for simplicity. Define $\Phi_l = (\phi_0, \cdots, \phi_{d-1}),\ \Phi_r = \Phi_l^-$

$$U_\Phi(x) = U_{\Phi_l}(x)W(x)e^{\mathrm{i}\phi_d\sigma_z}W(x)U_{\Phi_r}(x) = \underbrace{\cos(\phi_d)U_{\Phi_l}(x)W(x)^2U_{\Phi_r}(x)}_{①} + \underbrace{\mathrm{i}\sin(\phi_d)U_{\Phi_l}(x)W(x)\sigma_z W(x)U_{\Phi_r}(x)}_{②} \tag{H7}$$

Similar to expansion in Eq. (H4), we conclude the following bounds,

$$\begin{aligned}&\left\|\mathrm{Im}\left[\langle 0|①|0\rangle\right] - \prod_{j=0}^{2d}\cos(\phi_j)\cdot\sum_{j=0}^{d-1}2t_jT_{2d-2j}(x)\right\|_\infty \le 2\left(\frac{1}{2}+\frac{1}{6}\right)\left(\sum_{j=0}^{d-1}|s_j|\right)^3 + \mathcal{O}\left(\|\Phi\|_1^5\right) \le \frac{1}{6}\|\Phi\|_1^3 + \mathcal{O}\left(\|\Phi\|_1^5\right), \\ &\left\|\mathrm{Im}\left[\langle 0|②|0\rangle\right] - \prod_{j=0}^{2d}\cos(\phi_j)\cdot\tan(\phi_d)\right\|_\infty \le \frac{s_d}{4}\|\Phi\|_1^2 + \frac{s_d}{48}\|\Phi\|_1^4 + \mathcal{O}(\|\Phi\|_1^6) \le \frac{1}{4}\|\Phi\|_1^3 + \mathcal{O}\left(\|\Phi\|_1^5\right).\end{aligned} \tag{H8}$$

Using the triangle inequality, we prove Eq. (39) when d' is odd.