Stable factorization for phase factors of quantum signal processing

Lexing Ying

Department of Mathematics, Stanford University, Stanford, CA 94305, USA

This paper proposes a new factorization algorithm for computing the phase factors of quantum signal processing. The proposed algorithm avoids root finding of high degree polynomials by using a key step of Prony's method and is numerically stable in the double precision arithmetics. Experimental results are reported for Hamiltonian simulation, eigenstate filtering, matrix inversion, and Fermi-Dirac operator.

1 Introduction

1.1 Background

This paper is concerned with the problem of quantum signal processing. Quantum computing has been mostly working with unitary operators, since the quantum gates and circuits are unitary. However, in recent years, we have witnessed great progress in representing non-unitary operators efficiently with quantum circuits.

Let A be an $N \times N$ Hermitian matrix with $N = 2^n$ and $\|A\|_2 < 1$ (after scaling if needed). For simplicity, we only consider Hermitian matrices in this paper and refer the readers to [6, 4] for more general cases. One of the most successful methods for presenting A on a quantum circuit is the Hermitian block encoding

$$A \leftrightarrow \begin{bmatrix} A & * \\ * & * \end{bmatrix} \equiv U_A,$$

where U_A is a Hermitian unitary matrix of size $(2^m \cdot N) \times (2^m \cdot N)$, A is the top-left corner of U_A, and U_A can be implemented using a quantum circuit with $n + m$ input qubits.

In most of the quantum problems in scientific computing, such as Hamiltonian simulation, filtering, and quantum linear algebra [3, 10, 4, 13], one is often interested the Hermitian matrix $f(A)$ of A, where $f(x)$ is a real function defined on $[-1, 1]$ with $\|f\|_\infty < 1$. The block encoding scheme requires $f(A)$ to be represented as the top-left block of a larger unitary matrix $U_{f(A)}$ implemented by a quantum circuit

$$f(A) \leftrightarrow \begin{bmatrix} f(A) & * \\ * & * \end{bmatrix} \equiv U_{f(A)}.$$

A key question is whether there is an algorithm that builds the quantum circuit $U_{f(A)}$ from the circuit U_A by using only the knowledge of the function $f(x)$ but treating U_A as a

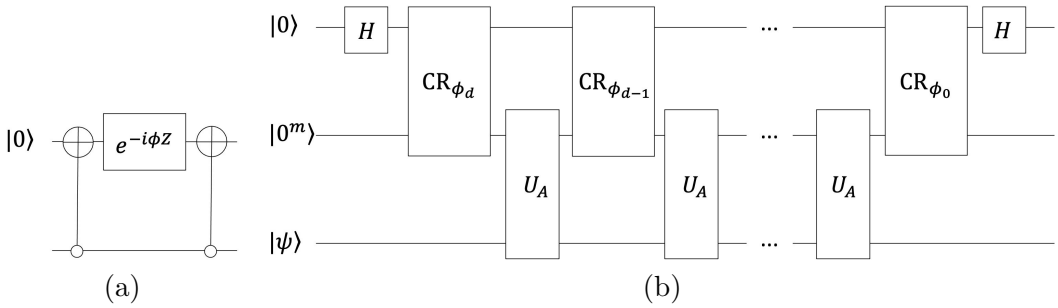

Figure 1: (a) a controlled rotation circuit CR_ϕ with angle ϕ. (b) quantum eigenvalue transformation. Here H is the Hadamard gate, U_A is the block encoding of a Hermitian A, and CR_{ϕ_j} is with the control angle ϕ_j. The whole circuit implements $U_{a(A)}$ for a real polynomial $a(x)$.

black box

$$U_A \equiv \begin{bmatrix} A & * \\ * & * \end{bmatrix} \Rightarrow U_{f(A)} \equiv \begin{bmatrix} f(A) & * \\ * & * \end{bmatrix}.$$

This question is answered by the quantum eigenvalue transformation described in [9, 6]. To simplify the discussion, we assume that all Hermitian matrices mentioned below satisfy $\|A\|_2 < 1$ and all functions defined on $[-1, 1]$ satisfy $\|f\|_\infty < 1$. The quantum eigenvalue transform proceeds as follows (see [8] for example for details).

- Split the polynomial $f(x)$ into the even and odd parts $f^{\mathrm{e}}(x)$ and $f^{\mathrm{o}}(x)$ on $x \in [-1, 1]$
- Approximate the even part $f^{\mathrm{e}}(x)$ with an even degree polynomial $a^{\mathrm{e}}(x)$ and implement $a_{\mathrm{e}}(A)$ with a circuit shown in Figure 1(b) with appropriate phase factors $\phi_0^{\mathrm{e}}, \ldots, \phi_{d_{\mathrm{e}}}^{\mathrm{e}}$. Here d_{e} is the equal to degree of $a^{\mathrm{e}}(x)$.
- Approximate the odd part $f^{\mathrm{o}}(x)$ with an odd degree polynomial $a^{\mathrm{o}}(x)$ and implement $a_{\mathrm{o}}(A)$ with a circuit shown in Figure 1(b) with appropriate phase factors $\phi_0^{\mathrm{o}}, \ldots, \phi_{d_{\mathrm{o}}}^{\mathrm{o}}$. Here d_{o} is the equal to degree of $a^{\mathrm{o}}(x)$.
- Combine the circuits implementing each component together by linear combination of unitaries (LCU) [2].

The key remaining step is how to construct the phase factors $\phi_0, \ldots, \phi_d$ for an even or odd polynomial $a(x)$ of degree d. This is answered by the quantum signal processing theorem [9, 6, 10]: Given a polynomial $a(x) \in \mathbb{R}[x]$ on $[-1, 1]$ of degree d, parity $d \mod 2$, and $\|a\|_\infty = \max_{x\in[-1,1]} |a(x)| < 1$, there exists a sequence of phase factors $\Phi = (\phi_0, \ldots, \phi_d) \in [-\pi, \pi]^{d+1}$ such that $a(x) = \mathrm{Re}(p(x))$, where $p(x)$ is defined via

$$U(x, \Phi) = \begin{pmatrix} p(x) & r(x) \\ r^*(x) & p^*(x) \end{pmatrix} = e^{i\phi_0 Z} e^{i \arccos(x) X} e^{i\phi_1 Z} e^{i \arccos(x) X} \cdots e^{i\phi_{d-1} Z} e^{i \arccos(x) i X} e^{i\phi_d Z}, \tag{1}$$

where

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

are the Pauli matrices.

In this paper, we use the following notational convention

$$a(x) = \mathrm{Re}(p(x)), \quad c(x) = \mathrm{Im}(p(x)), \quad b(x) = \mathrm{Re}(r(x)), \quad d(x) = \mathrm{Im}(r(x)). \tag{2}$$

It is often convenient to work with variable $t \in [-\pi, \pi]$ and lift these functions to the t space via the transform $x = \cos(t)$, i.e., given $f(x)$ for $x \in [-1, 1]$, define

$$f(t) := f(x = \cos(t))$$

for $t \in [0, \pi]$ and extend to $t \in [-\pi, 0]$ analytically. For example $f(x) = x$ lifts to $f(t) = \cos(t)$ and $f(x) = \sqrt{1 - x^2}$ to $f(t) = \sin(t)$. In the t variable, (1) can be written more compactly as

$$U(t, \Phi) = \begin{pmatrix} p(t) & r(t) \\ r^*(t) & p^*(t) \end{pmatrix} = e^{i\phi_0 Z} e^{itX} e^{i\phi_1 Z} e^{itX} \cdots e^{i\phi_{d-1} Z} e^{itX} e^{i\phi_d Z}. \tag{3}$$

We can also use complex variable $z = e^{it}$ and lift $f(t)$ analytically to a Laurent polynomial $f(z)$ with

$$f(z = e^{it}) = f(t)$$

on the unit circle. For example $f(t) = \cos(t)$ lifts to $f(z) = \frac{z+z^{-1}}{2}$ and $f(t) = \sin(t)$ to $f(z) = \frac{z-z^{-1}}{2i}$. In the following discussion, we often work with the lifted functions $a(z)$, $b(z)$, $c(z)$, and $d(z)$ over the complex plane.

1.2 Previous work

There are two main approaches for computing the phase factors. The first one [6, 7, 1] is based on polynomial factorization. Following the notation of [7], this approach starts by choosing a function $b(\cdot)$ that has the right parity and satisfies $a^2(t) + b^2(t) < 1$. Let $\{\xi_j\}$ be the set of $2d$ roots of the Laurent polynomial $1 - a^2(z) - b^2(z)$ inside the unit circle. Define

$$e(z) = z^{-d} \prod_{|\xi_j|<1} (z - \xi_j).$$

By setting $\alpha \equiv \frac{1-a^2(z)-b^2(z)}{e(z)e(1/z)}$, the functions $c(z)$ and $d(z)$ are then equal to [7]

$$c(z) = \left(\sqrt{\alpha} \cdot \frac{e(z) + e(1/z)}{2}\right), \quad d(z) = \left(\sqrt{\alpha} \cdot \frac{e(z) - e(1/z)}{2i}\right). \tag{4}$$

With $a(z)$, $b(z)$, $c(z)$, and $d(z)$ available, $p(z) = a(z) + ic(z)$ and $r(z) = b(z) + id(z)$ as defined in (2). Given $p(z)$ and $r(z)$, the algorithm for extracting the phase factors $\Phi = (\phi_0, \ldots, \phi_d)$ is quite straightforward (see for example Theorem 3 of [5]). For completeness, it is also included in Section 3.1 following our notation.

Though this approach is direct, the implementation requires finding roots of a high degree Laurent polynomial, which is often unstable in double precision arithmetics. It was shown that [7] that $O(d \log d/\epsilon)$ classical bits are needed and the algorithms are often implemented with variable precision. In [1], an algorithm based on the halving and capitalization techniques is proposed to mitigate the numerical issue and it was able to scales to more than 3000 phase factors.

The second approach is based on optimization [4], i.e., minimizing directly

$$\min_{\Phi} \int_{[-1,1]} |\mathrm{Re}(U(x,\Phi)_{11}) - a(x)|^2 \mathrm{d}x,$$

but with the search space restricted to the symmetric phase factors Φ (i.e. $\phi_j = \phi_{d-j}$). Though this minimization problem is highly non-convex, [4] demonstrates numerically that, starting from the initial guess $\Phi^0 = (\pi/4, 0, \ldots, 0, \pi/4)$, a quasi-Newton method is able to find the *maximal solution* that corresponds to $b(t) = 0$ in our notation. The numerical results in [4] demonstrated robust computation of the phase factors up to 10000 phase factors. A recent study [14] proves that for $\|a\|_\infty \leq O(1/d)$ a projected gradient method converges to the maximal solution.

1.3 Contribution

The main contribution of this paper is a new stable algorithm of the factorization approach. It is based on two observations. First, the factorization approach does not really need the roots $\{\xi_j\}$ since the function $e(z) = z^{-d}\prod_{|\xi_j|<1}(z-\xi_j)$ only depends on the characteristic polynomial $\prod_{|\xi_j|<1}(z-\xi_j)$ of these roots. We show that this characteristic polynomial can be computed directly via a key component of Prony's method [12, 11], without knowing the roots $\{\xi_j\}$. This avoids the root-finding, which is the main source of instability of the factorization approach. Second, in order to compute the characteristic polynomial in a robust way, we propose to pick $b(z)$ randomly with a dominant highest frequency, i.e., in some sense opposite to the symmetric phase factors. This allows us to compute the characteristic polynomial using the standard numerical linear algebra routines.

The resulting algorithm is conceptually simple and easy to implement. On the numerical side, compared with the-state-of-the-art results in [4], our algorithm achieves comparable accuracy ($\sim 10^{-12}$) and has the same $O(d^2)$ computational cost. The longest sequence reported in our experiments scales to over 50000 phase factors.

The rest of the paper is organized as follows. Section 2 reviews the Prony's method. Section 3 describes the main algorithms. The numerical results are given in Section 4.

2 Review of Prony's method

Let us explain Prony's method with a simple but key example. Let $(f_k)_{k\in\mathbb{Z}}$ be a sequence of the form

$$f(k) = \sum_{j=1}^{d} e^{i\omega_j k} r_j,$$

where d is the number of terms, $\{\omega_j\}$ are the frequencies, and $\{r_j\}$ are the weights. Assume that d, $\{\omega_j\}$, and $\{r_j\}$ are all unknown to us. The computation problem is to recover d, $\{\omega_j\}$ (up to 2π), and $\{r_j\}$ from potentially noisy values of $(f_k)_{k\in\mathbb{Z}}$.

Prony's method starts by considering the infinite vector $[e^{i\omega_j k}]_{k\in\mathbb{Z}}$ for some j and the upward shift operator S. Applying S to this vector gives

$$S\begin{bmatrix}\vdots\\ e^{i\omega_j k}\\ \vdots\end{bmatrix} = \begin{bmatrix}\vdots\\ e^{i\omega_j (k+1)}\\ \vdots\end{bmatrix} \quad \text{i.e.} \quad (S - e^{i\omega_j})\begin{bmatrix}\vdots\\ e^{i\omega_j k}\\ \vdots\end{bmatrix} = 0.$$

Taking the product over all $(S - e^{i\omega_j})$ leads to

$$\prod_{s=1}^{d}(S - e^{i\omega_s}) \begin{bmatrix} \vdots \\ e^{i\omega_j k} \\ \vdots \end{bmatrix} = 0.$$

Taking a linear combination of the vectors over j with unknown weights r_j gives

$$\prod_{s=1}^{d}(S - e^{i\omega_s}) \begin{bmatrix} \vdots \\ \sum_{j=1}^{d} e^{i\omega_j k} r_j \\ \vdots \end{bmatrix} = 0 \Rightarrow \prod_{s=1}^{d}(S - e^{i\omega_s}) \begin{bmatrix} \vdots \\ f_k \\ \vdots \end{bmatrix} = 0.$$

Define the polynomial $m(z) \equiv m_0 + \ldots + m_d z^d \equiv \prod_{s=1}^{d}(z - e^{i\omega_s})$. Then the last equality becomes

$$m(S) \begin{bmatrix} \vdots \\ f_k \\ \vdots \end{bmatrix} \equiv m_0 \cdot S^0 \begin{bmatrix} \vdots \\ f_k \\ \vdots \end{bmatrix} + \cdots + m_d \cdot S^d \begin{bmatrix} \vdots \\ f_k \\ \vdots \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} \vdots & \vdots & \cdots & \vdots \\ f_k & f_{k+1} & \cdots & f_{k+d} \\ \vdots & \vdots & \cdots & \vdots \end{bmatrix} \begin{bmatrix} m_0 \\ \ldots \\ m_d \end{bmatrix} = 0. \quad (5)$$

The final linear system contains a great deal of information.

- The rank of the matrix in (5) gives d.
- Any non-zero vector in the null space of the matrix in (5) gives the coefficients of $m_0, \ldots, m_d$ of the polynomial $m(z)$.
- The roots of $m(z)$ gives $\{e^{i\omega_j}\}$.
- Solving the least-squares problem

$$\min_{r_j} \sum_k \left| \sum_{j=1}^{d} e^{i\omega_j k} r_j - f_k \right|^2$$

  gives $\{r_j\}$.

Though we describe Prony's method using infinite vectors, it is clear now that only $d+1$ rows of the matrix is needed. Due to the shifting nature of the matrix, only $2d+1$ consecutive values of (f_k) are required.

The main advantages of the Prony's method are that (1) it is adaptive in the sense that $\{\omega_j\}$ do not need to fall in any discrete grid, (2) it is conceptually simple, and (3) it leverages standard numerical routines such as root-finding and null-space computation. The main disadvantage is that root-finding can often be unstable when noise is present.

3 Algorithm

3.1 Key components

We start by choosing the function $b(t)$ to be of the form

$$b(t) = b_d \sin(dt) + b_{d-2} \sin((d-2)t) + \ldots. \quad (6)$$

Here the leading coefficient is the most dominant one and the rest of the coefficients $(b_{d-2}, \ldots)$ are chosen randomly. The reason for doing so will be explained below.

Recall that the key function of the factorization approach is $e(z) = z^{-d} \prod_{|\xi_j|<1}(z - \xi_j)$, where the second term $\prod_{|\xi_j|<1}(z - \xi_j)$ is the characteristic polynomial of the $2d$ roots $\{\xi_j\}$ of $1 - a^2(z) - b^2(z)$ inside the unit circle.

3.1.1 Characteristic polynomial.

The first idea is that it is possible to compute the characteristic polynomial directly without first calculating the roots. This avoids the root-finding, which is the main source of instability of the factorization approach. A simple but key observation is that these roots are the poles of the reciprocal $g(z) = \left(1 - a^2(z) - b^2(z)\right)^{-1}$ inside the unit disk.

Since $g(z)$ is meromorphic, $g(z)$ takes the form

$$g(z) = \sum_{\xi_j} \frac{w_j}{\xi_j - z} + \text{constant},$$

where the sum is taken over the roots both inside and outside $\mathbb{D}$. Let us consider the integrals

$$\frac{1}{2\pi i} \int_\gamma \frac{g(z)}{z^k} \frac{\mathrm{d}z}{z} \tag{7}$$

for integer values of $k \leq -1$, where γ is the boundary of $\mathbb{D}$ in the counter clockwise orientation. For a fixed $k \leq -1$,

$$\begin{aligned}
&\frac{1}{2\pi i} \int_\gamma \frac{g(z)}{z^k} \frac{\mathrm{d}z}{z} = \frac{1}{2\pi i} \int_\gamma \left(\sum_{|\xi_j|<1} + \sum_{|\xi_j|>1} \right) \frac{w_j}{\xi_j - z} z^{-(k+1)} \mathrm{d}z \\
&= \frac{1}{2\pi i} \sum_{|\xi_j|<1} w_j \int_\gamma \frac{1}{\xi_j - z} z^{-(k+1)} \mathrm{d}z = \frac{1}{2\pi i} \sum_{|\xi_j|<1} w_j \xi_j^{-(k+1)} \int_\gamma \frac{1}{\xi_j - z} \mathrm{d}z = - \sum_{|\xi_j|<1} w_j \xi_j^{-(k+1)},
\end{aligned}$$

where the second equality relies on the analyticity of $\frac{w_j}{\xi_j - z}$ in $\mathbb{D}$ for $|\xi_j| > 1$ and the third equality uses the residue theorem at $\{\xi_j\}$. This computation shows that the integrals $\frac{1}{2\pi i} \int_\gamma g(z) z^k \mathrm{d}z$ for $k \leq -1$ contain important information about the poles inside $\mathbb{D}$.

The integral $\frac{1}{2\pi i} \int_\gamma \frac{g(z)}{z^k} \frac{\mathrm{d}z}{z}$ over the unit circle is also closely related to the Fourier transform of the function $g(t) \equiv g(e^{it})$:

$$\frac{1}{2\pi i} \int_\gamma \frac{g(z)}{z^k} \frac{\mathrm{d}z}{z} = \frac{1}{2\pi i} \int_0^{2\pi} g(t) e^{-ikt} i \mathrm{d}t = \frac{1}{2\pi} \int_0^{2\pi} g(t) e^{-ikt} \mathrm{d}t = \hat{g}_k. \tag{8}$$

In order to recover the characteristic polynomial $\prod_{|\xi_j|<1}(z - \xi_j)$ (the key part of $e(z)$), we apply Prony's method to the Fourier coefficients. Slightly different from the description in Section 2, we define the semi-infinite (instead of infinite) vector

$$\hat{g}_- \equiv \begin{bmatrix} \hat{g}_{-1} \\ \hat{g}_{-2} \\ \vdots \end{bmatrix} \equiv \frac{1}{2\pi i} \int_\gamma g(z) \begin{bmatrix} z^0 \\ z^1 \\ \vdots \end{bmatrix} \mathrm{d}z \equiv \begin{bmatrix} -\sum_{|\xi_j|<1} w_j \xi_j^0 \\ -\sum_{|\xi_j|<1} w_j \xi_j^1 \\ \vdots \end{bmatrix}$$

Let S be the shift operator that shifts the semi-infinite vector upward (i.e., dropping the first element). For any ξ_j with $|\xi_j| < 1$,

$$S \begin{bmatrix} \xi_j^0 \\ \xi_j^1 \\ \vdots \end{bmatrix} = \begin{bmatrix} \xi_j^1 \\ \xi_j^2 \\ \vdots \end{bmatrix}, \quad \text{i.e.,} \quad (S - \xi_j) \begin{bmatrix} \xi_j^0 \\ \xi_j^1 \\ \vdots \end{bmatrix} = 0.$$

Since the operators $S - \xi_j$ all commute,

$$\prod_{|\xi_i|<1} (S - \xi_i) \begin{bmatrix} \xi_j^0 \\ \xi_j^1 \\ \vdots \end{bmatrix} = 0. \tag{9}$$

Since $\hat{g}_-$ is a linear combination of such semi-infinite vectors with weights $\{-w_j\}$,

$$\prod_{|\xi_i|<1} (S - \xi_i)\, \hat{g}_- = 0.$$

Since $b(z)$ is chosen randomly, with probability 1 the roots $\{\xi_i\}$ are disjoint. Therefore, the polynomial $\prod_{|\xi_i|<1} (z - \xi_i)$ is of degree $2d$. By denoting it as

$$m(z) = m_0 z^0 + \cdots + m_{2d} z^{2d},$$

(9) becomes $m_0(S^0 \hat{g}_-) + \cdots + m_{2d}(S^{2d}\hat{g}_-) = 0$, i.e.,

$$\begin{bmatrix} \hat{g}_{-1} & \hat{g}_{-2} & \cdots & \hat{g}_{-(2d+1)} \\ \hat{g}_{-2} & \hat{g}_{-3} & \cdots & \hat{g}_{-(2d+2)} \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \begin{bmatrix} m_0 \\ \dots \\ m_{2d} \end{bmatrix} = 0. \tag{10}$$

At this point, $(m_0, \dots, m_{2d})$ can be computed as a non-zero vector in the null-space of the matrix in (10). Once $m(z)$ is obtained, we set $e(z) = z^{-d} m(z)$ as defined. Once $e(z)$ is ready, the Laurent polynomials $c(z)$, $d(z)$, $p(z) = a(z) + ic(z)$, and $r(z) = b(z) + id(z)$ follow from (4).

3.1.2 Phase factors from $p(z)$.

The construction of the actual phase factors is given as follows, essentially following Theorem 3 of [5] but in terms of the t variable.

For each $n = d$ down to 0, perform the following two steps

- In the t variable, $p(t)$ and $r(t)$ are trigonometric polynomials of degree n. Write $p(t) = p_n e^{int} + \dots$ and $r(t) = r_n e^{int} + \dots$, where p_n and r_n are the degree n coefficients. Solve ϕ_n from $e^{2i\phi_n} = p_n / r_n$.
- Transform $p(t)$ and $r(t)$ via

$$\begin{pmatrix} p(t) & r(t) \end{pmatrix} \Leftarrow \begin{pmatrix} p(t) & r(t) \end{pmatrix} \begin{pmatrix} e^{-i\phi_n} & 0 \\ 0 & e^{i\phi_n} \end{pmatrix} \begin{pmatrix} \cos(t) & -i\sin(t) \\ -i\sin(t) & \cos(t) \end{pmatrix}. \tag{11}$$

  This brings the top coefficients of $p(t)$ and $r(t)$ to zero, hence reducing the degree by one.

Within this loop, the switch between the angular function $p(t)$ and the coefficients $\{p_j\}_{-n \le j \le n}$ can be done with the fast Fourier transform (FFT). Because of the $O(d \log d)$ complexity of the FFT, the overall cost of this loop is $O(d^2 \log d)$.

3.1.3 Robust computation of polynomial coefficients

. The remaining issue is to compute $(m_0, \ldots, m_{2d})$ in a numerically stable way. This is in fact not always guaranteed. Consider for example the case that $a(z)$ has negligible coefficients for large frequency. If we set $b(z) = 0$, then $g(z) = (1-a^2(z)-b^2(z))^{-1} = (1-a^2(z))^{-1}$ might lack high frequency content. This implies that all coefficients $\hat{g}_k$ might be be negligible for large k values. A direct consequence is that the matrix in (10) might have a numerical rank much smaller than $2d$. In order to resolve this issue, we choose $b(z)$ to have a large leading coefficient as suggested in (6). This is the second contribution of this paper.

Figure 2 illustrates the difference between $b(t) = 0$ and $b(t) \sim \sin(dt) + \ldots$ for the Fermi-Dirac operator (the last example in Section 4) at $\beta = 100$. Notice that the leading term $\sin(dt)$ in $b(t)$ introduces a dominant anti-diagonal in the matrix of (10), ensuring that it has numerical rank $2d$.

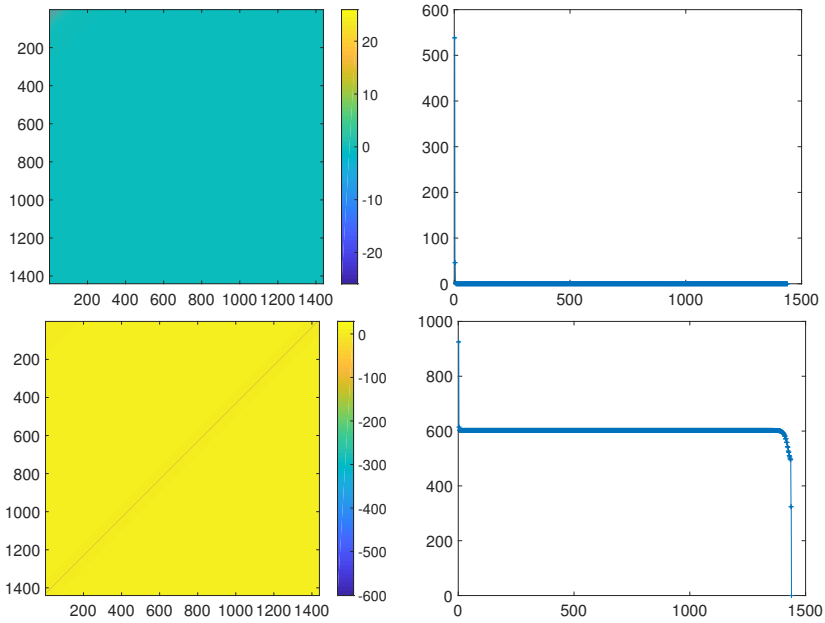

Figure 2: Comparison between $b(t) = 0$ and $b(t) \sim \sin(dt) + \ldots$. The top row is for $b(z) = 0$ and the bottom row is $b(t) \sim \sin(dt) + \ldots$. Within each row, the left plot is the matrix in (10) and the right is its singular values. Notice that the leading term $\sin(dt)$ in $b(t)$ introduces a dominant anti-diagonal in the matrix, ensuring that it has numerical rank $2d$.

3.2 Implementation

To implement this algorithm numerically, we need to take care several issues.

- The computation (8) requires the Fourier transform of $g(t) = (1 - a^2(t) - b^2(t))^{-1}$ for $t \in [-\pi, \pi]$. When $1 - a^2(t) - b^2(t)$ is close to zero, $g(t)$ is near singular and hence it is hard for numerical quadrature. In practice, we make sure that $\|a\|_\infty$ and $\|b\|_\infty$ are bounded by $1/3$.

- To compute the Fourier coefficients $\{\hat{g}_k\}$, choose an even N_s and define for $n = 0, \ldots, N_s - 1$ the point $t_n = \exp\left(i\frac{2\pi n}{N_s}\right)$ on the unit circle. Using samples $\{g(t_n)\}$ at the points $\{t_n\}$

corresponds to approximating (7) with the trapezoidal rule. The trapezoidal rule is exponentially convergent for smooth functions when N_s is sufficient large. In the current setting since $1 - a^2(t) - b^2(t)$ is bounded well away from zero, $g(t)$ does not exhibit singular behaviors. As a result, the highest non-trivial frequency in $g(t)$ is on the same order of the highest non-trivial frequency in $a(t)$ and $b(t)$. Therefore, by setting N_s to be about 40 times d in practice, we ensure that the trapezoidal rule is exponentially accurate. Applying the fast Fourier transform to $\{g(t_n)\}$ gives the Fourier coefficients $\{\hat{g}_k\}$.

- The semi-infinite matrix in (10). In the implementation, we only pick the first l rows of this semi-infinite matrix and define

$$H \equiv \begin{bmatrix} \hat{g}_{-1} & \hat{g}_{-2} & \cdots & \hat{g}_{-(2d+1)} \\ \hat{g}_{-2} & \hat{g}_{-3} & \cdots & \hat{g}_{-(2d+2)} \\ \vdots & \vdots & \vdots & \vdots \\ \hat{g}_{-l} & \hat{g}_{-(l+1)} & \cdots & \hat{g}_{-(2d+l)} \end{bmatrix} \tag{12}$$

  with $l \geq 2d+1$. In practice setting $l = 2d+2$ seems to be sufficient.

- The computation of the vector m. The most straightforward way is to compute the singular value decomposition (SVD) of H in (12) and take m to be the last column of the V matrix, which unfortunately has $O(d^3)$ time complexity.

  This complexity can be improved based on the following observation. Let $s_1, \ldots, s_{2d+1}$ be the singular values of H. Numerically, our choice of $b(t)$ leads to a large gap between s_{2d} and s_{2d+1} that is actually proportional to s_1. As a result, we propose the following iterative procedure

$$m \Leftarrow \text{normalize}\left(\left(\epsilon I + H^T H\right)^{-1} m\right), \tag{13}$$

  where ϵ is small positive constant. The linear system solve within each iteration is done with the conjugate gradient (CG) method and the iteration stops when the difference between the new and old m is less than the machine accuracy. Due to the large spectral gap of the matrix H, the inner CG method typically converges within a constant number of iterations and the outer iteration stops in 3-4 iterations. Since the matrix $\epsilon I + H^T H$ is of size $(2d+1)\times(2d+1)$, the empirical cost for computing m is $O(d^2)$. Combined with the $O(d^2 \log d)$ cost of extracting the phase factors (11), the overall cost is $O(d^2 \log d)$.

  The complexity for computing m can be further reduced by observing that H is a Hankel matrix. Therefore, the matrix-vector multiplications of H and H^T can be accelerated to the $O(d \log d)$ cost via the fast Fourier transform (FFT). This improvement does not impact the overall computational cost, since the construction of the phase factors from $p(z)$ (Section 3.1.2) already takes $O(d^2 \log d)$ steps. On the other hand, since there is no need to store the full H matrix, the memory cost is reduced to $O(d)$, allowing for working with very large d values.

A few remarks are in order here.

- The symmetric phase factors obtained in [4] correspond to $b(t)$=0, which helps the optimization approach. In the current algorithm, the choice of $b(t) = b_d \sin(dt) + \ldots$ helps the direct factorization method and the resulting phase factors are non-symmetric.

- Whether symmetric or non-symmetric phase factors are preferred in practice is not clear. The main part of implementing QSVT on quantum circuits as in (3) is actually related to the terms e^{itX}, i.e., the implementation of the circuit U_A. The actual choice of the phase factors might very well depend on the circuit architecture.

4 Results

4.1 Setup

The algorithm is implemented with the standard double precision arithmetics. All numerical results are obtained on a laptop with a 2.6 GHz 6-Core Intel Core i7 CPU. The computation of the vector m is performed via the iteration (13). The complexity is quadratic in terms of the degree d and the actual computation typically finishes within a couple of minutes.

As mentioned in Section 1, when the target function $f(x)$ is not a polynomial, the first step is to construct an accurate polynomial approximation $a(x)$. Since polynomial approximation in x is equivalent to trigonometric approximation in t, this task is performed the t space. Let us introduce an equally spaced grid $t_n = \exp\left(i\frac{2\pi n}{N_s}\right)$ on the unit circle, where the grid size N_s is taken in practice to be 40 times d as before.

- First, the values of $f(t)$ at $\{t_n\}$ are computed.
- After applying fast Fourier transforms to $\{f(t_n)\}$, we identify a frequency d such that all Fourier coefficients above frequency d are below a threshold multiplied by the maximum Fourier coefficient in absolute value. In the experiments, the threshold is chosen to be around 10^{-12} since this is right above the accuracy of the QSP algorithm in double precision arithmetics [4] and further improvement below this threshold is not necessary. Here d is enforced to have the same parity as $f(x)$.
- The Fourier coefficients above frequency d are then set to zero and Fourier transform back gives the desired trigonometric approximation to $f(t)$ in the t space. The final function $a(t)$ is also scaled to have infinity norm equal to 0.3.

Regarding the choice of $b(t)$, simply setting $b(t) = 0.4 \cdot \sin(dt)$ suffices for in the examples presented below. However, in principle, one might need to choose to $b(t)$ according to (6) in order to avoid identical roots.

The polynomial coefficient vector m is computed with the iterative procedure (13). The error of the constructed phase factors is measured in the relative L_∞ norm. More precisely, we compute a function $\tilde{p}(x)$ following (1) using the constructed phase factors $\Phi = (\phi_0, \ldots, \phi_d)$. With $\tilde{p}(x)$ as an approximation to $p(x)$, the error of the phase factor computation is estimated with

$$\frac{\|\mathrm{Re}(\tilde{p}(x)) - a(x)\|_\infty}{\|a(x)\|_\infty}, \tag{14}$$

over the interval $[-1, 1]$.

4.2 Examples

In what follows, we present the numerical results for four examples: Hamiltonian simulation, eigenstate filtering, matrix inversion, and Fermi-Dirac operator. Among them, the

first three examples are also studied numerically in the paper [4], arguably the most complete numerical study on the phase factor computation. In each example, we have chosen the parameters to be on par or harder compared with those used in [4], so that the actual instances have at least the same level of difficulty. Overall, our algorithm exhibits similar accuracy but short wall clock time when compared with [4]. The longest sequence reported below consists of more than 50000 phase factors.

4.2.1 Hamiltonian simulation

Assume that the Hamiltonian H satisfies $\|H\|_2 \leq 1$. Hamiltonian simulation for a period of time τ boils down to the quantum signal processing problem for $f(x) = e^{-i\tau x}$. The even and odd parts are

$$\mathrm{Re}(f(x)) = \cos(\tau x), \quad \mathrm{Im}(f(x)) = \sin(\tau x).$$

In terms of the variable t,

$$\mathrm{Re}(f(t)) = \cos(\tau \cos(t)), \quad \mathrm{Im}(f(t)) = \sin(\tau \cos(t)).$$

Since both functions are not polynomials, we first use the procedure described above to compute trigonometric polynomial approximations $a_{\mathrm{Re}}(t) \approx 0.3 \cdot \mathrm{Re}(f(t))$ and $a_{\mathrm{Im}}(t) \approx 0.3 \cdot \mathrm{Im}(f(t))$, both scaled so that L_∞ norm is below $1/3$. $a_{\mathrm{Re}}(x)$ is even in x with even degree d_{Re} while $a_{\mathrm{Im}}(x)$ is odd in x with odd degree d_{Im}. We perform the tests with $\tau = 1000, 2000, 3000, 4000, 5000$ and the results are summarized in Figure 3.

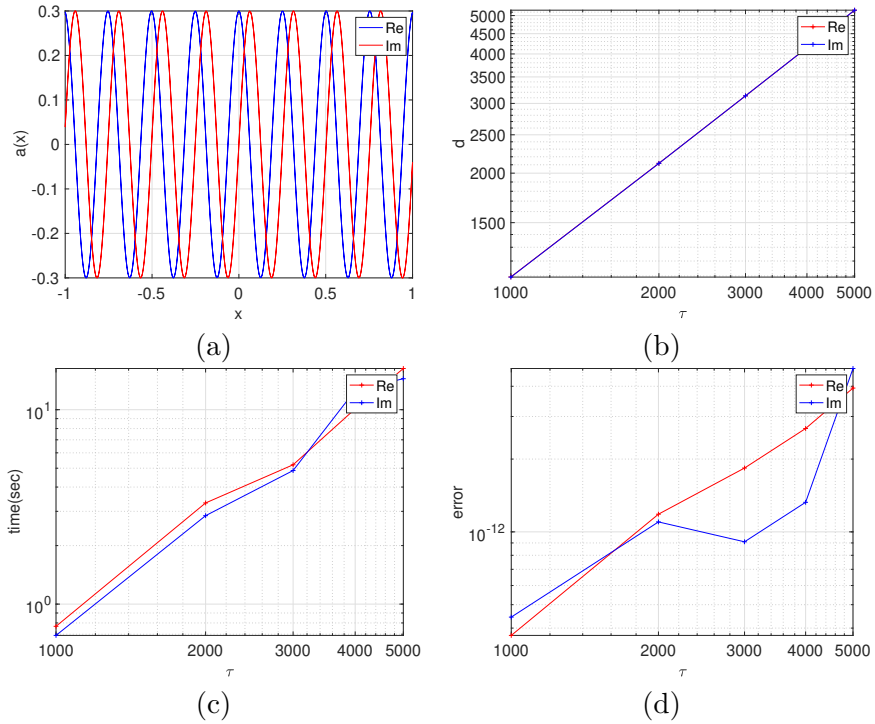

Figure 3: Hamiltonian simulation. (a) $a_{\mathrm{Re}}(x)$ and $a_{\mathrm{Im}}(x)$ for a smaller $\tau = 25$. (b) degree d of the polynomial approximation as a function of τ. (c) The total phase factor construction time in seconds as a function of τ. (d) The relative L_∞ norm error (14) as a function of τ.

4.2.2 Eigenstate filtering

For a fixed gap Δ, we follow [4] and consider the filtering function centered at the origin

$$f(x) = \frac{T_k\left(-1 + 2\frac{x^2-\Delta^2}{1-\Delta^2}\right)}{T_k\left(-1 + 2\frac{-\Delta^2}{1-\Delta^2}\right)},$$

where T_k is the Chebyshev polynomial. The parameter k is set to be $20/\Delta$ so that the $f(x)$ is negligible outside the Δ neighborhood of the origin. In terms of variable t, the function

$$f(t) = \frac{T_k\left(-1 + 2\frac{\cos(t)^2-\Delta^2}{1-\Delta^2}\right)}{T_k\left(-1 + 2\frac{-\Delta^2}{1-\Delta^2}\right)}.$$

Since $f(t)$ is already a trigonometric polynomial, we simply set $a(t) = 0.3 \cdot f(t)$. The tests are performed with $\Delta = 0.08, 0.04, 0.02, 0.01, 0.005$ and the results are summarized in Figure 4.

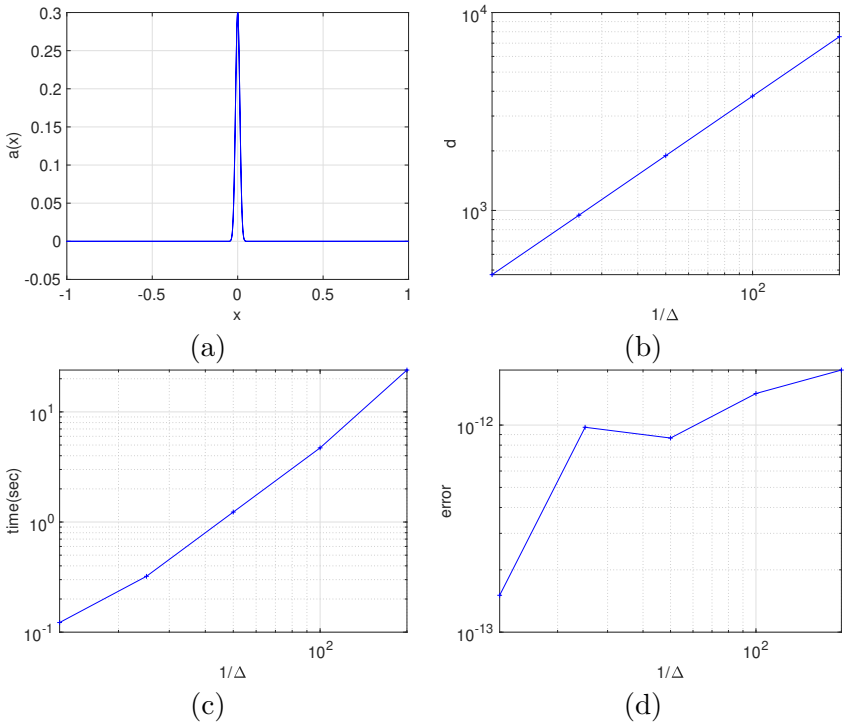

Figure 4: Eigenstate filtering. (a) $a(x)$ for $\Delta = 0.08$. (b) The degree d of the polynomial approximation as a function of $1/\Delta$. (c) The total phase factor construction time in seconds as a function of $1/\Delta$. (d) The relative L_∞ norm error (14) as a function of $1/\Delta$.

4.2.3 Matrix inversion

We consider the inversion of the matrices with spectrum resided in $D_\kappa = [-1, -1/\kappa] \cup [1/\kappa, 1]$, where κ is the condition number. The QSP problem here amounts to approximating

the function $1/x$ over D_κ. We choose

$$f(x) = \frac{1 - e^{-(5\kappa x)^2}}{x},$$

where the difference between $f(x)$ and $1/x$ over D_κ is negligible under the double precision arithmetics. In the t variable, this is

$$f(t) = \frac{1 - e^{-(5\kappa \cos(t))^2}}{\cos(t)}.$$

The procedure mentioned above is used to compute a trigonometric approximation $a(t)$ to $f(t)$ (up to a constant factor) with $\|a\|_\infty = 0.3$. The tests are performed with $\kappa = 16, 64, 256, 1024$ and the results are summarized in Figure 5. The longest sequence for $\kappa = 1024$ has more than 50000 phase factors.

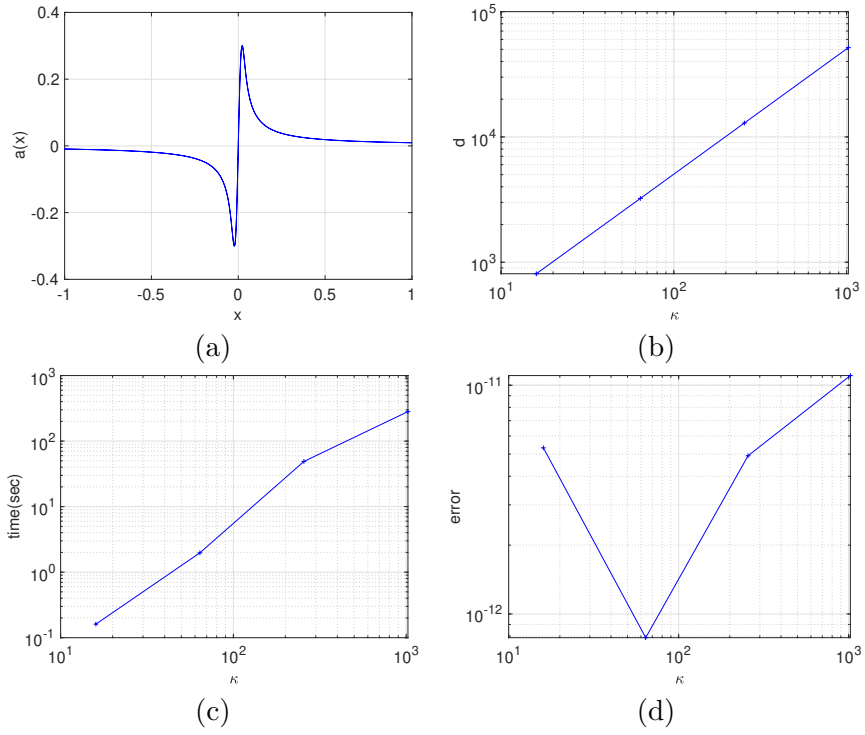

Figure 5: Matrix inversion. (a) $a(x)$ for $\kappa = 10$. (b) The degree d of the polynomial approximation as a function of κ. (c) The total phase factor construction time in seconds as a function of κ. (d) The relative L_∞ norm error (14) as a function of κ.

4.2.4 Fermi-Dirac operator

Finally, we consider the (shifted) Fermi-Dirac function

$$f(x) = \frac{1 - e^{\beta x}}{1 + e^{\beta x}}.$$

In the t variable, it takes the form

$$f(t) = \frac{1 - e^{\beta \cos(t)}}{1 + e^{\beta \cos(t)}}.$$

The procedure mentioned above is used to compute a trigonometric approximation $a(t)$ to $f(t)$ (up to a constant factor) with $\|a\|_\infty = 0.3$. We perform the tests for $\beta = 100, 200, 400, 800, 1600$ and the results are summarized in Figure 6.

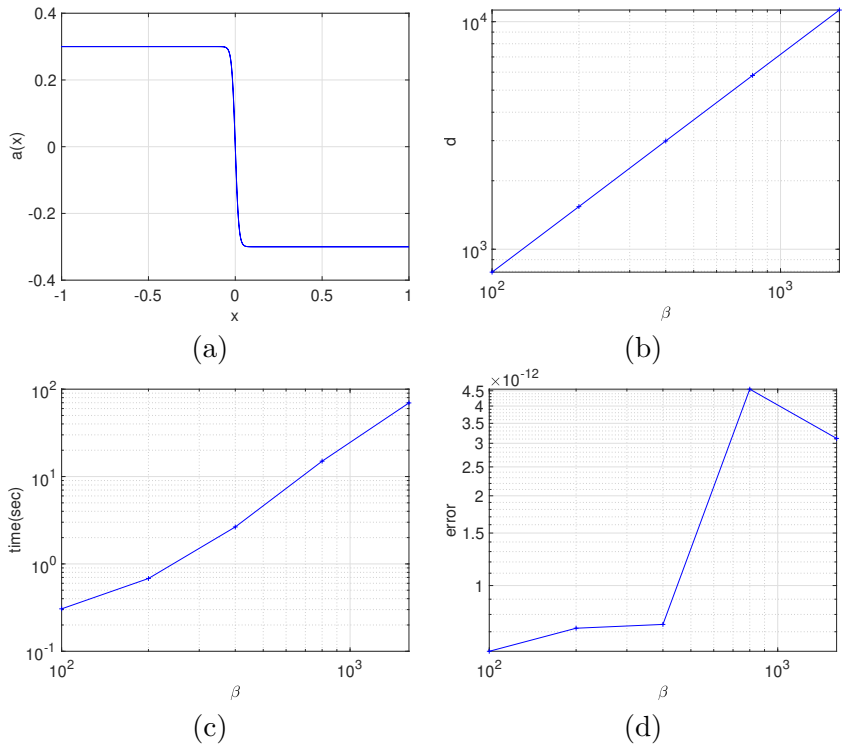

Figure 6: Fermi-Dirac operator. (a) $a(x)$ for $\beta = 100$. (b) The degree d of the polynomial approximation as a function of β. (c) The total phase factor construction time in seconds as a function of β. (d) The relative L_∞ norm error (14) as a function of β.

5 Discussions

In this paper, we proposed a new factorization algorithm for computing the phase factors of quantum signal processing. The proposed algorithm avoids root finding of high degree polynomials by using a key component of the Prony's method. The resulting algorithm is numerically stable in the double precision arithmetics. We have demonstrated the numerical performance with several important examples, including Hamiltonian simulation, eigenstate filtering, matrix inversion, and Fermi-Dirac operator. For future work, the immediate question is to prove theoretically the stability of the algorithm with the proposed choice of $b(t)$.

Acknowledgments:

The author thanks Lin Lin for introducing the topic of quantum signal processing and the reviewers for constructive comments.

References

[1] 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. doi:10.48550/ARXIV.2003.02831.

[2] A. M. Childs, R. Kothari, and R. D. Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017. doi:10.1137/16M1087072.

[3] A. M. Childs, D. Maslov, Y. Nam, N. J. Ross, and Y. Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018. doi:10.1073/pnas.1801723115.

[4] Y. Dong, X. Meng, K. B. Whaley, and L. Lin. Efficient phase-factor evaluation in quantum signal processing. *Physical Review A*, 103(4):042419, 2021. doi:10.1103/PhysRevA.103.042419.

[5] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe. Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics. *arXiv preprint arXiv:1806.01838*, 2018. doi:10.48550/arXiv.1806.01838.

[6] 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. doi:10.1145/3313276.3316366.

[7] J. Haah. Product decomposition of periodic functions in quantum signal processing. *Quantum*, 3:190, 2019. doi:10.22331/q-2019-10-07-190.

[8] L. Lin. Lecture notes on quantum algorithms for scientific computation. *arXiv preprint arXiv:2201.08309*, 2022. doi:10.48550/arXiv.2201.08309.

[9] G. H. Low and I. L. Chuang. Optimal hamiltonian simulation by quantum signal processing. *Physical review letters*, 118(1):010501, 2017. doi:10.1103/PhysRevLett.118.010501.

[10] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2(4):040203, 2021. doi:10.1103/PRXQuantum.2.040203.

[11] D. Potts and M. Tasche. Parameter estimation for nonincreasing exponential sums by Prony-like methods. *Linear Algebra and its Applications*, 439(4):1024–1039, 2013. doi:10.1016/j.laa.2012.10.036.

[12] R. Prony. Essai experimental et analytique. *J. Ecole Polytechnique*, pages 24–76, 1795.

[13] J. Van Apeldoorn, A. Gilyén, S. Gribling, and R. de Wolf. Quantum SDP-solvers: Better upper and lower bounds. *Quantum*, 4:230, 2020. doi:10.22331/q-2020-02-14-230.

[14] J. Wang, Y. Dong, and L. Lin. On the energy landscape of symmetric quantum signal processing. *arXiv preprint arXiv:2110.04993*, 2021. doi:10.48550/arXiv.2110.04993.