Quantum Singular Value Transform and Function Evaluation Problems

Olga Okrut[1] and Kaden Khweled[2]

[1]Department of Mathematics, University of California, Berkeley
[2]Department of Physics, University of California, Berkeley

May 1, 2024

Abstract

A Quantum Singular Value Transform (QSVT), a unifying framework for the number of quantum algorithms, represents a powerful technique to perform a polynomial transformation of a singular values of the unitary operators on a quantum computer. Here, we provide a theoretical exploration of the QSVT and its practical application to the function evaluation problem. We demonstrate the application of the QSVT to the forth degree polynomial fitting and the matrix inversion problems and discuss challenges and opportunities in practical application of these algorithms.

1 Introduction

The QSVT algorithm, pioneered by Gilyén, Su, Low, and Wiebe [1], is believed to be a unification of various quantum algorithms such as Grover's search, amplitude amplification, evaluation of matrix functions, to name a few [2]. The QSVT algorithm generalizes the QSP and efficiently applies a polynomial transformation to the singular values of a linear operator governing a particular subsystem embedded into a larger unitary operator. The transformation is achieved by the sequence of specific SU(2) rotations on the embedded subsystem(s) [2, 3, 1] where each rotation being parameterized by the phase angels $\phi_i \in \mathbb{R}^d$. The phase angles must be computed classically using optimization algorithms before constructing SU(2) rotations [4, 5].

Matrix function computation, an evaluation of $f(A)$ where $f(x)$ is a smooth real or complex valued function, is the core idea of various computational tasks, including Hamiltonian simulation, solving linear equations, thermal state preparation, and constructing a quantum-based propagator for a system of differential equations [6]. In this project, we explore the QSVT from the perspective of the cosine-sine decomposition (CSD) and qubitization connecting these concepts to the singular values and the Singular Value Decomposition (SVD) of an operator. We further explore the application of the QSVT to the function evaluation problem using the Pennylane scientific package for quantum computation [7, 5, 8] and discuss challenges and opportunities represented by the practical application of the QSVT.

2 Singular Values and SVD

Definition. Let $A \in \mathbb{C}^{m\times n}$, and let r be the rank of A. The singular value decomposition (SVD) of the normalized matrix A can be written as [3, 2, 9]

$$A = W\Sigma V^\dagger \tag{1}$$

or equivalently

$$A\left|v_i\right\rangle = \sigma_i \left|w_i\right\rangle, \; A^\dagger \left|w_i\right\rangle = \sigma_i \left|v_i\right\rangle$$

where the $\sigma_i \geq 0$ are called the *singular values* of the matrix A. The matrices $W \in \mathbb{C}^{m\times m}$ and $V^\dagger \in \mathbb{C}^{n\times n}$ are unitary matrices where their columns $\left|w_i\right\rangle$ and $\left|v_i\right\rangle$, respectively, contain left and right singular vectors. The matrix $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with the singular values on its main diagonal: $\Sigma = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_{r+1})$. We note that the matrix Σ is uniquely defined while W and V are not [9].

An alternative to 1 definition of the SVD is the definition in terms of the orthonormal bases vectors that follows the form of the eigenvalue decomposition [2]

$$A = \sum_{i=1}^{N} \sigma_i \left|w_i\right\rangle \left\langle v_i\right| \tag{2}$$

To be able to transform a unitary matrix by a function f, one has to introduce the core concept we will rely on when exploring and applying the QSVT and QSP.

Definition. Let $A \in \mathbb{C}^{m\times n}$ be a matrix of the rank r, and let 1 be its SVD. Let $f : \mathbb{R} \to \mathbb{R}$ be a scalar function such that $f(\sigma_i)$ is defined for all $i = 1, 2, \ldots r$. Then the generalized matrix function is defined as [9, 2, 3]

$$f(A) = Wf(\Sigma)V^\dagger \tag{3}$$

where $f(\Sigma) = \text{diag}(f(\sigma_1), f(\sigma_2), ..., f(\sigma_r))$. An alternative definition of the matrix function can be provided using the second definition of the SVD as in 2. Let A be defined as in 3. Then the generalized matrix function is defined as

$$f(A) = \sum_{i=1}^{N} f(\sigma_i) \left|w_i\right\rangle \left\langle v_i\right| \tag{4}$$

Singular values and the SVD have a wide range of applications in various fields such as Data Science, Machine Learning, image and signal processing, and more. In recent years, quantum algorithm communities have shown significant interest in these concepts and have been exploring their potential applications in the domain of quantum algorithms. To summarize the definitions introduced in this section, we conclude this section with an illustrative example.

Example Let the matrix $A \in \mathbb{R}^{2\times 2}$ be defined as [10]

$$\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

It has SVD as follows

$$A = W\Sigma V^\dagger = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 3/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ -\sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix}$$

If we want to transform the matrix A by the function rule $f(x) = x^2$ (square of the matrix), then we apply $f(x)$ to the matrix with the singular values of A:

$$f(A) = Vf(\Sigma)W^\dagger = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 9/4 & 0 \\ 0 & 1/4 \end{bmatrix} \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} = \begin{bmatrix} 5/4 & -1 \\ -1 & 5/4 \end{bmatrix}$$

The other example is transforming the matrix A by the function rule $f(x) = \frac{1}{x}$ which is the matrix inversion problem. We can act similarly, calculating only the inverse of the diagonal elements of the singular matrix.

$$f(A) = Vf(\Sigma)W^\dagger = \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 2/3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -\sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & \sqrt{2}/2 \end{bmatrix} = \begin{bmatrix} 4/3 & 2/3 \\ 2/3 & 4/3 \end{bmatrix}$$

3 QSP and QSVT

We will start exploring QSVT with a simple case of QSP, which allows transforming a scalar through a function rule. We will then generalize it for arbitrary matrix transformation through a function rule.

3.1 QSP

The Quantum Signal Processing (QSP) is a unitary representation of a scalar value through the use of two types of single-qubit rotations, namely signal rotation and signal processing operators [2, 3]. The definition of QSP may differ slightly depending on the chosen convention, but all definitions share the same underlying idea.

Definition. [O-convention of the QSP] Let $a \in [-1, 1]$ be a scalar with one-qubit block encoding (we discuss the block encoding in section 3.2), and let the unitary $U(a)$ be defined by

$$U(a) = \begin{bmatrix} a & \sqrt{1-a^2} \\ \sqrt{1-a^2} & -a \end{bmatrix} \tag{5}$$

Then

$$O(a) = U(a)Z = \begin{bmatrix} a & -\sqrt{1-a^2} \\ \sqrt{1-a^2} & a \end{bmatrix} \tag{6}$$

is a rotation matrix [3], and Z is a Pauli quantum operator.

Definition. [W-convention of the QSP] Let $a \in [-1, 1]$ be a scalar with one-qubit block encoding, and let unitary $W(a)$ be defined by

$$W(a) = e^{i\arccos(a)X} = \begin{bmatrix} a & i\sqrt{1-a^2} \\ i\sqrt{1-a^2} & a \end{bmatrix} \tag{7}$$

where X is a Pauli quantum operator [2]. Both operators $O(a)$ and $W(a)$ are the *signal rotation operators* that represent an x-rotation by the angle $\theta = -2\arccos(a)$.

A signal processing operator represents a z-rotation by the angle -2ϕ and is defined by [2]

$$S(\phi) = e^{i\phi Z} \tag{8}$$

By interleaving the signal rotation and the signal processing operators, one can achieve a polynomial function transformation of the scalar a according to the Quantum Signal Processing Theorem.

Theorem [Quantum Signal Processing]. There exists a sequence of the phase factors $\phi_i = (\phi_0, \ldots, \phi_d) \in \mathbb{R}^{d+1}$ such that [2, 3, 1]

$$U_\phi(a) = e^{i\phi_0 Z} \prod_{i=1}^{d} O(a) e^{i\phi_i Z} = \begin{bmatrix} P(a) & -Q(a)\sqrt{1-a^2} \\ Q(a)\sqrt{1-a^2} & P^*(a) \end{bmatrix} \tag{9}$$

if following the O convention, and

$$U_\Phi(a) = e^{i\phi_0 Z} \prod_{i=1}^{d} W(a) e^{i\phi_i Z} = \begin{bmatrix} P(a) & iQ(a)\sqrt{1-a^2} \\ iQ^*(a)\sqrt{1-a^2} & P^*(a) \end{bmatrix} \tag{10}$$

if following the W convention. The polynomials $P, Q \in \mathbb{C}[a]$ must satisfy the following conditions:

- $deg(P) \leq$ and $deg(Q) \leq d-1$
- P has parity d mod 2 and Q has parity $d-1$ mod 2.
- $|P(a)|^2 + (1-a^2)|Q(a)|^2 = 1$

Note. O and W conventions could be converted into one another using the relation [3]

$$W(a) = e^{-\frac{\pi i}{4} Z} O(a) e^{\frac{\pi i}{4} Z}$$

and the relation between the phase angles is

$$\phi_i^O = \begin{cases} \phi_0^W - \pi/4 & i = 0 \\ \phi_i^W & i = 1, \ldots, d-1 \\ \phi_d^W + \pi/4 & i = d \end{cases}$$

To transform a matrix or a scalar using by the function $f(x)$ using numerical approach, one needs a tool that can help in finding an approximation of the desired function $f(x)$ within a certain error margin. The main tool used for functions approximation when working with the QSP or the QSVT frameworks is the Chebyshev polynomials. These polynomials are bounded and orthonormal and can provide a low error interpolation of a target function.

Definition. The d-th degree Chebyshev polynomials of the first kind are the polynomials of the form [1]

$$T_d(x) = \cos(d \arccos(x)) \tag{11}$$

where each successive term is defined recursively as $T_{d+1}(x) = 2xT_d(x) - T_{d-1}(x)$.

Provided that a target to approximation function $f(x)$ satisfies the Lipschitz criteria, in other words $|f(x) - f(y) \leq L|x-y|$ for finite $L > 0$, then such a function $f(x)$ has a unique decomposition into Chebyshev polynomials

$$f(x) = \sum_{d=0}^{\infty} \alpha_d T_d(x) \tag{12}$$

where the Chebyshev coefficients α_d absolutely converge [1, 9].

In practice, one uses the n-th truncation of Chebyshev polynomials limiting the infinite sum in 12 up to n-th degree: $f(x) = \sum_{d=0}^{n} \alpha_d T_d(x)$. To compute the Chebyshev coefficients α_d, the so-called technique of the *Chebyshev interpolant* is used where the interval of the interpolation of the function $f(x)$ is partitioned into the set of points $\{cos(\frac{i\pi}{n}) : 0 \leq j \leq n\}$ [1].

It's important to note that while the Quantum Signal Processing Theorem 3.1 guarantees the existence of a sequence of phase angles ϕ_i, it doesn't provide instructions on how to obtain that sequence. In practice, the Chebyshev coefficients α_d are used as an input parameters, and optimization algorithms are then applied to solve for the phase angles ϕ_i [3, 1, 4]. As the several scientific papers and tutorials suggest [3, 4], this area still needs further development and is currently being explored by scientific communities. Some examples of the frameworks that attempt to address this challenge include pyqsp and QSPPACK. In our project, we will rely on third-party optimization algorithms to optimize for the phase angles, rather than developing our own.

The Lemma 9 in [1] guarantees the connection between the Chebyshev polynomials and the polynomial transformation in the QSP.

Lemma. Let $T_d \in \mathbb{R}[x]$ be the d-the degree Chebyshev polynomial of the firs kind. Let $\phi \in \mathbb{R}^d$ be the sequence of phase angles such that $\phi_1 = (1-d)\frac{\pi}{2}$ and for all $i \in [d]\backslash\{1\}$ and let $\phi_i = \frac{\pi}{2}$. Then $P(a) = T_d(a)$. In other words,

$$\prod_{i=1}^{d} e^{i\phi_i Z} U(a) = \begin{bmatrix} P(a) & * \\ * & * \end{bmatrix} = \begin{bmatrix} T_d(a) & * \\ * & * \end{bmatrix} \tag{13}$$

where $*$ denotes the entries that could be omitted.

The other important aspect to mention is the *signal basis* in which the desired polynomial transformation is obtained. Similarly to the projective measurement, we can measure out the unitary U_ϕ in $\{|0\rangle, |1\rangle\}$ or $\{|+\rangle, |-\rangle\}$ basis to obtain the polynomial transformation $P(a)$. In other words, the goal is to obtain only the upper-left corner entry from the QSP unitary. For instance, measuring in $\{|0\rangle, |1\rangle\}$ basis, we get $P(a) = \langle 0| |U_\phi| |0\rangle$. When $a = \pm 1$ then $W(a)$ is proportional to the identity matrix, and the polynomial $P(a)$ is limited to ± 1. To overcome such a limitation, one can measure out the unitary U_ϕ in $\{|+\rangle, |-\rangle\}$ basis: $P(a) = \langle +| |U_\phi| |+\rangle$ [2]. When the projective measurement is generalized over matrix transformations rather than the scalar transformation, then they are called the *projectors* [2, 3, 1].

We conclude this subsection with an example illustrating the connecting between Chebyshev polynomials and the QSP.

Example Let the $\phi_1 = -3\pi/2$, $\phi_2 = \phi_3 = \phi_4 = \pi/2$ be the sequence of phase angles. Then following the O convention 6, we can obtain the unitary U_ϕ:

$$U_\phi = S(\phi_1)U(a)S(\phi_2)U(a)S(\phi_3)U(a)S(\phi_4)U(a) = \begin{bmatrix} 8a^4 - 8a^2 + 1 & * \\ * & * \end{bmatrix}$$

Applying the projector, we can measure out the polynomial located in the top left corner of the unitary U_ϕ:

$$\langle 0| U_\phi |0\rangle = 8a^4 - 8a^2 + 1$$

Note that $8a^4 - 8a^2 + 1$ is the 4-th degree Chebyshev polynomial $T_4(a)$ by definition.

3.2 Qubitization

There are two main methods to extend the results of the polynomial conversion from the QSP into a higher dimension, namely qubitization and Cosine-Sine Decomposition 3.3. Both methods achieve the same outcome, but they use different mathematical principles. In this section, we will explain the concept of qubitization and generalize the results of the QSP.

The high-level idea behind the qubitization is to find two-dimensional invariant subspaces $\mathcal{H}_i$ of the larger space $\mathcal{H}$ such that one can apply the QSP on each of these smaller subspaces $\mathcal{H}_i$ [1].

In order to perform computations on a matrix, this matrix must be inputted into a quantum computer. However, quantum computers can only handle unitary matrices, hence, any not-unitary matrix must be encoded in terms of the unitary operators. If A is an arbitrary matrix (non-unitary, non-Hermitian) without a specific structure, encoding such a matrix as a unitary operator is generally complex and expensive process [3]. The block encoding method shows promise, but it is still an active area of research within the quantum scientific communities. Despite this, the theoretical results of block encoding can be used to justify the use of quantum singular value transformation (QSVT).

Definition. Given an n-qubit matrix A, if we can find $\alpha > 0$, $\epsilon > 0$ and an $(m+n)$-qubit unitary matrix U_A so that

$$||A - \alpha(|0\rangle^{\otimes m} \otimes I_n)U_A(|0\rangle^{\otimes m} \otimes I_n)|| \leq \epsilon \tag{14}$$

then U_A is called an (α, m, ϵ)-*block-encoding* of A [3].

Example. The simplest example of block encoding is a $(n+1)$-qubit unitary matrix U such that

$$U_A = \begin{bmatrix} A & * \\ * & * \end{bmatrix} = |0\rangle \langle 0| \otimes A + \dots$$

where $||A|| \leq 1$.

In terms of the matrix block encoding, qubitization can be represented as a decomposition of the block encoding into two-dimensional subspaces. Let matrix A have the eigenvalues λ with corresponding eigenvectors $|\psi_\lambda\rangle$, and let define the higher dimensional projector $\Pi = |0\rangle \langle 0| \otimes I$ such that $\Pi U \Pi = |0\rangle \langle 0| \otimes A$. Let $|\lambda\rangle = |0\rangle \otimes |\psi_\lambda\rangle$ such that $\Pi U \Pi |\lambda\rangle = \lambda |\lambda\rangle$. Then the unitary U_A can be expressed as the direct sum over invariant subspaces [1, 3]

$$U_A = \bigoplus_\lambda \begin{bmatrix} \lambda & \sqrt{1-\lambda^2} \\ \sqrt{1-\lambda^2} & -\lambda \end{bmatrix} \tag{15}$$

which is closely reassembles the reflection operator as defined in O convention 6. In other words, a block encoded unitary U_A is decomposed into the direct sum of the reflection operators.

Similarly to the QSP, the polynomial transformation can be induced on each of the invariant subspaces and, thus, on the entire matrix A. For the sequence of the phase angles ϕ_i, the block encoding of the polynomial matrix transformation $P(A)$ is defined by:

$$\begin{bmatrix} P(A) & * \\ * & * \end{bmatrix} = e^{i\phi_0(2\Pi-1)}U^\dagger e^{i\phi_1(2\Pi-1)}U e^{i\phi_2(2\Pi-1)}U^\dagger e^{i\phi_3(2\Pi-1)}U \dots e^{i\phi_{k-1}(2\Pi-1)}U^\dagger e^{i\phi_k(2\Pi-1)}U \tag{16}$$

To implement such a polynomial transformation as a quantum circuit, the projectors Π are implemented as the Π-controlled NOT gate that flips the value of an ancilla bit conditioned on the state of the main register being in the space projected onto by Π [1].

3.3 Cosine-Sine Decomposition

The CS Decomposition offers an alternative to the qubitization approach to find a singular value transformation. Here we will focus on square blocks but the most general case of CSD where diagonal blocks are not necessarily square. Given a unitary matrix U we can split it into a two-by-two blocks of other unitarys U_{ij} we can then find the simultaneous SVDs of these sub matrices [11].

Theorem [Cosine Sine Decomposition] Let U $\in \mathbb{C}^{d\times d}$

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \text{ with } U_{ij} \in \mathbb{C}^{r_i \times c_j} \tag{17}$$

Then there exists a unitary $V_i \in \mathbb{C}^{r_i \times r_i}$ and $W_j \in \mathbb{C}^{c_j \times c_j}$ such that

$$\begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} = \begin{bmatrix} V_1 & \mathbf{0} \\ \mathbf{0} & V_2 \end{bmatrix} \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} W_1 & \mathbf{0} \\ \mathbf{0} & W_2 \end{bmatrix}^\dagger \tag{18}$$

Where D is a unitary matrix of the form

$$D = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & & & I & & \\ & C & & & S & \\ & & I & & & \mathbf{0} \\ I & & & \mathbf{0} & & \\ & S & & & -C & \\ & & \mathbf{0} & & & -I \end{bmatrix} \tag{19}$$

C and S have diagonal entries between 0 and 1, with the property $C^2 + S^2 = I$ for which the technique is named. Some of the $\mathbf{0}$ blocks may not be rectangular and the I blocks might be of different dimensions, the important part is that C and S have the same dimensions. This form of $\boldsymbol{D}$ leads to a decomposition of direct sums of 3 spaces [11].

$$D = \begin{bmatrix} \mathbf{0} & I \\ I & \mathbf{0} \end{bmatrix} \oplus \begin{bmatrix} C & S \\ S & -C \end{bmatrix} \oplus \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & -I \end{bmatrix} \tag{20}$$

Supposing all these are square we can decompose these blocks into the rotation matrix

$$\begin{bmatrix} \lambda_i & \sqrt{1-\lambda_i} \\ \sqrt{1-\lambda_i} & -\lambda_i \end{bmatrix} \tag{21}$$

Each λ_i is are the singular values of U_{11}. We can then see that the decomposition of each sub unitary is

$$U_{ij} = V_i D_{ij} W_j^\dagger \tag{22}$$

Briefly discussing about why D is of this form, we need to consider U_{11} if we let the SVD be $V_1 D_{11} W_1^\dagger$ then the form of D_{11} comes about from the fact that all the singular values of U must be less than 1. From here we can figure out the off diagonal elements by rearranging equation 22 to isolate D_{ij}. There are 4 facts that show why D takes this form; the columns of D are orthonormal, the diagonal elements must be positive, off diagonal blocks must be upper triangular, and $C^2 + S^2 = I$ must be obeyed. And finding D_{22} can be shown by exploiting the fact that D is unitary [**?**].

4 Function Evaluation with QSVT using Pennylane Scientific Package

In this section, we will put the theoretical knowledge we have gained into practice by utilizing Pennylane scientific package and its QSVT class and functionality. Pennylane is an open-source Python framework for quantum computing and quantum machine learning that was developed by Xanadu Quantum Technologies, Inc. After carefully examining the Pennylane package, we were able to apply the QSP and QSVT algorithms to perform a polynomial transformation on a scalar and a matrix. In this section, we will summarize our experiments and provide details on these experiments.

Upon literature overview and careful examinations of several tutorials on the subject of matter, we summarize below the QSP/QSVT procedure in several steps [3, 2, 1, 5, 8].

Step 1. Before running the QSP or the QSVT procedure the following conditions must be met:

- For the block encoding of a scalar, this scalar must be between -1 and 1, otherwise the re-scaling of the scalar is required.
- For the block encoding of a matrix, its singular values must not be larger than 1, otherwise the re-scaling of the matrix is required.
- For a matrix block encoding, it may have an arbitrary shape, not necessary a square shape.

Step 2. Upon satisfying conditions of **Step 1**, one runs a classical algorithm to solve for the Chebyshev coefficients in the Chebyshev expansion of a function.

Step 3. Using the coefficients of the Chebyshev expansion, one runs an optimization algorithm to find the sequence of phase angles.

Step 4. The computed phase angles are used as an input of the parameterized quantum gates to construct the QSP or the QSVT quantum circuit representing a transformation of a scalar or a matrix by the function rule.

Experiment 1. [*Polynomial fitting with the QSP*] The polynomial of the form $f(a) = 8a^4 - 8a^2 + 1$ has been approximated with the sequence of phase angles $\vec{\phi} = (-3\pi/2, \pi/2, \pi/2, \pi/2, \pi/22)$ analytically calculated using 3.1. We also pre-computed the phase angles using the pyqsp which utilizes the Laurent polynomials to optimize for phase angles and using the Pennylane native gradient-based optimization algorithm represented by adaptive gradient optimizer or by a stochastic gradient descent. While both methods were in disagreement with each other, we were able to obtain a suitable sequence of phase angles to approximate the polynomial with the relative error not exceeding 0.0102 ($\epsilon < 0.0102$). We notice that we had to re-run the optimization algorithms for several times until we obtained the suitable set of phase angles. That could be explained by the heuristic nature in the design of these optimization algorithms. These optimization algorithms may have a tendency sticking in local minimum. The result of the polynomial fitting is depicted on [Figure 1]. The code is attached in Appendix, Listing 1.

Remark. A reader can observe that in the experiment we used a different number of phase angles to find the same polynomial compared to the number of phase angles in the example of section 3.1. We should note that depending on the convention (W or O), one expects to have a different length of the sequence of the phase angles. For instance, the Pennylane currently supports only the W convention leading to the number of the phase angles being $(d + 1)$ for the d-th degree polynomial. Moreover, as we observed in several scientific papers [1, 4] and tutorials [5, 7] on the subject matter, for a more complex polynomial fitting with the QSP or the QSVT (such as matrix inversion problem in 4), the length of the sequence of phase angles strongly correlates with the choice of the classical optimization algorithms and the desired accuracy of the polynomial fitting, making it difficult to generalize an estimate on the length of the ϕ-sequence.

Experiment 2. [*The matrix inversion problem in solving the system of linear equations*]. In this experiment we will invert a matrix and solve a linear equation with it, we use the framework given in reference [5] on the pennylane website. In order to find the inverse of a matrix A we want to invert each of the singular values of A first. So using QSVT we approximate the function $s \cdot \frac{1}{x}$.

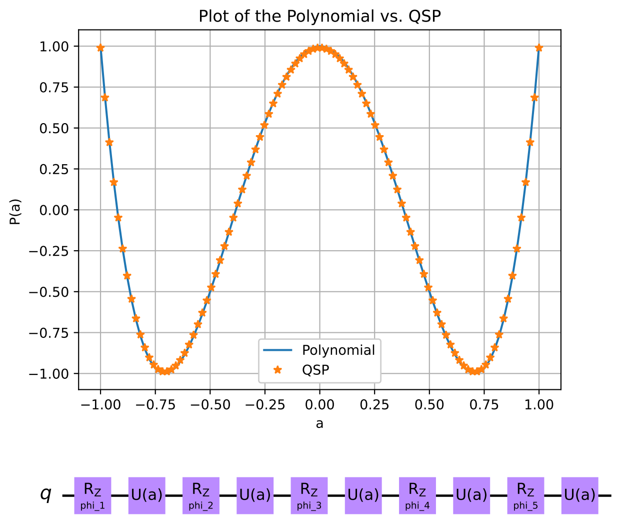

Figure 1: QSP polynomial fitting for the fourth degree polynomial $f(a) = 8a^4 - 8a^2 + 1$ and a schematic representation of the QSP quantum circuit

We include a scaling factor s here since QSVT values are bounded by [-1,1] and $\frac{1}{x}$ falls outside those bounds. We also assign a value for κ as to not reach any undefined values for $\frac{1}{x}$. One note here is if we use one QSVT circuit, the resulting function will have one polarity, so instead we sum together and even and odd circuit, giving a better approximation. Given the value of κ we can use pyqsp package to find s and phase angles to estimate our target function. The other method of obtaining phase angles is to use gradient based optimization, this process is long and costly. First we randomly generate phase angles, then using a mean square loss function we can use Pennylanes step and cost function over several iterations to adjust the phase angles, until a desired cost is hit. Once we have optimized our angles we are ready to solve the linear system, $A^{-1}|b\rangle = |x\rangle$. We prepare a normalized state $|b\rangle$, then we create the quantum circuit, the transform will have some imaginary part and we only want to apply the real part of our operator so we make the following estimation

$$U_{QSVT}(\phi) = \frac{1}{2}(U_{QSVT}(\phi) + U^*_{QSVT}(\phi)) \tag{23}$$

Finally we can apply use this quantum circuit to effectively apply A^{-1} to $|b\rangle$, so the state $|x\rangle$ should be the result. In our tests we were able to get a relative error $\epsilon < 0.0035$. The code we used can be seen in section 7.

5 Conclusion

The QSVT framework is a sophisticated approach to generalize various algorithms, including matrix function evaluation problems. QSP, a subset of QSVT, is particularly noteworthy in this regard. The theoretical overview of QSVT is based on the concept of dividing a larger system into smaller subsystems and then inducing a polynomial transformation on the singular values of these subsystems. The polynomial transformation is performed by applying signal rotation and signal processing operators interchangeably on the invariant subspaces of the larger space.

After conducting two computational experiments for the matrix function evaluation problem and 4-th degree polynomial fitting, it has become clear that there is a significant difference between theoretical predictions and practical implementation of the QSVT. In order to implement the QSVT procedure as a quantum circuit, the sequence of phase angles needs to be known in advance. However, calculating these phase angles requires classical optimization algorithms. Unfortunately, there are no standard algorithms available that can efficiently optimize a highly parameterized space. We have tested optimization algorithms based on the Laurent polynomials as well as optimization procedures available in the Pennylane scientific package. However, these algorithms are computationally fragile, and can easily get stuck in local minima. Additionally, they may rely on an initial guess of the phase angles, which makes them poorly convergent. We have also found that the Pennylane package can take a significant amount of time (up to $30 - 60$ minutes) to optimize for the sequence of the phase angles.

There is a lack of clear procedure to analyze the errors in the function evaluation problem since they occur at multiple stages of the algorithm such as Chebyshev expansion, phase angles optimization, and matrix block encoding. This makes it challenging to analyze the error on the QSVT algorithm, especially when dealing with quantum algorithms that may use QSVT procedure for several subroutines simultaneously (nested QSVT algorithms) or require high precision on the solution.

We made an assumption that it is possible to embed any matrix into a larger unitary matrix. However, in practice, we have not found any evidence of block encoding for arbitrary dense matrices. This limitation currently restricts the applications of QSVT to accessing only sparse matrices.

6 Acknowledgment

Team members are thankful to William Beatrez for a productive discussion about experimental part of the project.

References

[1] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics. *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2018.

[2] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. Grand unification of quantum algorithms. *PRX Quantum*, 2:040203, Dec 2021.

[3] Lin Lin. Lecture notes on quantum algorithms for scientific computation. Jan 2022.

[4] Yulong Dong, Xiang Meng, K. Birgitta Whaley, and Lin Lin. Efficient phase-factor evaluation in quantum signal processing. *Phys. Rev. A*, 103:042419, Apr 2021.

[5] Jay Soni and Jarrett Smalley. Qsvt in practice (tutorial). September 2023.

[6] Di Fang, Lin Lin, and Yu Tong. Time-marching based quantum solvers for time-dependent linear differential equations. *Quantum*, March 2023.

[7] Juan M. Arrazola. Intro to qsvt (tutorial). May 2023.

[8] Jay Soni. Function fitting using quantum signal processing (tutorial). May 2022.

[9] Caterina Fenu Francesca Arrigo, Michele Benzi. Computation of generalized matrix functions. Dec 2015.

[10] Ryan Holbrook. Math for machines (tutorial on ml).

[11] Kevin Tian Ewin Tang. A cosine-sine guide to the quantum singular value transformation. Feb 2023.

[12] Yulong Dong and Lin Lin. Random circuit block-encoded matrix and a proposal of quantum linpack benchmark. *Phys. Rev. A*, 103:062412, Jun 2021.

7 Appendix

```
def target_polynomial(a):
  return 8 * a ** 4 - 8 * a ** 2 + 1

#list of the precomputed phase angles
phase_angles = [-3*np.pi/2, np.pi/2, np.pi/2, np.pi/2, np.pi/22]

def qsvt_output(a):
    out = qml.matrix(qml.qsvt(a, phase_angles, wires=[0]))
    return out[0, 0]  # top-left entry

a_vals = np.linspace(-1, 1, 100)
qsvt = [np.real(qsvt_output(a)) for a in a_vals]
target = target_polynomial(a_vals)

# plotting polynomial approximation
plt.plot(a_vals, target, label="Polynomial")
plt.plot(a_vals, qsvt, "*", label="QSP")
plt.title('Plot␣of␣the␣Polynomial␣vs.␣QSP')
plt.xlabel('a')
plt.ylabel('P(a)␣')
plt.legend()
plt.grid(True)
plt.show()
```

Listing 1: Python code for the QSP polynomial approximation.

```

import pennylane as qml

kappa = 4
s = 0.10145775  # determined by pyqsp
np.random.seed(42)  # set seed
phi = np.random.rand(51)

def target_func(x):
    return s * (1 / x)

def sum_even_odd_circ(x, phi, ancilla_wire, wires):
    phi1, phi2 = phi[: len(phi) // 2], phi[len(phi) // 2:]

    qml.Hadamard(wires=ancilla_wire)  # equal superposition

    # apply even and odd polynomial approx
    qml.ctrl(qml.qsvt, control=(ancilla_wire,), control_values=(0,))(x,
        phi1, wires=wires)
    qml.ctrl(qml.qsvt, control=(ancilla_wire,), control_values=(1,))(x,
        phi2, wires=wires)

    qml.Hadamard(wires=ancilla_wire)


def loss_func(phi):
    sum_square_error = 0
    for x in samples_x:
        qsvt_matrix = qml.matrix(sum_even_odd_circ)(x, phi, ancilla_wire="
            ancilla", wires=[0])
        qsvt_val = qsvt_matrix[0, 0]
        sum_square_error += (np.real(qsvt_val) - target_func(x)) ** 2

    return sum_square_error / len(samples_x)


cost = 1
iter = 0
opt = qml.AdagradOptimizer(0.1)

while cost > 0.5e-4:
    iter += 1
    phi, cost = opt.step_and_cost(loss_func, phi)
    if iter % 10 == 0 or iter == 1:
        print(f"iter:␣{iter}")
    if iter > 100:
        break

def real_u(A, phi):
    qml.Hadamard(wires="ancilla1")

    qml.ctrl(sum_even_odd_circ, control=("ancilla1",), control_values=(0,)
        )(A, phi, "ancilla2", [0, 1, 2])
    qml.ctrl(qml.adjoint(sum_even_odd_circ), control=("ancilla1",),
        control_values=(1,))(A.T, phi, "ancilla2", [0, 1, 2])

    qml.Hadamard(wires="ancilla1")

A = np.array(
    [
        [0.65713691, -0.05349524, 0.08024556, -0.07242864],
        [-0.05349524, 0.65713691, -0.07242864, 0.08024556],
        [0.08024556, -0.07242864, 0.65713691, -0.05349524],
        [-0.07242864, 0.08024556, -0.05349524, 0.65713691],
    ]
)

b = np.array([1, 2, 3, 4], dtype="complex")
target_x = np.linalg.inv(A) @ b

norm_b = np.linalg.norm(b)
normalized_b = b / norm_b
norm_x = np.linalg.norm(target_x)
normalized_x = target_x / norm_x

@qml.qnode(qml.device("default.qubit", wires=["ancilla1", "ancilla2", 0,
    1, 2]))
def linear_system_solver_circuit(phi):
    qml.StatePrep(normalized_b, wires=[1, 2])
    real_u(A.T, phi)
    return qml.state()


transformed_state = linear_system_solver_circuit(phi)[:4]  # first 4
    entries of the state
rescaled_computed_x = transformed_state * norm_b / s
normalized_computed_x = rescaled_computed_x / np.linalg.norm(
    rescaled_computed_x)

print("target␣x:", np.round(normalized_x, 3))
print("computed␣x:", np.round(normalized_computed_x, 3))
```

Listing 2: Python code for solving linear equation.