

Quantum and quantum-inspired linear algebra

Ewin Tang Christopher Kang

July 24–28, 2023

0 Preface

These are the lecture notes for a 5-lecture mini-course I taught at the 2023 PCMI graduate summer school. Christopher Kang served as TA, and contributed to this document by helping create problem sets.¹ The abstract for the mini-course is as follows:

Abstract

An exciting recent line of work unifies many interesting quantum algorithms under a powerful linear algebraic framework known as “quantum singular value transformation” (QSVT). We’ll introduce this framework, build some tools in polynomial approximation that are helpful for applying it, and investigate what kinds of results it can achieve. Our goal will be to understand how and when to use QSVT in quantum algorithm design, and ultimately, whether it can reveal new quantum speedups for interesting problems in data analysis, machine learning, or quantum simulation.

I ended up covering the following topics. The first two lectures give a near-complete picture of the theory of QSVT as given by Gilyén, Su, Low, and Wiebe [GSLW19]. Here, I crib from my exposition of this with Kevin Tian [TT24]; our proofs are ultimately the same as the originals, but ours are clean enough that they can be reasonably presented in a lecture.

The third lecture covers polynomial approximation, the central mathematical tool appearing in QSVT applications. The material is taken from one of my favorite textbooks [Tre19], with the exposition again following mine and Kevin Tian’s [TT24], which gives versions of these results which are targeted towards users of QSVT.

The final two lectures discuss quantum-inspired algorithms, which is a theory of QSVT for classical algorithms which I worked on in my undergrad and PhD. From this theory, we can conclude that quantum linear algebra techniques don’t give exponential speedups for a broad range of classical machine learning tasks; but also, it illuminates the extent to which QSVT can work without a quantum computer. I cover ideas mainly from [CGLLTW22; BT24], which give the cleanest versions of this theory, but again I shamelessly take expositional choices from other writing I did on this topic [Tan22; Tan23].

For these notes, I’ll assume comfort with quantum computing basics (gates, circuits, measurement), linear algebra (unitary matrices, tensor products, singular value decomposition and eigendecomposition), and probability.

¹Update (April 21, 2026): Thanks to Gregory Rosenthal for suggesting many helpful edits to these notes!

Contents

0	Preface	1
1	Introducing the block-encoding	3
1.1	Block-encodings	4
1.2	Extensibility properties of block-encodings	5
1.3	The “fundamental theorem” of block-encodings	7
1.4	Wielding the block-encoding	8
	Problem Set 1: The block-encoding	10
2	Proving QSVT	11
2.1	Quantum signal processing (QSP)	11
2.2	Lifting with the CS decomposition	14
2.3	Proving QSVT	16
	Problem Set 2: The QSVT	19
3	Approximating many things by polynomials	20
3.1	Chebyshev polynomials and properties	20
3.2	Approximating functions from Chebyshev series	22
3.3	Lower bounds on polynomial approximation	25
	Problem Set 3: Polynomial approximation	27
4	Introducing quantum-inspired linear algebra	28
4.1	A vignette: The swap test, and what’s in an access model	29
4.2	Extensibility properties	32
	Problem Set 4: Dequantizing QSVT	35
5	Quantum-inspired algorithms: sketching and beyond	36
5.1	Another vignette: Tools for the matrix-vector product	36
5.2	Oversampling and query access to matrices	37
5.3	Sketching to estimate matrix products	38
5.4	General singular value transformation	40
	Problem Set 5: The power of classical	42

1 Introducing the block-encoding

First, some background behind the framework I’m about to introduce. Quantum computers were first envisioned for use in efficiently simulating quantum systems. One commonly studied instance of this task is *Hamiltonian simulation*, i.e. simulation of the dynamics of a closed quantum system.

Problem (Hamiltonian simulation). Let H be a Hamiltonian made up of m terms, meaning that

$$H = \sum_{a=1}^m \lambda_a E_a \text{ where } E_a \text{ is a tensor product of single-qubit unitary matrices.}$$

Given a quantum state $|\psi\rangle$ and a time $t \in \mathbb{R}$, produce a state close to $e^{-iHt} |\psi\rangle$. Or, slightly stronger: implement a unitary U close to e^{-iHt} , so that $\|U - e^{-iHt}\| \leq \varepsilon$, where $\|\cdot\|$ denotes operator norm.²

We describe H as a sum of individual interaction terms E_a , where λ_a dictates the strength of the interaction. These E_a ’s need not be as we describe them, but they are generally simple operators encoding basic kinds of interactions between qubits. Then, $e^{-iHt} |\psi\rangle$ is the state of the system after evolving according to the Schrödinger differential equation $\partial_t |\psi(t)\rangle = -iH |\psi(t)\rangle$ with initial state $|\psi(0)\rangle = |\psi\rangle$, for time t .

Original Hamiltonian simulation algorithms [Llo96] proceeded by using Trotter approximations: since the E_a ’s are “simple”, we would be happy if we can break up the exponential into pieces, each with only one term. We can be happy, since for r large enough,

$$e^{-iHt} \approx (e^{-iE_1 t/r} e^{-iE_2 t/r} \dots e^{-iE_m t/r})^r, \tag{1}$$

This method for Hamiltonian simulation is simple and easy-to-implement. However, it is far from optimal. For the moment, let’s ignore dependence on $\log(d)$, m , and t for the moment³ and focus on error ε . The approximation quality of Trotter is poor, since $r = \text{poly}(1/\varepsilon)$ is needed for the Trotter error to be ε ; so, quantum algorithms with this strategy only get $\text{poly}(1/\varepsilon)$ gate complexity. More refined “product formulas” than Eq. (1) are able to get sub-polynomial dependence on error [CW12]. However, the optimal algorithms for Hamiltonian simulation, which get $\log(1/\varepsilon)/\log \log(1/\varepsilon)$ gate complexity [BCKKS17; LC17; LC19], use an entirely different technique. This technique developed into the framework I will present now [GSLW19].

This framework proceeds by:

1. Defining a type of quantum circuit called a “block-encoding”;
2. Showing that, given λ_a and E_a , we can construct an efficient block-encoding of H , up to a scaling constant;

²And $\|\cdot\|$ denotes Euclidean norm when applied to a vector.

³What happens with these parameters is a story for another time. In short, it heavily depends on the Hamiltonian whether the algorithms have gate complexity $\log(d)mt$ or something more like $(\log(d)t)^{1+o(1)}$. Product formulas can typically get the right dependence on these parameters [CSTWZ21; CS19]. But for the sort of “physical” Hamiltonians that I’m familiar with, the algorithms with best dependence on all parameters (including ε) use the techniques here, along with exploiting some locality properties about the Hamiltonian [HHKL21].

3. Showing that we can get a block-encoding of (an approximation of) e^{-iHt} with few uses of the block-encoding to H ;
4. Using our new block-encoding to apply the approximation of e^{-iHt} to a state.

1.1 Block-encodings

We begin with the block-encoding, which is a way to access a matrix via a quantum circuit.

Definition 1.1 (Block-encoding, variant of [GSLW19, Definition 43], [Ral20, Definition 1]). Given $A \in \mathbb{C}^{r \times c}$, we say $U \in \mathbb{C}^{d \times d}$ is a Q -block encoding of A if U is implementable with $\mathcal{O}(Q)$ gates and

$$B_{L,1}^\dagger U B_{R,1} = A, \quad (2)$$

where $B_{L,1} \in \mathbb{C}^{d \times r}$, $B_{R,1} \in \mathbb{C}^{d \times c}$ are the first r and c columns of the identity matrix. Equivalently,

$$U = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}, \quad (3)$$

where \cdot denotes unspecified block matrices. We denote $\Pi_L = B_{L,1} B_{L,1}^\dagger$, $\Pi_R = B_{R,1} B_{R,1}^\dagger$ to be the corresponding projections onto the spans of $B_{L,1}$ and $B_{R,1}$, respectively.

We'll often consider d , r , and c as powers of two so that we can write everything in terms of qubits. The equation in Definition 1.1 then becomes

$$(\langle 0 |^{\otimes a_L} \otimes I) U (|0 \rangle^{\otimes a_R} \otimes I) = A \quad (4)$$

for $a_L = \log_2(d/r)$ and $a_R = \log_2(d/c)$.

Remark 1.2 (Other definitions of the block-encoding). In the literature, you'll often see block-encodings defined with an accuracy parameter ε and a rescaling parameter α , allowing for approximation:

$$\|A - \alpha B_{L,1}^\dagger U B_{R,1}\| \leq \varepsilon.$$

In these lecture notes, instead of saying we have an ε -accurate α -rescaled block-encoding of A , we'll say that we have a block-encoding of \hat{A}/α where \hat{A} satisfies $\|\hat{A} - A\| \leq \varepsilon$. This choice to not incorporate error into definition of block-encoding makes proving things easier in my experience. Definitions sometimes also allow $B_{L,1}$ and $B_{R,1}$ to be arbitrary isometries [GSLW19, Definition 11]; this is not any more general, since then $B_L^\dagger U B_R$ is a block-encoding in the sense above, where $B_L, B_R \in \mathbb{C}^{d \times d}$ are unitary completions of $B_{L,1}$ and $B_{R,1}$.

Remark 1.3 (Normalization). Because block-encoded matrices must be a sub-matrix of a unitary, it's only possible⁴ to produce block-encodings of matrices with bounded operator norm, $\|A\| \leq 1$. The actual scaling of A is important: it is generally easier to get a block-encoding of matrices with smaller operator norm, so one often ends up working with block-encodings of A/α for some large α ; but α then appears in the running time, meaning that smaller matrices produce slower algorithms. The mechanics of this aren't particularly obvious, but keep in mind that scale is an essential component to manage when working with block-encodings.

⁴In a later problem set, you'll see that is also a sufficient condition—any matrix with bounded operator norm can be the submatrix of a unitary matrix.

We can view the block-encoding as a generalization of a unitary quantum circuit.

Lemma 1.4. *A quantum circuit implementing the unitary U with Q gates is a Q -block encoding of U .*

In the way that we apply a circuit implementing U to perform the map $|\psi\rangle \mapsto U|\psi\rangle$, a block-encoding of A can be used to perform the map $|\psi\rangle \mapsto A|\psi\rangle$, with some chance of failure. This allows us to perform more general types of linear algebraic operations than what purely unitary circuits offer.

Lemma 1.5. *Given $U \in \mathbb{C}^{d \times d}$, a Q -block encoding of $A \in \mathbb{C}^{r \times c}$, and a state $|\psi\rangle \in \mathbb{C}^c$, there is a quantum circuit with $\mathcal{O}(Q)$ gates that produces the state $\frac{A|\psi\rangle}{\|A|\psi\rangle\|}$ with probability $\|A|\psi\rangle\|^2$.*

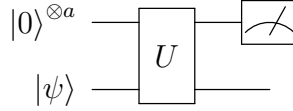


Figure 1: A basic block-encoding circuit. If U is a block-encoding of the matrix $A \in \mathbb{C}^{r \times r}$, then provided the outcome of the measurement on the first wire is $|0\rangle^{\otimes a}$, then the output of the circuit is $A|\psi\rangle$.

Proof. The circuit is shown in Fig. 1: we can take the state $|\psi\rangle$ and add a_R qubits initialized to $|0\rangle$. Then, we apply the block-encoding U and measure the first a_L qubits. If they all have outcome 0, then by Eq. (4), the resulting state is $A|\psi\rangle$. This occurs with probability $\|A|\psi\rangle\|^2$. \square

1.2 Extensibility properties of block-encodings

The question now becomes: when can we produce an efficient block-encoding of a matrix? In fact, we can reduce Hamiltonian simulation to such a problem: a tensor product of single-qubit unitary matrices is implementable by a layer of single-qubit gates, and so has a $\log(d)$ -block encoding by Lemma 1.4. So, we have block encodings of the terms $\{E_a\}_{a \in [m]}$ for the Hamiltonian $H = \sum_{a=1}^m \lambda_a E_a$; can we get a block-encoding of (an approximation of) e^{-iHt} ?

Block-encodings enjoy several *extensibility properties*: that is, given block-encodings of A and B , we can get block-encodings of AB and $c_0A + c_1B$, whenever (1) A and B have dimensions such that these expressions make sense and (2) the scale of c_0 and c_1 is sufficiently small. This will allow us to get a block-encoding of H/α for some rescaling constant α .

Lemma 1.6 (Multiplying block-encodings). *Let U and V be Q_U - and Q_V -block-encodings of $A \in \mathbb{C}^{r \times s}$ and $B \in \mathbb{C}^{s \times t}$, respectively. Then we can construct a $(Q_U + Q_V)$ -block encoding of AB .*

Proof. The circuit implementing AB is shown in Fig. 2. We can see that this is a block-encoding of AB by inspection, as this is a composition of two of the circuits in Fig. 1. \square

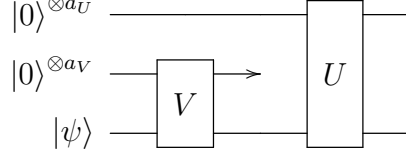


Figure 2: If U is a block-encoding of A and V is a block-encoding of B , then this circuit is a block-encoding of AB , shown being applied to input $|\psi\rangle$. Here, a_U and a_V are the padding needed for the respective block-encodings.

We can construct block-encodings of linear combinations of block-encodings using the *Linear Combination of Unitaries* (LCU) algorithm.

Lemma 1.7 (Summing block-encodings). *Let $U^{(i)}$ be a $Q^{(i)}$ -block-encoding of $A^{(i)} \in \mathbb{C}^{r \times c}$ for all $i = 0, \dots, k-1$. Then we can construct a $(k + \sum_{i=0}^{k-1} Q^{(i)})$ -block encoding of $\sum \alpha_i U^{(i)}$, for $\alpha_i \in \mathbb{C}$ such that $\sum |\alpha_i| \leq 1$.*

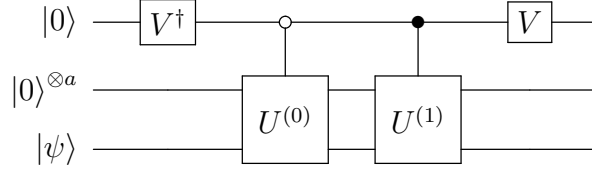


Figure 3: If $U^{(1)}$ and $U^{(2)}$ are block-encodings of $A^{(1)}$ and $A^{(2)}$, then this circuit is a block-encoding of $|V_{0,0}|^2 A^{(0)} + |V_{0,1}|^2 A^{(1)}$, shown being applied to input $|\psi\rangle$. Here, the gate blocks containing $U^{(0)}$ and $U^{(1)}$ denote conditioning on $|0\rangle$ and conditioning on $|1\rangle$.

Proof. First, consider taking the linear combination of $k = 2$ block-encodings. The circuit implementing a linear combination is shown in Fig. 3. The controlled- $U^{(0)}$ and controlled- $U^{(1)}$ apply the unitary

$$\begin{pmatrix} U^{(0)} & \\ & I \end{pmatrix} \begin{pmatrix} I & \\ & U^{(1)} \end{pmatrix} = \underbrace{\begin{pmatrix} U^{(0)} & \\ & U^{(1)} \end{pmatrix}}_{(|0\rangle\langle 0| \otimes U^{(0)} + |1\rangle\langle 1| \otimes U^{(1)})} \quad (5)$$

So, the full circuit is performing

$$\underbrace{\begin{pmatrix} V_{0,0}I & V_{0,1}I \\ V_{1,0}I & V_{1,1}I \end{pmatrix}^\dagger \begin{pmatrix} U^{(0)} & \\ & U^{(1)} \end{pmatrix} \begin{pmatrix} V_{0,0}I & V_{0,1}I \\ V_{1,0}I & V_{1,1}I \end{pmatrix}}_{(V^\dagger|0\rangle\langle 0|V) \otimes U^{(0)} + (V^\dagger|1\rangle\langle 1|V) \otimes U^{(1)}} \quad (6)$$

The top-right corner of this matrix, which is where the block-encoding should be, equals

$$|V_{0,0}|^2 U^{(0)} + |V_{1,0}|^2 U^{(1)}.$$

So, for any non-negative real α_0, α_1 summing to one, we can find some one-qubit unitary V whose first column is $\sqrt{\alpha_0}, \sqrt{\alpha_1}$, giving the desired block-encoding. If, say, α_0 was negative, we could use the circuit for $|\alpha_0|$, but use a controlled unitary of $-U^{(0)}$ instead of $U^{(0)}$ to negate it in the block-encoding.

The general version is of the following form. First, if U is a block-encoding of A then so is $I \otimes U$, so without loss we can pad the dimension until all $U^{(i)}$'s are all the same size, $d \times d$. Second, without loss we can pad our linear combination until k is a power of two by adding $U^{(i)} = I$ and $\alpha_i = 0$ to the linear combination. Let $V \in \mathbb{C}^{k \times k}$ be a unitary such that

$$V |0\rangle = \sum_{i=0}^k \sqrt{|\alpha_i|} |i\rangle$$

and let $U \in \mathbb{C}^{kd \times kd}$ be the unitary

$$\sum_{i=0}^{k-1} (|k\rangle \langle k|) \otimes \left(\frac{\alpha_i}{|\alpha_i|} U^{(i)} \right).$$

Then $(V^\dagger \otimes I)U(V \otimes I)$ is a block-encoding of $\sum \alpha_i U^{(i)}$. The cost of applying V is $\mathcal{O}(k)$, and assuming that the cost of applying the controlled version of $U^{(i)}$ is only a constant factor larger than the cost of applying $U^{(i)}$ itself, the cost of U is $\mathcal{O}(k + \sum_i Q^{(i)})$. \square

We will need a final assertion about block-encodings. So far, we haven't specified a particular gate model for our quantum circuit; if you like, it'll be easiest to consider quantum circuits with arbitrary one- and two-qubit gates. Whatever gate model is used, we need that the number of gates to apply a circuit is equal to the number of gates to apply its inverse (up to constant factors).

Lemma 1.8 (Taking the conjugate transpose of a block-encoding). *If U is a Q -block encoding of A , then U^\dagger is a Q -block encoding of A^\dagger .*

1.3 The “fundamental theorem” of block-encodings

These extensibility theorems are powerful: one might notice that we can combine them to get block-encodings of polynomials of A . “Polynomials of A ” has a clear meaning when A is Hermitian: for a function $f : \mathbb{R} \rightarrow \mathbb{C}$, $f(A)$ is defined to be the function that applies f to the eigenvalues of A : for $A = \sum \lambda_i u^{(i)} (u^{(i)})^\dagger$ the unitary eigendecomposition of A , $f(A) = \sum f(\lambda_i) u^{(i)} (u^{(i)})^\dagger$. But for general A , we need to be more careful. Now, we define the notion of applying a scalar function to a matrix, referred to in this literature as ‘singular value transformation’.

Definition 1.9 (Singular value transformation, [GSLW19, Definition 16]). Let $f : \mathbb{R} \rightarrow \mathbb{C}$ be even or odd,⁵ and let $A \in \mathbb{C}^{r \times c}$ have SVD $A = \sum_{i \in [\min(r,c)]} \sigma_i u_i v_i^\dagger$ (in particular, $\{u_i\}_{i \in [r]}$ and $\{v_i\}_{i \in [c]}$ are collections of orthonormal vectors). Then we define

$$f^{(\text{SV})}(A) = \begin{cases} \sum_{i \in [\min(r,c)]} f(\sigma_i) u_i v_i^\dagger & f \text{ is odd} \\ \sum_{i \in [c]} f(\sigma_i) v_i v_i^\dagger & f \text{ is even} \end{cases}$$

where σ_i is defined to be zero for $i > \min(r, c)$.

⁵A function is even if $f(x) = f(-x)$ and odd if $f(x) = -f(-x)$.

When $f(x) = p(x)$ is an even or odd polynomial, $p^{(\text{SV})}(A)$ can be written as a polynomial in the expected way, e.g. if $p(x) = x^2 + 1$, $p^{(\text{SV})}(A) = A^\dagger A + I$ and if $p(x) = x^3 + x$, $p^{(\text{SV})}(A) = AA^\dagger A + A$. Notice that the monomials alternate A and A^\dagger so that the dimensions align.

Now, we can consider when it's possible to convert a block-encoding of A to a block-encoding of $f^{(\text{SV})}(A)$, focusing on polynomials in particular.

Definition 1.10 (Achievable polynomial). A degree- n polynomial $p \in \mathbb{C}[x]$ is “achievable” if there is an explicit way to create a block-encoding of $p^{(\text{SV})}(A)$ from a block-encoding of A .

This definition isn't entirely formal (we technically need some efficiency guarantee on the block-encoding of $p^{(\text{SV})}(A)$), but will be useful language for us going forward. The extensibility properties directly show that some polynomials are achievable.

Corollary 1.11 (Corollary of the extensibility properties). *Polynomials of the form $p(x) = \sum_{k=0}^n a_k x^k$ are achievable, provided that $\sum |a_k| \leq 1$ and p is odd or even.*

A proof of this is deferred to the problem set. However, this is a very small class of polynomials: again see the problem set for an example of a limitation of this class of polynomials.

The main result of the block-encoding framework is that all bounded polynomials with real coefficients are achievable; this result gives QSVT, quantum singular value transformation, its name.

Theorem 1.12 (Taking polynomials of block-encodings [GSLW19, Theorem 17 and Corollary 18]). *If a polynomial with real coefficients $p \in \mathbb{R}[x]$ is even or odd and satisfies $|p(x)| \leq 1$ for all $x \in [-1, 1]$, then it is achievable.*

This result is about the best we could hope for. The constraint that $p(x)$ is bounded by 1 is necessary, since the QSVT algorithm converts a block-encoding of A to a block-encoding of $p^{(\text{SV})}(A)$ in a black-box way, i.e. without looking at A . But if some $p(x)$ had magnitude larger than one, then $p^{(\text{SV})}(A)$ can have norm greater than one, and therefore never be in a block-encoding. The even/odd constraint is not very important, since the polynomial is only ever applied to singular values, which are non-negative. The only limitation of this result is that it only applies to polynomials with real coefficients, but I don't know of situations where this is of concern.

We'll prove Theorem 1.12 in the next lecture. First, we see how to apply it.

1.4 Wielding the block-encoding

Hamiltonian simulation gives a nice view into how to use block-encodings and QSVT. As we discussed before, we can construct a $m \log(d)$ -block-encoding of $H/(\sum |\lambda_i|)$. Without loss of generality, we can rescale $H \leftarrow H/(\sum |\lambda_i|)$ and $t \leftarrow t \sum |\lambda_i|$ so that our block-encoding is of H . Our goal is to get an (approximate) block-encoding of $f(H)$, where

$$f(x) = \exp(-ixt) = \cos(tx) - i \sin(tx). \quad (7)$$

Since $\cos(tx)$ and $\sin(tx)$ are bounded even and odd functions, respectively, we can find good polynomial approximations of them. (We will see how to do this in lecture 3.) That is, we can find c and s such that, for all $x \in [-1, 1]$,

$$|c(x) - \cos(tx)| \leq \varepsilon \quad |s(x) - \sin(tx)| \leq \varepsilon$$

By Theorem 1.12, we can get block-encodings of $c^{(\text{SV})}(H)$ and $s^{(\text{SV})}(H)$. By Lemma 1.7, we can get a block-encoding of $\frac{1}{2}(c^{(\text{SV})}(H) - is^{(\text{SV})}(H)) \approx \frac{1}{2}e^{-iHt}$. This is enough if we wish to apply it to an input state $|\psi\rangle$, but to decrease the failure probability we can remove the $\frac{1}{2}$ through *oblivious amplitude amplification*, which can be done with QSVT.

I haven't yet discussed gate complexity or the error analysis, but we will see that the running time of the whole algorithm is dictated by how small the degree can be of polynomials approximating $\cos(tx)$ and $\sin(tx)$. Up to constant factors of wiggle room in the parameters, the number of times one needs to apply the block-encoding for H is equal to this degree, and we get optimal algorithms for Hamiltonian simulation by choosing the optimal polynomial approximations.

Problem Set 1: The block-encoding

Problem 1.1 (Taking tensor products of block-encodings). Let U and V be Q -block encodings of A and B , respectively. Show how to get a Q -block-encoding of $A \otimes B$.

Problem 1.2 (Extensibility properties). Prove Corollary 1.11 of the lecture notes. Specifically, show that the two extensibility properties allow us to convert a Q -block encoding of A to a nQ -block encoding of $p^{(\text{SV})}(A)$.

Problem 1.3 (Extensibility properties do not suffice). Let $p(x) = \sum_{k=0}^n a_k x^k$ be a polynomial whose coefficients satisfy $\sum |a_k| \leq 1$. Show that $p(x)$ cannot approximate $\sin(100x)$ for any choice of n . That is, show that there is some $x \in [-1, 1]$ such that

$$|p(x) - \sin(100x)| \geq 0.01.$$

We will see in Lecture 3 that $\sin(100x)$ can in fact be approximated by a low-degree polynomial; it's just that this class of polynomials doesn't suffice.

Problem 1.4 (Oblivious amplitude amplification). QSVT is a unifying technique which includes many major quantum algorithms, including amplitude amplification [MRTC21]. In this problem, we show that Oblivious Amplitude Amplification (OAA), as described in [BCCKS17, Lemma 3.6], can be written in our block-encoding framework.

Identify the block-encoding within the aforementioned unitary. What polynomial would effect the same transformation as described in [BCCKS17, Lemma 3.6]?

Remark 1.13. See [Ral20] for more information on how to get block-encodings of density matrices and observables, and how to use this to estimate physical quantities like expectations of Gibbs states. See [BCCKS17] for further discussion of Hamiltonian simulation, placing it in the context of the more general problem of understanding the “fractional query model”, “discrete query model”, and “continuous query model”. See [LC19] (the original paper) or [GSLW19] for a more thorough explanation of the Hamiltonian simulation algorithm.

2 Proving QSVT

Let's start by recalling the definition of a block-encoding from the previous lecture.

Definition 1.1 (Block-encoding, variant of [GSLW19, Definition 43], [Ral20, Definition 1]). Given $A \in \mathbb{C}^{r \times c}$, we say $U \in \mathbb{C}^{d \times d}$ is a Q -block encoding of A if U is implementable with $\mathcal{O}(Q)$ gates and

$$B_{L,1}^\dagger U B_{R,1} = A, \quad (2)$$

where $B_{L,1} \in \mathbb{C}^{d \times r}$, $B_{R,1} \in \mathbb{C}^{d \times c}$ are the first r and c columns of the identity matrix. Equivalently,

$$U = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}, \quad (3)$$

where \cdot denotes unspecified block matrices. We denote $\Pi_L = B_{L,1} B_{L,1}^\dagger$, $\Pi_R = B_{R,1} B_{R,1}^\dagger$ to be the corresponding projections onto the spans of $B_{L,1}$ and $B_{R,1}$, respectively.

In this lecture, we will prove Theorem 1.12, which states that all bounded polynomials with real coefficients are achievable, in the sense defined in Definition 1.10. The full statement, including quantitative bounds, are as follows.

Theorem 2.1 ([GSLW19, Theorem 17 and Corollary 18]). *If a degree- n polynomial with real coefficients $p \in \mathbb{R}[x]$ is even or odd and satisfies $|p(x)| \leq 1$ for all $x \in [-1, 1]$, then we can use a Q -block encoding of A to construct a $n(\log(d) + Q)$ -block encoding of $p^{(\text{SV})}(A)$.*

Our proof proceeds as follows. We begin with the case where A is a scalar and $U \in \mathbb{C}^{2 \times 2}$; this is known as quantum signal processing. Then, we show that the circuit used for the scalar case “lifts” to the matrix case; to do this, we use an argument with block matrices.

2.1 Quantum signal processing (QSP)

The idea of QSP is that we can perform a *known* function on an *unknown* (parametrized) operator by interleaving the unknown operator with rotations.

Definition 2.2 (Quantum signal processing). For a sequence of phase factors $\Phi = \{\phi_j\} \in \mathbb{R}^{n+1}$, it defines a *quantum signal processing* circuit⁶

$$\text{QSP}(\Phi, x) := \left(\prod_{j=1}^n \underbrace{\begin{pmatrix} e^{i\phi_j} & 0 \\ 0 & e^{-i\phi_j} \end{pmatrix}}_{e^{i\phi_j \sigma_z}} \underbrace{\begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix}}_{=:R(x)} \right) \begin{pmatrix} e^{i\phi_0} & 0 \\ 0 & e^{-i\phi_0} \end{pmatrix}. \quad (8)$$

Here, the product goes from n on the left-hand side to 1 on the right-hand side. The matrix $\sigma_z = \begin{pmatrix} 1 & \\ & -1 \end{pmatrix}$ is the Pauli Z matrix.

⁶We define QSP with the reflection operation $R(x)$; a different convention is to use the rotation $e^{i \arccos(x) \sigma_x} = \begin{pmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{pmatrix}$, denoted $W(x)$ in [GSLW19]. These two types of circuits are equivalent up to a shift in phase factors [MRTC21, Appendix A.2]. See the problem set for more discussion of this. Using $W(x)$ is perhaps more natural, since then this corresponds to alternating rotations in the σ_X and σ_Z basis.

Definition 2.3 (QSP-achievable polynomial [GSLW19, Corollary 8]). We say that a polynomial $p(x) \in \mathbb{C}[x]$ is *QSP-achievable* if there is a sequence of phase factors $\Phi = \{\phi_j\} \in \mathbb{R}^{n+1}$ such that

$$\mathbf{QSP}(\Phi, x) = \begin{pmatrix} p(x) & \cdot \\ \cdot & \cdot \end{pmatrix}. \quad (9)$$

To find out what polynomials are QSP-achievable, we first take a look at what the form of QSP is. It turns out that we can express it as a recurrence of polynomials.

Lemma 2.4 (QSP as a recurrence). *For some phase factors $\Phi = \{\phi_j\} \in \mathbb{R}^{n+1}$,*

$$\mathbf{QSP}(\{\phi_j\}_{0 \leq j \leq k}, x) = \begin{pmatrix} p_k(x) & \overline{q_k(-x)}\sqrt{1-x^2} \\ q_k(x)\sqrt{1-x^2} & \overline{p_k(-x)} \end{pmatrix}, \quad (10)$$

where $p_k(x)$ and $q_k(x)$ satisfy the following recurrence relation:

$$p_{k+1}(x) = e^{i\phi_{k+1}}(xp_k(x) + (1-x^2)q_k(x)); \quad (11)$$

$$q_{k+1}(x) = e^{-i\phi_{k+1}}(p_k(x) - xq_k(x)). \quad (12)$$

For the base case, $p_0(x) = e^{i\phi_0}$ and $q_0(x) = 0$.

Proof. The base case is because

$$\mathbf{QSP}(\{\phi_0\}, x) = \begin{pmatrix} e^{i\phi_0} & \\ & e^{-i\phi_0} \end{pmatrix} \quad (13)$$

For the inductive case, we just do the annoying computation.

$$\mathbf{QSP}(\{\phi_j\}_{0 \leq j \leq k+1}, x) \quad (14)$$

$$= e^{i\phi_{k+1}\sigma_z} R(x) \cdot \mathbf{QSP}(\{\phi_j\}_{0 \leq j \leq k}, x) \quad (15)$$

$$= \begin{pmatrix} e^{i\phi_{k+1}}x & e^{i\phi_{k+1}}\sqrt{1-x^2} \\ e^{-i\phi_{k+1}}\sqrt{1-x^2} & -e^{-i\phi_{k+1}}x \end{pmatrix} \begin{pmatrix} p_k(x) & \overline{q_k(-x)}\sqrt{1-x^2} \\ q_k(x)\sqrt{1-x^2} & \overline{p_k(-x)} \end{pmatrix} \quad (16)$$

$$= \begin{pmatrix} e^{i\phi_{k+1}}(xp_k(x) + (1-x^2)q_k(x)) & e^{i\phi_{k+1}}(\overline{p_k(-x)} + x\overline{q_k(-x)})\sqrt{1-x^2} \\ e^{-i\phi_{k+1}}(p_k(x) - xq_k(x))\sqrt{1-x^2} & e^{-i\phi_{k+1}}(-x\overline{p_k(-x)} + (1-x^2)\overline{q_k(-x)}) \end{pmatrix} \quad (17)$$

$$= \begin{pmatrix} p_{k+1}(x) & \overline{q_{k+1}(-x)}\sqrt{1-x^2} \\ q_{k+1}(x)\sqrt{1-x^2} & \overline{p_{k+1}(-x)} \end{pmatrix} \quad (18)$$

Feel free to stare at the last line for a little bit to confirm that the entries indeed all match up to what I claim them to be. \square

With this recurrence, we can give a characterization of which polynomials are QSP-achievable.

Theorem 2.5 ([GSLW19, Theorem 3]). *A degree- n polynomial $p(x) \in \mathbb{C}[x]$ is QSP-achievable with some $\Phi \in \mathbb{R}^{n+1}$ if and only if there is some polynomial $q(x)$ such that:*

- (a) q has degree $\leq n-1$;
- (b) (p, q) are (even, odd) or (odd, even);
- (c) $|p(x)|^2 + (1-x^2)|q(x)|^2 = 1$ for all x .

Proof. First, we consider the “only if” direction. Suppose $p(x)$ is QSP-achievable with the phase factors $\Phi \in \mathbb{R}^{n+1}$. Then, by Lemma 2.4, there is some $q(x)$ such that

$$\mathbf{QSP}(\Phi, x) = \begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix},$$

derived from the recurrence described in that lemma. From this recurrence, we can verify that at all times, conditions (a) and (b) are satisfied. Finally, condition (c) is always satisfied because $\mathbf{QSP}(\Phi, x)$ is a product of unitary matrices, and so is unitary: the first column having norm one is equivalent to $|p(x)|^2 + (1-x^2)|q(x)|^2 = p(x)\bar{p}(x) + (1-x^2)q(x)\bar{q}(x) = 1$, and this argument works for every $x \in [-1, 1]$. Because it holds for infinitely many x , the equality holds as polynomials.

Second, we consider the “if” direction. Suppose we have some $p(x)$ of degree n and $q(x)$ satisfying (a), (b), and (c). We want to construct phase factors that implement $p(x)$. We proceed by induction: when $n = 0$, this means that $p(x)$ is scalar and $q(x)$ has degree ≤ -1 (meaning it must be zero). Thus, $p(x) \equiv e^{i\phi}$ for some ϕ ; we can implement this with $\Phi = \{\phi\}$. For the inductive step, consider $p(x)$ of degree $n + 1$. If we could show that there exists some φ such that

$$(e^{i\varphi\sigma_z} R(x))^\dagger \begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix} = \begin{pmatrix} p_\downarrow(x) & \bar{q}_\downarrow(-x)\sqrt{1-x^2} \\ q_\downarrow(x)\sqrt{1-x^2} & \bar{p}_\downarrow(-x) \end{pmatrix} \quad (19)$$

for $p_\downarrow, q_\downarrow$ some even or odd polynomials with degree one lower than p and q , then we would be done. By assumption, the matrices on the left-hand side of Eq. (19) are unitary, so the right-hand side matrix is also unitary. Thus, p_\downarrow and q_\downarrow satisfy all the properties of the induction hypothesis, and there are phase factors $\{\phi_0, \dots, \phi_n\} \in \mathbb{R}^{n+1}$ giving the equality

$$(e^{i\varphi\sigma_z} R(x))^\dagger \begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix} = \mathbf{QSP}(\{\phi_0, \dots, \phi_n\}, x). \quad (20)$$

$$\begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix} = \mathbf{QSP}(\{\phi_0, \dots, \phi_n, \varphi\}, x) \quad (21)$$

So it comes down to finding the right value of φ that could remove a degree from p and q in Eq. (19). By properties (a) and (b), we can write

$$p(x) = a_{n+1}x^{n+1} + a_{n-1}x^{n-1} + \dots \quad (22)$$

$$q(x) = b_n x^n + b_{n-2}x^{n-2} + \dots \quad (23)$$

The condition (c) implies that $|a_{n+1}| = |b_n|$. Now, let's do the annoying matrix calculation we were putting off. Since $R(x)$ is its own inverse, $(e^{i\varphi\sigma_z} R(x))^\dagger = R(x)e^{-i\varphi\sigma_z}$, so

$$(e^{i\varphi\sigma_z} R(x))^\dagger \begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix} \quad (24)$$

$$= \begin{pmatrix} e^{-i\varphi}x & e^{i\varphi}\sqrt{1-x^2} \\ e^{-i\varphi}\sqrt{1-x^2} & -e^{i\varphi}x \end{pmatrix} \begin{pmatrix} p(x) & \bar{q}(-x)\sqrt{1-x^2} \\ q(x)\sqrt{1-x^2} & \bar{p}(-x) \end{pmatrix} \quad (25)$$

$$= \begin{pmatrix} e^{-i\varphi}xp(x) + e^{i\varphi}(1-x^2)q(x) & (e^{i\varphi}\bar{p}(-x) + e^{-i\varphi}x\bar{q}(-x))\sqrt{1-x^2} \\ (e^{-i\varphi}p(x) - e^{i\varphi}xq(x))\sqrt{1-x^2} & -e^{i\varphi}x\bar{p}(-x) + e^{-i\varphi}(1-x^2)\bar{q}(-x) \end{pmatrix} \quad (26)$$

So, we need the following polynomials to have lower degree:

$$p_\downarrow(x) = e^{-i\varphi}xp(x) + e^{i\varphi}(1-x^2)q(x) \quad (27)$$

$$q_\downarrow(x) = e^{-i\varphi}p(x) - e^{i\varphi}xq(x) \quad (28)$$

The “leading” coefficient of x^{n+2} for p_\downarrow and x^{n+1} for q_\downarrow are the same: $e^{-i\varphi}a_{n+1} - e^{i\varphi}b_n$. If we choose φ such that $e^{i\varphi} = \sqrt{a_{n+1}/b_n}$, then this coefficient is 0, and so the degrees of p_\downarrow and q_\downarrow are $\leq n - 1$ and $\leq n - 2$, as desired. \square

The above characterization of when a polynomial is QSP-achievable is still quite difficult to understand. With some more work, we can arrive at a clearer understanding of QSP-achievability, if we give up the imaginary degree of freedom in our polynomials.

Theorem 2.6 ([GSLW19, Theorem 5, Lemma 6]). *Let $p_{\text{Re}}(x)$ be a degree- n real-valued polynomial. Then there exists a degree- n $p \in \mathbb{C}[x]$ such that p is QSP-achievable and $p_{\text{Re}} = \text{Re}(p)$ if and only if*

- (a) p_{Re} is even or odd;
- (b) $|p_{\text{Re}}(x)| \leq 1$ for $x \in [-1, 1]$.

What’s happening here is that if we have a real polynomial where the “unit norm” constraint is merely an inequality, then we can add imaginary components to make it an equality, so that by Theorem 2.5 these supplemented polynomials are achievable. This is the only part of the QSVT proof which I’ll punt on: see [GSLW19] or [TT24, Appendix A] for a proof, which involves manipulating roots of these polynomials.

From this, we can get our desired block-encodings, at least in this scalar case. For some even or odd $p(x) \in \mathbb{R}[x]$, by Theorem 2.6, we can find a phase sequence Φ such that $\mathbf{QSP}(\Phi, x)$ has $p(x) + ip_{\text{Im}}(x)$ in the top-left corner for some $p_{\text{Im}}(x) \in \mathbb{R}[x]$. Then $\mathbf{QSP}(-\Phi, x)$ has $p(x) - ip_{\text{Im}}(x)$ in its top-left corner. So, using LCU, we can average these two to get a block-encoding of $p(x)$. This gives a proof of Theorem 2.1 in the scalar case.

2.2 Lifting with the CS decomposition

To generalize to higher dimensions, we need a new version of QSP (Definition 2.2). In this discussion, we follow the exposition of [TT24].

Definition 2.7 ([GSLW19, Definition 15]). The *phased alternating sequence* associated with a block-encoding U (following notation of Definition 1.1) and $\Phi = \{\phi_j\}_{0 \leq j \leq n} \in \mathbb{R}^{n+1}$ is

$$U_\Phi := \begin{cases} e^{i\phi_n(2\Pi_L - I)} U e^{i\phi_{n-1}(2\Pi_R - I)} \prod_{j=0}^{\frac{n-3}{2}} U^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} U e^{i\phi_{2j}(2\Pi_R - I)} & \text{if } n \text{ is odd, and} \\ e^{i\phi_n(2\Pi_R - I)} \prod_{j=0}^{\frac{n-2}{2}} U^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} U e^{i\phi_{2j}(2\Pi_R - I)} & \text{if } n \text{ is even.} \end{cases}$$

Remark 2.8. The phased alternating sequence U_Φ can be seen as a generalization of the quantum signal processing circuit $\mathbf{QSP}(\Phi, x)$. When $d = 2$ and $r = c = 1$, $2\Pi_L - I = 2\Pi_R - I = \sigma_z$, so thinking about $R(x)$ as a block-encoding of x ,

$$\mathbf{QSP}(\Phi, x) = [R(x)]_\Phi \text{ where } R(x) = \begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix}.$$

Theorem 2.9 ([GSLW19, Theorem 17]). *Let unitary $U \in \mathbb{C}^{d \times d}$ be a Q -block encoding of A . Suppose $\Phi = \{\phi_j\}_{0 \leq j \leq n} \in \mathbb{R}^{n+1}$ is such that $\mathbf{QSP}(\Phi, x)$ computes the degree- n polynomial $p(x) \in \mathbb{C}[x]$, as in Definition 2.2. Then, U_Φ is a $n(\log(d) + Q)$ -block encoding of $p^{(\text{SV})}(A)$.*

This theorem implies Theorem 2.1: by Theorem 2.6, for any real polynomial p , there is a polynomial q which is QSP-achievable and whose real part is p . Our theorem shows that q is achievable, and therefore its complex conjugate \bar{q} is as well; therefore, $\frac{q+\bar{q}}{2} = p$ is achievable. This does not inflate the cost of the block-encoding by more than a constant.

Now, we prove Theorem 2.9. We will see in the problem set that the rotations in U_Φ can be done in $\mathcal{O}(\log(d))$ gates, and so the gate complexity of U_Φ is indeed $\mathcal{O}(n(\log(d) + Q))$. All that remains is to show that it is indeed a block-encoding of $p^{(\text{SV})}(A)$.

We begin our proof by introducing the CS decomposition (CSD), a decomposition of a partitioned unitary matrix, following Paige and Wei [PW94]. The main idea of the CSD is that when a unitary matrix U is split into two-by-two blocks U_{ij} for $i, j \in \{1, 2\}$, one can produce “simultaneous singular value decompositions (SVDs)” of the blocks, of the form $U_{ij} = V_i D_{ij} W_j^\dagger$.⁷

Theorem 2.10 (The cosine-sine decomposition [TT24, Theorem 1]). *Let $U \in \mathbb{C}^{d \times d}$ be a unitary matrix, partitioned into blocks of size $\{r_1, r_2\} \times \{c_1, c_2\}$:*

$$U = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix}, \text{ where } U_{ij} \in \mathbb{C}^{r_i \times c_j} \text{ for } i, j \in \{1, 2\}.$$

Then, there exists unitary $V_i \in \mathbb{C}^{r_i \times r_i}$ and $W_j \in \mathbb{C}^{c_j \times c_j}$ for $i, j \in \{1, 2\}$ such that

$$\begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix} = \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} \begin{pmatrix} W_1 & \\ & W_2 \end{pmatrix}^\dagger,$$

where blanks represent zero matrices and $D_{ij} \in \mathbb{R}^{r_i \times c_j}$ are diagonal matrices, possibly padded with zero rows or columns. Specifically, we can write

$$D := \begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix} = \left(\begin{array}{cc|cc} 0 & & I & \\ & C & & S \\ \hline & & I & 0 \\ I & & & \\ & S & & -C \\ & & 0 & -I \end{array} \right) \quad (29)$$

where I , C , and S blocks are square diagonal matrices where C and S have entries in $(0, 1)$ on the diagonal, and 0 blocks may be rectangular.⁸ Because D is unitary, we also have $C^2 + S^2 = I$.

Remark 2.11. The form of D naturally induces decompositions $\mathbb{C}^d = \mathcal{X}_0 \oplus \mathcal{X}_C \oplus \mathcal{X}_1$ and $\mathbb{C}^d = \mathcal{Y}_0 \oplus \mathcal{Y}_C \oplus \mathcal{Y}_1$ into direct sums of three spaces. Hence, $D : \mathbb{C}^d \rightarrow \mathbb{C}^d$ can be seen as a map $D : \mathcal{X}_0 \oplus \mathcal{X}_C \oplus \mathcal{X}_1 \rightarrow \mathcal{Y}_0 \oplus \mathcal{Y}_C \oplus \mathcal{Y}_1$, such that D is a direct sum of three linear maps.

$$\left(\begin{array}{cc|cc} 0 & & I & \\ & C & & S \\ \hline & & I & 0 \\ I & & & \\ & S & & -C \\ & & 0 & -I \end{array} \right) = \underbrace{\begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}}_{\mathcal{X}_0 \rightarrow \mathcal{Y}_0} \oplus \underbrace{\begin{pmatrix} C & S \\ S & -C \end{pmatrix}}_{\mathcal{X}_C \rightarrow \mathcal{Y}_C} \oplus \underbrace{\begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}}_{\mathcal{X}_1 \rightarrow \mathcal{Y}_1}.$$

⁷In fact, there is some sense in which the SVD and the CSD are special cases of the same object, a *generalized Cartan decomposition*. We recommend the survey by Edelman and Jeong for readers curious about this connection [EJ23].

⁸Blocks may be non-existent. The I blocks may not necessarily be the same size, but C and S are the same size.

The key resulting intuition for QSVT is that, supposing everything is square, these blocks can be further decomposed into 2×2 blocks of the reflection matrix from quantum signal processing,

$$\begin{pmatrix} \lambda_i & \sqrt{1 - \lambda_i^2} \\ \sqrt{1 - \lambda_i^2} & -\lambda_i \end{pmatrix},$$

where $\{\lambda_i\}$ are the singular values of U_{11} .

2.3 Proving QSVT

We now apply the machinery of the two previous sections to prove Theorem 2.1. We begin with some helpful notation in this special case, following the partitioning given by Theorem 2.10.

Definition 2.12 (Variant of [GSLW19, Definition 12]). Let $U \in \mathbb{C}^{d \times d}$ be a Q -block encoding of $A \in \mathbb{C}^{r \times c}$ where $B_{L,1}$ and $B_{R,1}$ are the first r and c columns of the identity, respectively, as in Definition 1.1. By Theorem 2.10, there is a CS decomposition compatible with the partitioning of U :

$$U = \begin{pmatrix} A & U_{12} \\ U_{21} & U_{22} \end{pmatrix} = \underbrace{\begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix}}_V \underbrace{\begin{pmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{pmatrix}}_D \underbrace{\begin{pmatrix} W_1 & \\ & W_2 \end{pmatrix}^\dagger}_{W^\dagger}.$$

In Definition 2.12, we applied Theorem 2.10 to obtain an SVD of $A = V_1 D_{11} W_1$ that we have extended to the d -dimensional U . Recall also that we defined Π_L and Π_R to be the identity but with all but the first r and c 1's set to 0, respectively.

Our proof of Theorem 2.9 falls out of this decomposition: with it, we can reduce to a diagonal case, which mirrors QSP in the desired way.

Proof of Theorem 2.9. We recall the definition of U_Φ :

$$U_\Phi = \begin{cases} e^{i\phi_n(2\Pi_L - I)} U e^{i\phi_{n-1}(2\Pi_R - I)} \prod_{j=0}^{\frac{n-3}{2}} U^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} U e^{i\phi_{2j}(2\Pi_R - I)} & \text{if } n \text{ is odd, and} \\ e^{i\phi_n(2\Pi_R - I)} \prod_{j=0}^{\frac{n-2}{2}} U^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} U e^{i\phi_{2j}(2\Pi_R - I)} & \text{if } n \text{ is even.} \end{cases}$$

We observe that this SVD commutes appropriately with exponentiated reflections respecting the partition.

$$e^{i\phi(2\Pi_L - I)} = \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix}, \quad e^{i\phi(2\Pi_R - I)} = \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix},$$

with appropriate block sizes, and

$$\begin{aligned} \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix} \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} &= \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix}, \\ \begin{pmatrix} W_1 & \\ & W_2 \end{pmatrix} \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix} &= \begin{pmatrix} e^{i\phi} I & \\ & e^{-i\phi} I \end{pmatrix} \begin{pmatrix} W_1 & \\ & W_2 \end{pmatrix}. \end{aligned}$$

So, we continue:

$$\begin{aligned} &= \begin{cases} V e^{i\phi_n(2\Pi_L - I)} D e^{i\phi_{n-1}(2\Pi_R - I)} \left(\prod_{j=0}^{\frac{n-3}{2}} D^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} D e^{i\phi_{2j}(2\Pi_R - I)} \right) W^\dagger & \text{if } n \text{ is odd, and} \\ W (e^{i\phi_n(2\Pi_R - I)} \prod_{j=0}^{\frac{n-2}{2}} D^\dagger e^{i\phi_{2j+1}(2\Pi_L - I)} D e^{i\phi_{2j}(2\Pi_R - I)}) W^\dagger & \text{if } n \text{ is even.} \end{cases} \\ &= \begin{cases} V D_\Phi W^\dagger & \text{if } n \text{ is odd, and} \\ W D_\Phi W^\dagger & \text{if } n \text{ is even.} \end{cases} \end{aligned} \tag{30}$$

This reduces the problem to computing D_Φ . Recall from (29) that the structure of D is

$$\left(\begin{array}{c|c} D_{11} & D_{12} \\ \hline D_{21} & D_{22} \end{array} \right) = \left(\begin{array}{cc|cc} 0 & & I & \\ & C & & S \\ \hline & I & & 0 \\ & & S & -C \\ & & 0 & -I \end{array} \right) = \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} \oplus \begin{pmatrix} C & S \\ S & -C \end{pmatrix} \oplus \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}.$$

Similarly, where the blocks below denote the same direct sum decomposition above, for $\phi \in \mathbb{R}$,

$$\begin{aligned} e^{i\phi(2\Pi_L - I)} &= \left(\begin{array}{c|c} e^{i\phi I} & \\ \hline & e^{-i\phi I} \end{array} \right) = \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix} \oplus \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix} \oplus \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix}, \\ e^{i\phi(2\Pi_R - I)} &= \left(\begin{array}{c|c} e^{i\phi I} & \\ \hline & e^{-i\phi I} \end{array} \right) = \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix} \oplus \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix} \oplus \begin{pmatrix} e^{i\phi I} & \\ & e^{-i\phi I} \end{pmatrix}. \end{aligned}$$

Leveraging this direct sum decomposition of D , we can reduce our calculation to computing the alternating sequence for each block, yielding

$$\begin{aligned} D_\Phi &= \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix}_\Phi \oplus \begin{pmatrix} C & S \\ S & -C \end{pmatrix}_\Phi \oplus \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix}_\Phi \\ &= \begin{cases} \begin{pmatrix} 0 & \cdot \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} p^{(\text{SV})}(C) & \cdot \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} p(1)I & \cdot \\ \cdot & \cdot \end{pmatrix} & \text{if } n \text{ is odd, and} \\ \begin{pmatrix} p(0)I & \cdot \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} p^{(\text{SV})}(C) & \cdot \\ \cdot & \cdot \end{pmatrix} \oplus \begin{pmatrix} p(1)I & \cdot \\ \cdot & \cdot \end{pmatrix} & \text{if } n \text{ is even.} \end{cases} \end{aligned}$$

This last equality is a crucial but elementary calculation. The full derivation is given in [TT24], but the main point is that these cases are simple enough that they can be computed directly. The intuition behind why we get the ‘expected’ answers is that, by assumption and (8),

$$\prod_{j \in [n]} \begin{pmatrix} e^{i\phi_j} & 0 \\ 0 & e^{-i\phi_j} \end{pmatrix} \begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix} = \begin{pmatrix} p(x) & \cdot \\ \cdot & \cdot \end{pmatrix}.$$

So, supposing we could evaluate the polynomial at a matrix $x \leftarrow C$, we get that, using that $\sqrt{I - C^2} = S$,

$$\text{“} \prod_{j \in [n]} \begin{pmatrix} e^{i\phi_j} I & 0 \\ 0 & e^{-i\phi_j} I \end{pmatrix} \begin{pmatrix} C & S \\ S & -C \end{pmatrix} = \begin{pmatrix} p(C) & \cdot \\ \cdot & \cdot \end{pmatrix} \text{.”}$$

This should hold because block matrix multiplication operates by the same rules as scalar matrix multiplication, but requires care to handle the non-square case.

Returning to our calculation, for n odd, recalling (30) and $p(0) = 0$, we have

$$\begin{aligned}
\Pi_L U_\Phi \Pi_R &= \Pi_L V D_\Phi W^\dagger \Pi_R \\
&= \begin{pmatrix} I \\ \end{pmatrix} \begin{pmatrix} V_1 & \\ & V_2 \end{pmatrix} D_\Phi \begin{pmatrix} W_1^\dagger & \\ & W_2^\dagger \end{pmatrix} \begin{pmatrix} I \\ \end{pmatrix} = \begin{pmatrix} V_1 \\ \end{pmatrix} D_\Phi \begin{pmatrix} W_1^\dagger \\ \end{pmatrix} \\
&= \left(\begin{array}{c|c} V_1 \begin{pmatrix} 0 & \\ & p^{(SV)}(C) \end{pmatrix} W_1^\dagger & \\ \hline & p(1)I \end{array} \right) = \left(\begin{array}{c|c} p^{(SV)}(A) & 0 \\ \hline 0 & 0 \end{array} \right).
\end{aligned}$$

Similarly, for n even, we have

$$\begin{aligned}
\Pi_R U_\Phi \Pi_R &= \Pi_R W D_\Phi W^\dagger \Pi_R \\
&= \begin{pmatrix} W_1 \\ \end{pmatrix} D_\Phi \begin{pmatrix} W_1^\dagger \\ \end{pmatrix} \\
&= \left(\begin{array}{c|c} W_1 \begin{pmatrix} p(0)I & \\ & p^{(SV)}(C) \end{pmatrix} W_1^\dagger & \\ \hline & p(1)I \end{array} \right) = \left(\begin{array}{c|c} p^{(SV)}(A) & 0 \\ \hline 0 & 0 \end{array} \right).
\end{aligned}$$

We get what we want. □

Problem Set 2: The QSVT

Problem 2.1 (When will my reflection show who I am inside?). QSVT achieves polynomials by interspersing phase operators with signal rotation operators. However, these rotation operators may look different in the literature. Consider two potential operators, $W(x), R(x)$, with the following matrix forms:

$$W(x) = \begin{pmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{pmatrix} \quad R(x) = \begin{pmatrix} x & \sqrt{1-x^2} \\ \sqrt{1-x^2} & -x \end{pmatrix} \quad (31)$$

Where W is the rotation operator while R is the reflection operator. We can define two different kinds of QSP, $\mathbf{QSP}_W(\Phi, x)$ and $\mathbf{QSP}_R(\Phi, x)$ for these two different operators. For example,

$$\mathbf{QSP}_W(\Phi, x) := \left(\prod_{j=1}^n e^{i\phi_j \sigma_z} W(x) \right) e^{i\phi_0 \sigma_z}.$$

Suppose we have some series of phases $\Phi = (\phi_0, \dots, \phi_n)$ such that $\mathbf{QSP}_W(\Phi, x)$ forms a desired polynomial $p(x)$. Can we find a Φ' such that $\mathbf{QSP}_R(\Phi', x)$ performs the same polynomial? If so, find a formula for Φ' in terms of Φ ; if not, prove why.

Problem 2.2 (Perfectly balanced, as all things should be). The Chebyshev polynomials of the first and second kind are functions such that, for all $z \in \mathbb{C}$,

$$\begin{aligned} T_n\left(\frac{1}{2}(z + z^{-1})\right) &= \frac{1}{2}(z^n + z^{-n}) \\ U_n\left(\frac{1}{2}(z + z^{-1})\right) &= (z^{n+1} - z^{-(n+1)})/(z - z^{-1}) \end{aligned}$$

Prove that T_n and U_n are polynomials. Then, prove that

$$T_n(x)^2 + (1-x^2)U_{n-1}(x)^2 = 1. \quad (32)$$

Just a little more and we have a proof that these can be used in QSP/QSVT!

Problem 2.3 (They're the same picture!). Return to [BCKKS17, Lemma 3.6]. What are the angles of the phase operators? What are the polynomials that are being computed with these phase operators? (A recursive definition is fine.)

Problem 2.4 (Block-encodings for any matrix). Given a matrix $A \in \mathbb{C}^{d \times d}$ such that $\|A\| \leq 1$, show there exists a unitary $U \in \mathbb{C}^{2d \times 2d}$ such that U is a block-encoding of A :

$$U = \begin{pmatrix} A & \cdot \\ \cdot & \cdot \end{pmatrix}.$$

Prove that $2d$ is tight, i.e., there is some matrix A such that any unitary with A as a submatrix must be size at least $2d \times 2d$. *Note: this is true for non-square A as well, but the argument might get more annoying.*

Problem 2.5 (It's just a phase). In our QSVT algorithm, we needed to apply gates of the form $e^{i\phi(2\Pi - I)}$, where $\Pi = (|0\rangle^{\otimes a} \langle 0|^{\otimes a}) \otimes I$. How do you implement these?

3 Approximating many things by polynomials

In the previous lecture, we showed that we can get block-encodings of $p^{(\text{SV})}(A)$ from block-encodings of A , provided that $p(x)$ is a degree- n , even or odd polynomial such that $|p(x)| \leq 1$ for $x \in [-1, 1]$. Roughly, this turned a Q -block encoding to a nQ -block encoding

For applications of interest, the main goal is actually to apply a non-polynomial function; to capture these applications, we need tools for approximating the relevant functions with bounded polynomials. In this lecture, we introduce Chebyshev polynomials, our main tool for constructing such approximations. We will see that the class of low-degree bounded polynomials is expressive enough for many applications.

More philosophically, it's important to build intuition on when we can expect to approximate functions by polynomials, as polynomial approximation is a ubiquitous tool in quantum algorithms (and classical algorithms too).

3.1 Chebyshev polynomials and properties

Chebyshev polynomials are everywhere in applied math; we'll cover a small amount of the theory here.

Definition 3.1 (Chebyshev polynomial). The degree- n Chebyshev polynomial (of the first kind), denoted $T_n(x)$, is the function that satisfies, for all $z \in \mathbb{C}$,

$$T_n\left(\frac{1}{2}(z + z^{-1})\right) = \frac{1}{2}(z^n + z^{-n}). \quad (33)$$

We can see this is a polynomial by verifying that T_n satisfies the recurrence

$$T_n = 2x \cdot T_{n-1} - T_{n-2},$$

with $T_0 = 1$ and $T_1 = x$. Plugging in $z = \exp(i\theta)$ for $\theta \in [-\pi, \pi]$, we get another familiar definition of the Chebyshev polynomials,

$$T_n(\cos(\theta)) = \cos(n\theta).$$

From these definitions we have that $\|T_n(x)\|_{[-1,1]} := \max_{x \in [-1,1]} |T_n(x)| \leq 1$, and that T_n has the same parity as n , i.e. $T_n(-x) = (-1)^n T_n(x)$.

Under mild “niceness” conditions, any function can be written as a series of Chebyshev polynomials $f(x) = \sum_{k \geq 0} a_k T_k(x)$.

Lemma 3.2 ([Tre19, Theorem 3.1]). *Let $f : [-1, 1] \rightarrow \mathbb{R}$ be Lipschitz (i.e. $|f(x) - f(y)| \leq C|x - y|$ for finite C). Then f has a unique decomposition into Chebyshev polynomials*

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x),$$

where the Chebyshev coefficients a_k absolutely converge.

This is true for the same reason functions have Fourier series. In fact, the theory of Chebyshev polynomials is a parallel theory. For $z = e^{i\theta}$, define $g(z) = f\left(\frac{1}{2}(z + z^{-1})\right)$. Then $g(z)$ is a function on the unit circle with a Laurent series.

$$g(z) = \sum_k a_k T_k\left(\frac{1}{2}(z + z^{-1})\right) = \sum_k \frac{a_k}{2} (z^k + z^{-k}).$$

For $\theta \in [-\pi, \pi]$, define $h(\theta) = g(e^{i\theta}) = f(\frac{1}{2}(e^{i\theta} + e^{-i\theta}))$. Then we can think of $h(\theta)$ as a 2π -periodic function with a Fourier series.

$$h(\theta) = \sum_k a_k T_k(\frac{1}{2}(e^{i\theta} + e^{-i\theta})) = \sum_k \frac{a_k}{2} (e^{ik\theta} + e^{-ik\theta}).$$

Roughly the same kinds of statements can be made in these three different settings (see the appendices of [Tre19]); the Chebyshev version of the theory is most useful to us because our algorithmic techniques work best for polynomials.

If you are familiar with Fourier analysis, you might now expect the Chebyshev polynomials to obey some kind of orthogonality property. This is indeed the case, with the appropriate choice of inner product.

Lemma 3.3 (Orthogonality property). *$\{T_k\}_k$ are orthogonal under a particular choice of inner product.*

$$\frac{2}{\pi} \int_{-1}^1 \frac{T_k(x)T_\ell(x)}{\sqrt{1-x^2}} dx = \begin{cases} 2 & k = \ell = 0 \\ 1 & k = \ell \\ 0 & \text{otherwise} \end{cases}$$

Proof. Substituting $x = \cos(\theta)$ and $dx = -\sin(\theta)d\theta = -\sqrt{1-x^2}d\theta$,

$$\begin{aligned} \int_{-1}^1 \frac{T_k(x)T_\ell(x)}{\sqrt{1-x^2}} dx &= \int_\pi^0 -T_k(\cos(\theta))T_\ell(\cos(\theta))d\theta \\ &= \int_0^\pi \cos(k\theta)\cos(\ell\theta)d\theta = \begin{cases} \frac{\pi}{2} & k = \ell \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

(When $k = \ell = 0$, we have π instead.) □

From the orthogonality property, we can think about the Chebyshev series $f(x) = \sum_{k \geq 0} a_k T_k(x)$ as writing f as a vector in an orthogonal basis of functions, with a_k as the entries. Here, we can immediately deduce properties of these entries.

Lemma 3.4 (Chebyshev coefficients, [Tre19, Theorem 3.1]). *For $f(x)$ with a Chebyshev series $f(x) = \sum_{k \geq 0} a_k T_k(x)$, the Chebyshev coefficients can be computed with the integral*

$$a_k = \frac{2}{\pi} \int_{-1}^1 \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx. \tag{34}$$

For $k = 0$ the same formula holds with the factor $2/\pi$ changed to $1/\pi$.

Lemma 3.5 (Chebyshev coefficients are bounded). *If $\|f(x)\|_{[-1,1]} \leq 1$, then*

$$\begin{aligned} |a_k| &= \frac{2}{\pi} \left| \int_{-1}^1 \frac{f(x)T_k(x)}{\sqrt{1-x^2}} dx \right| \\ &\leq \frac{2}{\pi} \int_{-1}^1 \frac{\|f\|_{[-1,1]} \|T_k\|_{[-1,1]}}{\sqrt{1-x^2}} dx \\ &\leq \frac{2}{\pi} \int_{-1}^1 \frac{1}{\sqrt{1-x^2}} dx \\ &\leq 2. \end{aligned}$$

It should make us happy that Chebyshev coefficients are bounded, since other kinds of coefficients can behave much worse. For example, Chebyshev polynomials are bounded, $\|T_n(x)\|_{[-1,1]} = 1$, but if we write it out as a linear combination of monomials x^k , the coefficient for x^n is 2^{n-1} . That reasonable functions can have exponentially large coefficients in the monomial basis is a common stumbling block for the working algorithmist. The solution is often to work in a nicer basis, like Chebyshev polynomials.

3.2 Approximating functions from Chebyshev series

If we know that a function has a convergent Chebyshev series, we can approximate it by a low-degree polynomial by truncating the series.

Definition 3.6 (Chebyshev truncation). For a function $f : [-1, 1] \rightarrow \mathbb{C}$ written as a Chebyshev series $f(x) = \sum_{k=0}^{\infty} a_k T_k(x)$, we denote the degree- n Chebyshev truncation of f as

$$f_n(x) = \sum_{k=0}^n a_k T_k(x).$$

If the function f one wishes to approximate is standard, closed forms of the Chebyshev coefficients may be known, so one can take a Chebyshev truncation and explicitly bound the error:

$$\|f - f_n\|_{[-1,1]} = \left\| \sum_{k=n+1}^{\infty} a_k T_k(x) \right\|_{[-1,1]} \leq \sum_{k=n+1}^{\infty} |a_k| \|T_k(x)\|_{[-1,1]} = \sum_{k=n+1}^{\infty} |a_k|.$$

In other words, by choosing n such that the coefficient tail sum is bounded by ε , we obtain an ε -uniform approximation on $[-1, 1]$. If we had these Chebyshev coefficients explicitly, we could bound them directly to get a polynomial approximation. However, we may not know explicitly what the Chebyshev coefficients of our desired function is, so we can't easily bound them. The following shows that the Chebyshev coefficient tail is exponential, provided that the function is analytic around $[-1, 1]$.

Theorem 3.7 ([Tre19, Theorems 8.1 and 8.2]). *Let f be an analytic function in $[-1, 1]$ and, for some $\rho > 1$, analytically continuable to the interior of the Bernstein ellipse $E_\rho = \{\frac{1}{2}(z + z^{-1}) : |z| = \rho\}$, where it satisfies $|f(x)| \leq M$. Then its Chebyshev coefficients satisfy $|a_0| \leq M$ and $|a_k| \leq 2M\rho^{-k}$ for $k \geq 1$.*

Corollary 3.8. *Consequently, for each $n \geq 0$, its Chebyshev truncations satisfy*

$$\|f - f_n\|_{[-1,1]} \leq \sum_{k \geq n+1} |a_k| \leq 2M \sum_{k \geq n+1} \rho^{-k} = \frac{2M\rho^{-n}}{\rho - 1},$$

and choosing $n = \lceil \frac{1}{\log(\rho)} \log \frac{2M}{(\rho-1)\varepsilon} \rceil$, we have $\|f - f_n\|_{[-1,1]} \leq \varepsilon$.

Proof of Theorem 3.7. Recall from (34) (and since inverting z does not change the contour integral) that for $k \geq 1$,

$$a_k = \frac{1}{\pi i} \int_{|z|=1} z^{-(k+1)} f\left(\frac{1}{2}(z + z^{-1})\right) dz.$$

The boundary of E_ρ is given by $\frac{1}{2}(z + z^{-1})$ for $|z| = \rho$, and f is analytic in E_ρ , so we may choose a different contour without affecting the value of the integral:

$$a_k = \frac{1}{\pi i} \int_{|z|=\rho} z^{-(k+1)} f\left(\frac{1}{2}(z + z^{-1})\right) dz.$$

The conclusion follows from the facts that the circumference of $|z| = \rho$ is $2\pi\rho$ and the function is bounded by M . A similar argument gives the case $k = 0$, where (34) has $2\pi i$ in the denominator. \square

Fact 3.9. *The Bernstein ellipse E_ρ for $\rho = 1 + \delta \leq 2$ satisfies*

$$\text{interior}(E_\rho) \subset \left\{ x + iy \mid x, y \in \mathbb{R}, |x| \leq 1 + \frac{\delta^2}{2} \text{ and } |y| \leq \delta \right\}.$$

Corollary 3.10 (Application to Hamiltonian simulation). *Consider the function $\sin(tx)$. Then for $z = a + ib$ on the interior of the Bernstein ellipse E_ρ ,*

$$\begin{aligned} |\sin(tz)| &\leq \frac{1}{2} |e^{itz} - e^{-itz}| \\ &\leq \frac{1}{2} (|e^{-bt}| + |e^{bt}|) \leq e^{|bt|}. \end{aligned}$$

So, choosing $\rho = 1 + 1/t$, we can apply the theorem with $M = \mathcal{O}(1)$. When $t \geq 1$, this gives an ε -good approximation for the Chebyshev truncation of degree

$$n = \mathcal{O}\left(t \log \frac{t}{\varepsilon}\right).$$

So, we can use that a function $f(x)$ is analytic and bounded in a small ball around $[-1, 1]$ to conclude that the Chebyshev truncation is a good polynomial approximation. Further, we can get degree bounds which scale with $\log \frac{1}{\varepsilon}$, corresponding to the coefficient tail being exponential. This is a great bound for almost all purposes, but can be loose: for example, the Chebyshev coefficients of $\sin(tx)$ actually decay super-exponentially, and by being more careful it's possible to show a scaling of $\log(1/\varepsilon)/\log \log(1/\varepsilon)$ (for t constant) [TT24].

Remark 3.11 (Chebyshev approximation vs Taylor series approximation). You might be wondering what the difference is between truncating a Chebyshev series and truncating a Taylor series, perhaps a more commonly known tool for polynomial approximation. In fact, one can make a statement similar to Theorem 3.7 about Taylor series truncations (see [GSLW19, Corollary 66]), but it becomes difficult to apply in settings where one does not know the Taylor series. Note that this does not always give the right polynomial approximation; for example, truncating the Taylor series of e^x gives a degree which is quadratically worse than optimal [SV14].

Theorem 3.7 shows that if one can analytically continue f to a Bernstein ellipse with $\rho = 1 + \alpha$ for small α , then a degree $\approx \frac{1}{\alpha}$ polynomial obtains good approximation error on $[-1, 1]$. Unfortunately, since the approximation in Theorem 3.7 is based on Chebyshev truncation, the approximation rapidly blows up outside the range $[-1, 1]$ (i.e. growing as $O(|x|^n)$ for x sufficiently outside $[-1, 1]$). In interesting applications of the QSVT framework, this is an obstacle. For example, to use QSVT for solving a system of linear equations, we need a polynomial approximation to x^{-1} on $[\delta, 1]$ that is bounded on $[-1, 1]$. Upon linearly remapping $[\delta, 1]$ to $[-1, 1]$, this corresponds to a bounded approximation on $[-b, 1]$ for some $b > 1$, so Chebyshev truncations give us a very poor degree of control.

Chebyshev truncation is not enough for our purposes, since our criteria is *different* from uniform approximation on $[-1, 1]$. For quantum linear systems, we require a polynomial approximation close to $1/x$ on $[-1, -1/\kappa] \cup [1/\kappa, 1]$, but it merely needs to be bounded on $[-1/\kappa, 1/\kappa]$. This bounded requirement comes from the block-encoding machinery, which requires boundedness to work.

As [GSLW19] points out, there are generic ways to find approximations to piecewise smooth functions which satisfy this sort of “ ε -close on smooth pieces, but bounded near points of discontinuity” requirement, with $\log \frac{1}{\varepsilon}$ scaling in the degree.

Theorem 3.12 ([TT24, Theorem 21]). *Let f be an analytic function in $[-1, 1]$ and analytically continuable to the interior of E_ρ where $\rho = 1 + \alpha$, where it is bounded by M . For $\delta \in (0, \frac{1}{C} \min(1, \alpha^2))$ where C is a sufficiently large constant, $\varepsilon \in (0, 1)$, and $b > 1$, there is a polynomial q of degree $O(\frac{b}{\delta} \log \frac{b}{\delta\varepsilon})$ such that*

$$\begin{aligned} \|f - q\|_{[-1,1]} &\leq M\varepsilon, \\ \|q\|_{[-(1+\delta), 1+\delta]} &\leq M, \\ \|q\|_{[-b, -(1+\delta)] \cup [1+\delta, b]} &\leq M\varepsilon. \end{aligned}$$

Proof sketch.

1. Applying Theorem 3.7 gives f_n of degree $n \approx \frac{1}{\alpha}$ approximating f in the interval $[-1, 1]$, but f_n does not satisfy the other required conclusions due to its growth outside $[-1, 1]$.
2. We multiply f_n by a “threshold” r based on the Gaussian error function erf, whose tails decay much faster than the Chebyshev polynomials grow outside $[-1, 1]$. Our function r has the property that inside $[-1, 1]$, it is close to 1, and outside $[-(1 + \delta), 1 + \delta]$, it is close to 0.
3. Using bounds on the growth of erf, we show $r \cdot f_n$ is bounded on a Bernstein ellipse of radius $1 + \frac{\delta}{b}$ appropriately rescaled, and applying Theorem 3.7 once more gives the conclusion.

□

We now apply this theorem to give a polynomial approximation of δ/x which is close in the region $[\delta, 1]$ and bounded on $[-1, 1]$. If we use QSVT to apply this polynomial to a block-encoding of a matrix, it will approximately invert it if its singular values are at least δ ; this is a clean way to derive the HHL quantum algorithm for solving systems of linear equations [HHL09].

Corollary 3.13. *Let $\delta, \varepsilon \in (0, 1)$, and let $f(x) = \frac{\delta}{x}$. There exist both even and odd polynomials $p(x)$ of degree $O(\frac{1}{\delta} \log \frac{1}{\delta\varepsilon})$ such that $\|p\|_{[-1,1]} \leq 3$ and $\|p - f\|_{[\delta,1]} \leq \varepsilon$.*

Proof. Assume δ is sufficiently small, else taking a smaller δ only affects the bound by a constant. We rescale the region of interest: $x = \frac{1-\delta}{2}y + \frac{1+\delta}{2}$ is in $[\delta, 1]$ for $y \in [-1, 1]$, so let

$$g(y) := \delta \left(\frac{1-\delta}{2}y + \frac{1+\delta}{2} \right)^{-c}.$$

We require a bound of g on E_ρ for $\rho = 1 + \sqrt{\delta/4}$. Since f is largest closest to the origin, g is largest at the point closest to $-\frac{1+\delta}{1-\delta}$, i.e. $-\frac{1}{2}(\rho + \rho^{-1}) > -(1 + \frac{\delta}{8})$ by Fact 3.9. Further,

$$\begin{aligned} g\left(-\frac{1}{2}(\rho + \rho^{-1})\right) &\leq g\left(-\left(1 + \frac{\delta}{8}\right)\right) \\ &\leq \delta \left(-\frac{1-\delta}{2}\left(1 + \frac{\delta}{8}\right) + \frac{1+\delta}{2}\right)^{-1} \\ &= \left(1 - \frac{1-\delta}{16}\right)^{-1} \leq \frac{3}{2}. \end{aligned}$$

Let $\tilde{\delta} = \frac{\delta}{4C}$ for sufficiently large C , and $b = 4$. Theorem 3.12 yields $q(y)$ satisfying:

$$\|q(y) - g(y)\|_{[-1,1]} \leq \varepsilon, \quad \|q(y)\|_{[-(1+\tilde{\delta}), 1+\tilde{\delta}]} \leq 2, \quad \|q(y)\|_{[-4, -(1+\tilde{\delta})] \cup [1+\tilde{\delta}, 4]} \leq \varepsilon.$$

Shifting back $y = \frac{2}{1-\delta}(x - \frac{1+\delta}{2})$, it is clear for sufficiently large C that $y = -\frac{1+3\delta}{1-\delta}$ (which corresponds to $x = -\delta$) has $y < -(1 + \tilde{\delta})$, and $y = -\frac{3+\delta}{1-\delta}$ (which corresponds to $x = -1$) has $y > -4$. So,

$$\begin{aligned} \left\|q\left(\frac{2}{1-\delta}\left(x - \frac{1+\delta}{2}\right)\right) - f(x)\right\|_{[\delta, 1]} &\leq \varepsilon, \\ \left\|q\left(\frac{2}{1-\delta}\left(x - \frac{1+\delta}{2}\right)\right)\right\|_{[-\delta, \delta]} &\leq 2, \\ \left\|q\left(\frac{2}{1-\delta}\left(x - \frac{1+\delta}{2}\right)\right)\right\|_{[-1, -\delta]} &\leq \varepsilon. \end{aligned} \tag{35}$$

Depending on whether we wish the final function to be even or odd, we take

$$p(x) = q\left(\frac{2}{1-\delta}\left(x - \frac{1+\delta}{2}\right)\right) \pm q\left(\frac{2}{1-\delta}\left(-x - \frac{1+\delta}{2}\right)\right).$$

Then the guarantees of (35) give $\|p(x) - f(x)\|_{[\delta, 1]} \leq 2\varepsilon$ and $\|p(x)\|_{[-1, 1]} \leq 3$, and we rescale ε to conclude. The final degree of the polynomial is the degree of $q(y)$: $O(\frac{1}{\delta} \log \frac{1}{\delta\varepsilon})$. \square

3.3 Lower bounds on polynomial approximation

There are limitations to what kinds of functions can be approximated by low-degree polynomials. The most common lower bounds to keep in mind are the *Markov brothers'* and *Bernstein* inequalities:

Theorem 3.14 (Markov brothers' inequality [Sch41, Theorem 1]). *Let $p(x)$ be a degree n polynomial such that $\|p(x)\|_{[-1, 1]} \leq 1$. Then*

$$\|p'(x)\|_{[-1, 1]} \leq n^2 \tag{36}$$

Theorem 3.15 (Bernstein's inequality [Sch41, Theorem 2]). *Let $p(x)$ be a degree n polynomial such that $\|p(x)\|_{[-1, 1]} \leq 1$. Then, for $x \in (-1, 1)$,*

$$|p'(x)| \leq \frac{n}{\sqrt{1-x^2}}. \tag{37}$$

Remark 3.16. This means that a bounded polynomial has derivative $\mathcal{O}(n)$ near the center of $[-1, 1]$, but can be $\mathcal{O}(n^2)$ near the edge. This suggests that functions can be approximated better when its worst-conditioned pieces are on the edges of $[-1, 1]$. This is true: note that the above argument shows that sufficiently good bounded polynomial approximations to δ/x on $[-1, -\delta] \cup [\delta, 1]$ must have degree $\Omega(1/\delta)$.

However, suppose we have a block-encoding of $I - A$, where A is a Hermitian matrix. That $\|I - A\| \leq 1$ implies that A is PSD, and suppose its eigenvalues are in $[\delta, 1]$. We can then get an approximate block-encoding of A^{-1} via a polynomial approximation of

$$\frac{1}{1-x} \text{ for } x \in [-1 + \delta, 1].$$

There is a $\mathcal{O}(1/\sqrt{\delta})$ -degree polynomial approximation of this,

$$\frac{1}{x - (1 + \delta)} = \frac{-2}{\sqrt{2\delta - \delta^2}} \sum_{k=0}^{\infty} (1 + \delta - \sqrt{2\delta + \delta^2})^k T_k(x)$$

implying that we can invert a matrix quadratically faster if we get a block-encoding of $I - A$ rather than A [OD21].

You might think, this doesn't make sense, since surely it's possible to convert a block-encoding of A to a block-encoding of $I - A$. But here, the scaling of the block-encoding becomes very important: we need a block-encoding of $I - A$, not $(I - A)/2$, which is what one gets from extensibility properties of block-encoding. It's possible to amplify this block-encoding to a block-encoding of $(I - A)(1 - \varepsilon)$ via *uniform singular value amplification* [GSLW19, Theorem 30] (which is just QSVT with a different polynomial), but doing this costs too much, making the subsequent quadratic improvement not worth it. The brittle nature of this quadratic savings makes sense if we realize that a block-encoding of $I - A$ 'proves' that A is PSD, whereas we can construct a block-encoding of $(I - A)/2$ for any A . Really, the quadratic savings comes from PSDness—it's the same savings achieved by conjugate gradient [TB97, Lecture 38], indeed for similar reasons—so it shouldn't be possible to cheat and get this improvement for non-PSD A .

Problem Set 3: Polynomial approximation

Problem 3.1 (Polynomial approximation of monomials). First, compute the Chebyshev coefficients of the monomial $m^{(n)}(x) = x^n$. (Doing this via $T_k(\frac{1}{2}(z + z^{-1})) = \frac{1}{2}(z^k + z^{-k})$ formulation may be easiest.) How small can k be such that the Chebyshev truncation $m_k^{(n)}$ a good approximation of $m^{(n)}$:

$$\|m^{(n)} - m_k^{(n)}\|_{[-1,1]} \leq \varepsilon?$$

Problem 3.2 (Chebyshev interpolation [Tre19]). The *Chebyshev interpolant* of a function f , denoted p_n , is the unique degree- n polynomial such that $p_n(x_j) = f(x_j)$ for all $x_j = \cos(j\pi/n)$, $j = 0, 1, \dots, n$. Prove that⁹

$$\|f(x) - p_n(x)\|_{[-1,1]} \leq 2 \sum_{\ell \geq n} |a_\ell|.$$

Hint: when is $T_k(x_j) = T_\ell(x_j)$ for all points $\{x_j\}$?

Problem 3.3 (Jackson theorems, [Tre19]). Let $f : [-1, 1] \rightarrow \mathbb{R}$ be absolutely continuous and suppose f is of bounded variation, meaning that $\int_{-1}^1 |f'(x)| dx \leq V$. Then show that the Chebyshev coefficients of f satisfy

$$|a_k| \leq \frac{2V}{\pi k}.$$

Problem 3.4 (Optimal polynomial approximations; upper and lower bounds). Consider a function $f : [-1, 1] \rightarrow \mathbb{R}$ with a Chebyshev expansion $f(x) = \sum_{k \geq 0} a_k T_k(x)$. Prove that

$$\left(\frac{1}{2} \sum_{k=n+1}^{\infty} a_k^2\right)^{\frac{1}{2}} \leq \min_{\substack{p \in \mathbb{R}[x] \\ \deg p = n}} \|f(x) - p(x)\|_{[-1,1]} \leq \sum_{k=n+1}^{\infty} |a_k|$$

For what kind of Chebyshev coefficient decay is this characterization tight up to constants?

⁹Recall that our approximation results used that $\|f(x) - f_n(x)\|_{[-1,1]} \leq \sum_{\ell \geq n} |a_\ell|$. So, Chebyshev interpolants p_n give the same results as Chebyshev truncations f_n , up to a constant factor. Interpolants have the advantage of being computable in $n + 1$ function evaluations.

4 Introducing quantum-inspired linear algebra

We have established a theory of quantum linear algebra based on the block-encoding: you might have noticed that, with block-encodings, we can implicitly manipulate exponentially large matrices in polynomial time. This raises a natural question: perhaps we can harness Nature’s linear algebra processor to manipulate data exponentially faster than we can with classical computers. The key work in this line is Harrow, Hassidim, and Lloyd’s quantum algorithm for sampling from the solution to a sparse system of linear equations [HHL09]. QML has since rapidly developed into an active field of study with numerous proposals for quantum speedups for machine learning tasks in domains ranging from recommendation systems [KP17] to topological data analysis [LGZ16].

A key tool underlying many QML algorithms is the observation that certain kinds of *data structures* could allow for efficient preparation of block encodings of arbitrary matrices, assuming the ability to query these data structures in superposition (i.e. assuming that the data is in QRAM). In particular, this allows for $\log(rc)$ -block encodings of $A/\|A\|_F$. Many QML algorithms relied on this data structure for exponential speedup, with the belief that this additional assumption would not affect it. However, it was discovered that classical algorithms given this data structure can achieve the same results up to polynomial slowdown. These are known as “dequantized” algorithms. The existence of a dequantized algorithm means that its quantum counterpart cannot give exponential speedups on classical data, illuminating the landscape of QML speedups.

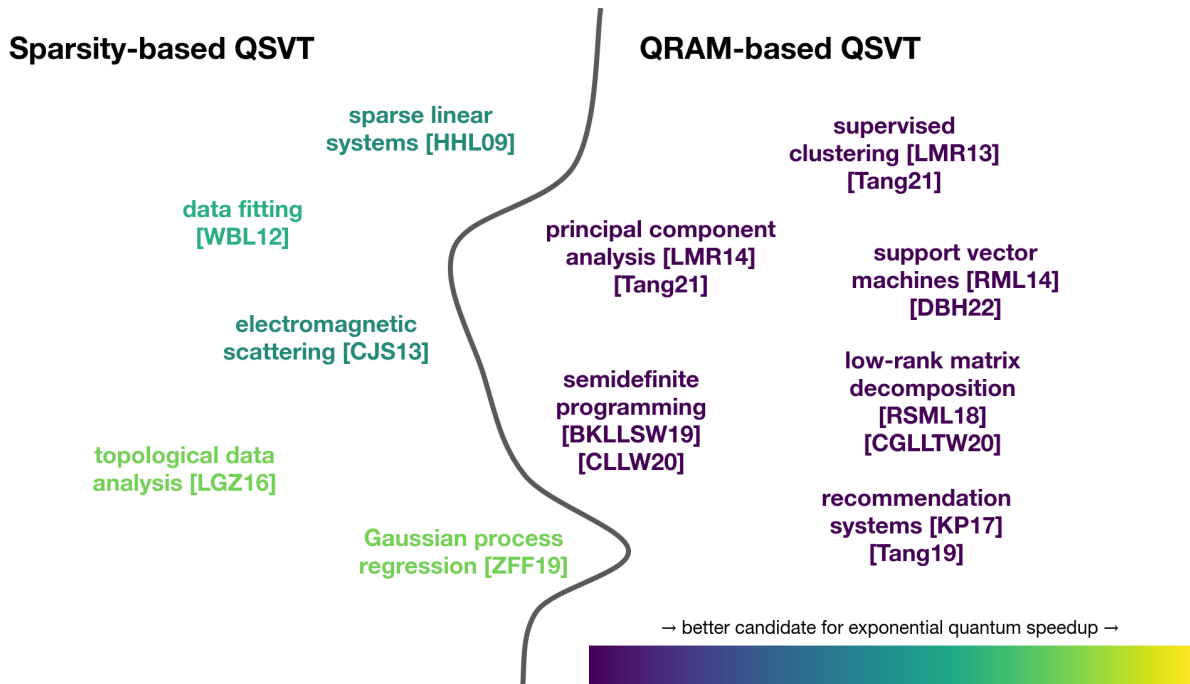


Figure 4: Pictured is the landscape of quantum machine learning algorithms after dequantization. Here, ‘better candidate for exponential quantum speedup’ refers to the number of ‘caveats’ they successfully address, as described by Aaronson [Aar15]. All of these algorithms can be placed in the QSVT framework, and in this framework, the algorithms that do not rely on sparsity assumptions can be dequantized, thereby showing that they cannot produce exponential speedups.

Quantum singular value transformation captures essentially all known linear algebraic

QML techniques [MRTC21], including all prior dequantized QML algorithms (up to minor technical details), so it is our natural target for dequantizing. We cannot hope to dequantize *all* of QSVT, because with sparse input data encoded appropriately, QSVT can simulate algorithms for BQP-complete problems [JW06; HHL09]. However, we show that we can dequantize the QSVT framework, provided that the input data comes in the state preparation data structure commonly used for quantum linear algebra. Such data structures only allow for efficient QML when the input is low-rank. Nevertheless, they are the only way we know how to run quantum linear algebra on unstructured classical data, so this setting covers all QML algorithms that do not rely on sparsity assumptions. We present a classical analogue of the QSVT framework that is only polynomially slower when the input is low rank, and apply it to dequantize QML algorithms.

4.1 A vignette: The swap test, and what’s in an access model

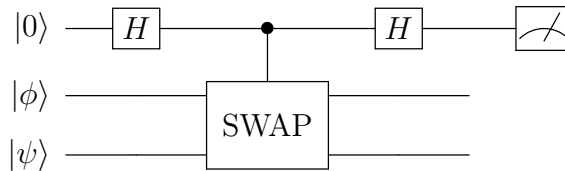


Figure 5: The quantum circuit for the swap test, taken from [BCWW01, Figure 1].

It seems counterintuitive that classical linear algebra algorithms can perform nearly as well as quantum ones, even on classical data. In some sense, what dequantization shows is that some quantum linear algebra algorithms do not fully exploit “quantumness,” since they can be mimicked classically using sampling procedures. We’ll investigate a simple example of a quantum linear algebra algorithm: the *swap test* [BCWW01].

Suppose we have two d -dimensional vectors $\phi, \psi \in \mathbb{C}^d$, both with unit norm.¹⁰ We wish to compute their overlap $|\langle\phi|\psi\rangle|^2$. There is a quantum algorithm, the swap test (shown in Fig. 5), to solve this: prepare the $\log(d)$ -qubit quantum states $|\phi\rangle = \sum_{i=1}^d \phi(i)|i\rangle$ and $|\psi\rangle = \sum_{i=1}^d \psi(i)|i\rangle$, along with one additional qubit in the state $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Then, apply a controlled SWAP between $|\phi\rangle$ and $|\psi\rangle$, with the additional qubit as the control, and then measure this qubit in the Hadamard basis; the measurement produces 1 with probability $\frac{1}{2} - \frac{1}{2}|\langle\phi|\psi\rangle|^2$, so we can use it to estimate the overlap. Averaging over more runs of this circuit gives an estimate to 0.01 error with only $\mathcal{O}(\log(d))$ quantum gates and a constant number of copies of the input states. Even approximating overlaps using classical computers requires $\Omega(d)$ time, since we need to read this many entries of the input to distinguish the two cases $\phi = e_i, \psi = e_i$ and $\phi = e_i, \psi = e_j$. So, we might naively conclude that the swap test achieves an exponential quantum advantage in the task of “computing overlaps”. This is not as farfetched a claim as it might appear: the general version of this task, where we wish to estimate $|\langle 0|^{\otimes n} U |0\rangle^{\otimes n}|$ for $U \in \mathbb{C}^{2^n \times 2^n}$ a unitary matrix encoded as a $\text{poly}(n)$ -sized quantum circuit, indeed gives a quantum advantage (since this task is BQP-hard). Further, this idea has been proposed before in QML: a preprint of Lloyd, Mohseni, and Rebentrost claims to achieve an exponential

¹⁰A heads-up: for these final lectures, the i th index of a vectors is denoted $v(i)$, which is a bit more consistent with this literature. Similarly, entries of matrices are denoted $A(i, j)$.

quantum advantage for clustering with the swap test, computing the distance of a vector to a centroid by estimating the overlap of states like the above [LMR13].

However, the comparison between $\mathcal{O}(\log(d))$ and $\Omega(d)$ hides the difference in input models: the quantum algorithm requires copies of the states $|\phi\rangle$ and $|\psi\rangle$, and the classical lower bound assumes that we are only given the input vectors as lists of entries. For applications to machine learning, it's reasonable to receive the data in the latter form, since the data is classical (in that it comes from classical sources, as is the case for the vast majority of data). For example, machine learning datasets are stored in this way. This leads us to the question: given ϕ and ψ classically, how can we efficiently prepare their corresponding quantum states? Though state preparation assumptions like these are common in quantum linear algebra, they cannot be satisfied in general: the typical way of satisfying them is to assume pre-processing to load the input into a certain kind of data structure in *quantum random access memory* (QRAM) [GLM08; Pra14; JR23]. QRAM is a speculative piece of quantum hardware which supports storing n bits of data and subsequently querying that data in superposition in (functionally) $\text{polylog}(n)$ time, similarly to how we consider classical RAM; for the sake of comparison, we assume the existence of QRAM.¹¹ If we assume that input is given in this data structure (see Fig. 6) for the sake of the quantum computer, then for a fair comparison, we should give our classical computer this same data structure.

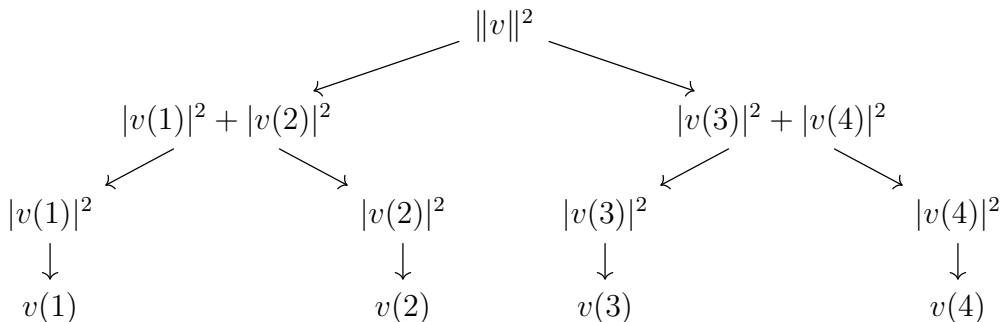


Figure 6: Dynamic data structure used to perform efficient state preparation of a vector $v \in \mathbb{C}^4$. The values displayed are stored in QRAM, along with pointers to other values as designated by the entries. Observe that, by starting from the root of the tree and recursing appropriately, we can sample $i \in [4]$ with probability proportional to $|v(i)|^2$ using only *classical* access to the data structure. A variety of data structures have similar properties, but this one has the advantage of supporting updating entries in $\mathcal{O}(\log n)$ accesses.

If A is in a state preparation data structure in QRAM (like the vector case, see Fig. 7), we can implement a block-encoding of $A/\|A\|_F$ efficiently [GSLW19, Lemma 50].¹² This type of block-encoding is the one commonly used for quantum linear algebra algorithms on classical data, since it works for arbitrary matrices and vectors, paying only a $\|A\|_F/\|A\|$ (the square root of what's known in the classical algorithms literature as stable rank) factor in sub-normalization.

¹¹Of course, neither forms of RAM could be “truly” $\text{polylog}(n)$ time, since storing n bits of data requires $\text{poly}(n)$ space and therefore $\text{poly}(n)$ time for the information to travel across that amount of space. The goal would be to optimize QRAM as well as classical RAM, so that accesses can be treated as $\mathcal{O}(\log(n))$ time, the cost of simply writing down the pointer into the data.

¹² $\|A\|_F$ denotes Frobenius norm, $(\sum_{i,j} |A(i,j)|^2)^{1/2}$.

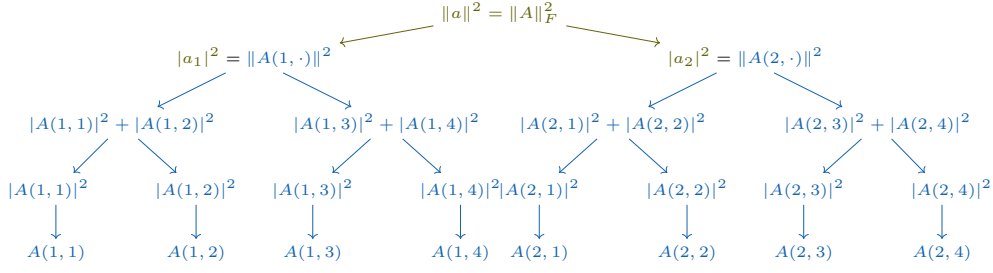


Figure 7: Dynamic data structure for a matrix $A \in \mathbb{C}^{2 \times 4}$. We compose the data structure for a , the vector of row norms, with the data structure for A 's rows.

If ϕ in this data structure, a classical computer can draw independent samples $i \in [n]$ with probability proportional to $|\phi(i)|^2$ with $\mathcal{O}(\log(n))$ accesses. Equipped with this additional type of input access, we can estimate the overlap much faster via a Monte Carlo method: pull one sample, s , from $|\phi\rangle$, and then compute the estimator $\psi(s)/\phi(s)$. This estimator has expected value $\langle \phi | \psi \rangle$ and variance 1, so by averaging over a constant number of runs, we can estimate of the overlap to 0.01 error using $\mathcal{O}(\log d)$ classical gates, assuming that the entries of ϕ and ψ are specified with $\mathcal{O}(\log d)$ bits. The swap test achieves the same dependence on dimension as the dequantized swap test, so it does not give an exponential speedup in this setting. (A more precise analysis would reveal that a quadratic quantum speedup in error is possible, from $\mathcal{O}(1/\varepsilon^2)$ to $\mathcal{O}(1/\varepsilon)$.) This argument against exponential quantum speedup remains valid provided we want to run the quantum algorithm in a setting where we could also perform the quantum-inspired algorithm.

The general principle of the dequantized swap test extends to other QML algorithms. In the typical RAM access model, we assume only that we can query entries efficiently. In other words, we receive our input $v \in \mathbb{C}^n$ as $\mathbf{Q}(v)$ with $\mathbf{q}(v) = 1$.

Definition 4.1 (Query access). For a vector $v \in \mathbb{C}^n$, we have $\mathbf{Q}(v)$, *query access* to v , if for all $i \in [n]$, we can query for $v(i)$. Let $\mathbf{q}(v)$ denote the (time) cost of such a query.

For comparison to quantum algorithms, we assume a stronger input model, sampling and query access, which supports the types of queries we need to perform the overlap estimation algorithm.

Definition 4.2 (Sampling and query access to a vector). For a vector $v \in \mathbb{C}^n$, we have $\mathbf{SQ}(v)$, *sampling and query access* to v , if we can:

1. query for entries of v as in $\mathbf{Q}(v)$;
2. obtain independent samples $i \in [n]$ from the distribution \mathcal{D}_v , which is defined such that the probability of sampling i is $|v(i)|^2 / \|v\|^2$;
3. query for $\|v\|$.

Let $\mathbf{sq}(v)$ denote the time cost of any query.

We will refer to samples from \mathcal{D}_v as *importance samples from v* , though one can view them as measurements of the quantum state $|v\rangle := \frac{1}{\|v\|} \sum v(i)|i\rangle$ in the computational basis.

If we only have $Q(v)$, then responding to queries from $SQ(v)$ (or preparing the state $|v\rangle$) requires linear-time pre-processing. When quantum algorithms use $|v\rangle$, it's sensible to give classical algorithms access to $SQ(v)$, since this is $Q(v)$ with access to computational basis measurements of $|v\rangle$. In fact, the usual methods for efficiently preparing $|v\rangle$ on a quantum computer from v given classically¹³ all seem to incidentally give a classical computer access to $SQ(v)$. For example, if input is loaded in the QRAM data structure, as commonly assumed in QML in order to satisfy state preparation assumptions [Pra14; Cil+18], then we have log-time sampling and query access to it. So, a fast classical algorithm for a problem in this classical model implies lack of quantum speedup for the problem, at least in the usual settings explored in the QML literature.

As the inner product estimation protocol suggests, $SQ(v)$ is a much more powerful access model than $Q(v)$. Classical algorithms can exploit the measurements of input data possible with sampling and query access to speed up linear algebra to become time-independent of the dimension. Specifically, sketching algorithms explore how to use randomness to perform a dimensionality reduction and “sketch” a large matrix A down to a constant-sized normalized submatrix of A which behaves similarly to the full matrix [Woo14]. The computational basis measurements one can produce in the quantum-inspired input model allow for the efficient estimation of matrix products through Monte Carlo methods [DKM06], which can be applied iteratively to produce dequantized algorithms that achieve surprisingly similar bounds to their quantum counterparts. We explore this in our next example.

4.2 Extensibility properties

Quantum-inspired algorithms typically don't give exact sampling and query access to the output vector. Instead, we get a more general version of sampling and query access, which assumes we can only access a sampling distribution that *oversamples* the correct distribution.¹⁴

We can show that *oversampling and query access is approximately closed under arithmetic operations*. These extensibility properties together imply that, given input matrices and vectors in data structures in QRAM, we can get oversampling and query access to low-degree polynomials of the input via extensibility properties; in the same setting, QSVT gives block-encodings of low-degree polynomials of the input, through similar properties. The classical algorithm's runtime is only polynomially slower than the corresponding quantum algorithm (except in the ε parameter). This dequantizes QSVT.

Definition 4.3 (Oversampling). For $p, q \in \mathbb{R}_{\geq 0}^n$ that are distributions, meaning $\sum_i p(i) = \sum_i q(i) = 1$, we say that p ϕ -oversamples q if, for all $i \in [n]$, $p(i) \geq q(i)/\phi$.

The motivation for this definition is the following: if p ϕ -oversamples q , then we can convert a sample from p to a sample from q with probability $1/\phi$ using rejection sampling: sample an i distributed as p , then accept the sample with probability $q(i)/(\phi p(i))$ (which is ≤ 1 by definition).

¹³This assumption is important. When input data is quantum (say, it is coming directly from a quantum system), a classical computer has little hope of performing linear algebra on it efficiently, see for example [ACQ22; CHM21].

¹⁴Oversampling turns out to be the “natural” form of approximation in this setting; other forms of error do not propagate through quantum-inspired algorithms well.

Definition 4.4 (Oversampling and query access to a vector). We have ϕ -oversampling and query access to a vector $v \in \mathbb{C}^n$, $\text{SQ}_\phi(v)$, if:

1. we can query for entries of v , $\text{Q}(v)$, and;
2. we have sampling and query access to an “entry-wise upper bound” vector \tilde{v} , $\text{SQ}(\tilde{v})$, where $\|\tilde{v}\|^2 = \phi\|v\|^2$ and $|\tilde{v}(i)| \geq |v(i)|$ for all indices $i \in [n]$.

Let $\text{sq}_\phi(v)$ denote the time cost of any query.

The parameter ϕ is the classical analogue of the scaling parameter α for block-encodings [GSLW19]. These appear in running times of algorithms because they correspond to overhead in rejection sampling and post-selection, respectively.

Lemma 4.5 (Sampling from oversampling). *Suppose we are given $\text{SQ}_\phi(v)$ and some $\delta \in (0, 1]$. Denote $\overline{\text{sq}}(v) := \phi \text{sq}_\phi(v) \log \frac{1}{\delta}$. We can sample from \mathcal{D}_v with probability $\geq 1 - \delta$ in $\mathcal{O}(\overline{\text{sq}}(v))$ time. We can also estimate $\|v\|$ to ν multiplicative error for $\nu \in (0, 1]$ with probability $\geq 1 - \delta$ in $\mathcal{O}(\frac{1}{\nu^2} \overline{\text{sq}}(v))$ time.*

Proof. Consider the following rejection sampling algorithm to generate samples: sample an index i from \tilde{v} , and output it as the desired sample with probability $r(i) := \frac{|v(i)|^2}{|\tilde{v}(i)|^2}$. Otherwise, restart. We can perform this: we can compute $r(i)$ in $\mathcal{O}(\text{sq}_\phi(v))$ time and $r(i) \leq 1$ since \tilde{v} bounds v .

The probability of accepting a sample in a round is $\sum_i \mathcal{D}_{\tilde{v}}(i)r(i) = \|v\|^2/\|\tilde{v}\|^2 = \phi^{-1}$ and, conditioned on a sample being accepted, the probability of it being i is $|v(i)|^2/\|v\|^2$, so the output distribution is \mathcal{D}_v as desired. So, to get a sample with $\geq 1 - \delta$ probability, run rejection sampling for at most $2\phi \log \frac{1}{\delta}$ rounds.

To estimate $\|v\|^2$, notice that we know $\|\tilde{v}\|^2$, so it suffices to estimate $\|v\|^2/\|\tilde{v}\|^2$ which is ϕ^{-1} . The probability of accepting the rejection sampling routine is ϕ^{-1} , so we run $3\nu^{-2}\phi \log \frac{2}{\delta}$ rounds of it for estimating ϕ^{-1} . Let Z denote the fraction of them which end in acceptance. Then, by a Chernoff bound we have

$$\Pr[|Z - \phi^{-1}| \geq \nu\phi^{-1}] \leq 2 \exp\left(-\frac{\nu^2 z \phi^{-1}}{2 + \nu}\right) \leq \delta,$$

so $Z\|\tilde{v}\|^2$ is a good multiplicative approximation to $\|v\|^2$ with probability $\geq 1 - \delta$. \square

Generally, compared to a quantum algorithm that can output (and measure) a desired vector $|v\rangle$, our algorithms will output $\text{SQ}_\phi(u)$ such that $\|u - v\|$ is small. So, $\overline{\text{sq}}(u)$ is the relevant complexity measure that we will analyze and bound: if we wish to mimic samples from the output of the quantum algorithm we dequantize, we will pay a one-time cost to run our quantum-inspired algorithm for “obtaining” $\text{SQ}_\phi(u)$, and then pay $\overline{\text{sq}}(u)$ cost per additional measurement. As for error, bounds on $\|u - v\|$ imply that measurements from u and v follow distributions that are close in total variation distance [Tan19, Lemma 4.1].

Lemma 4.6 (Summing SQ-accessible vectors [Tan19, Proposition 4.3]). *Given $\text{SQ}_{\phi_t}(v_t) \in \mathbb{C}^n$ and $\lambda_t \in \mathbb{C}$ for all $t \in [\tau]$, we have $\text{SQ}_\phi(\sum_{t=1}^\tau \lambda_t v_t)$ for $\phi = \tau \frac{\sum \phi_t \|\lambda_t v_t\|^2}{\|\sum \lambda_t v_t\|^2}$ and $\text{sq}_\phi(\sum \lambda_t v_t) = \sum_{t=1}^\tau \text{sq}_{\phi_t}(v_t)$ (after paying $\mathcal{O}(\sum_{t=1}^\tau \text{sq}_{\phi_t}(v_t))$ one-time pre-processing cost to query for norms).*

Proof. We prove it for $\tau = 2$, and the general statement follows from induction. Denote $u := \lambda_1 v_1 + \lambda_2 v_2$. To compute $u(s)$ for some $s \in [n]$, we just need to query $v_1(s)$ and $v_2(s)$, querying v_1 and v_2 each once. So, it suffices to get $\text{SQ}(\tilde{u})$ for an appropriate bound \tilde{u} . We choose

$$\tilde{u}(s) = \sqrt{2} \sqrt{|\lambda_1 \tilde{v}_1(s)|^2 + |\lambda_2 \tilde{v}_2(s)|^2},$$

so that $|\tilde{u}(s)| \geq |u(s)|$ by Cauchy–Schwarz, and $\|\tilde{u}\|^2 = 2(\|\lambda_1 \tilde{v}_1\|^2 + \|\lambda_2 \tilde{v}_2\|^2) = 2(\varphi_1^2 \|\lambda_1 v_1\|^2 + \varphi_2^2 \|\lambda_2 v_2\|^2)$, giving the desired value of ϕ .

We have $\text{SQ}(\tilde{u})$: we can compute $\|\tilde{u}\|^2$ by querying for the norms $\|\tilde{v}_1\|$ and $\|\tilde{v}_2\|$, compute $\tilde{u}(s)$ by querying $\tilde{v}_1(s)$ and $\tilde{v}_2(s)$. We can sample from \tilde{u} by first sampling $t \in \{1, 2\}$ with probability $\frac{\|\lambda_t \tilde{v}_t\|^2}{\|\lambda_1 \tilde{v}_1\|^2 + \|\lambda_2 \tilde{v}_2\|^2}$, and then taking our sample to be $j \in [n]$ from \tilde{v}_t . The probability of sampling $j \in [n]$ is correct:

$$\begin{aligned} \frac{\|\lambda_1 \tilde{v}_1\|^2}{\|\lambda_1 \tilde{v}_1\|^2 + \|\lambda_2 \tilde{v}_2\|^2} \frac{|\tilde{v}_1(j)|^2}{\|\tilde{v}_1\|^2} + \frac{\|\lambda_2 \tilde{v}_2\|^2}{\|\lambda_1 \tilde{v}_1\|^2 + \|\lambda_2 \tilde{v}_2\|^2} \frac{|\tilde{v}_2(j)|^2}{\|\tilde{v}_2\|^2} \\ = \frac{|\lambda_1 \tilde{v}_1(j)|^2 + |\lambda_2 \tilde{v}_2(j)|^2}{\|\lambda_1 \tilde{v}_1\|^2 + \|\lambda_2 \tilde{v}_2\|^2} = \frac{|\tilde{u}(j)|^2}{\|\tilde{u}\|^2}. \end{aligned}$$

If we pre-process by querying all the norms $\|\tilde{v}_\ell\|$ in advance, we can sample from the distribution over i 's in $\mathcal{O}(1)$ time, using an alias sampling data structure for the distribution (see the problem set), and we can sample from \tilde{v}_t using our assumed access to it, $\text{SQ}_{\varphi_t}(v_t)$. \square

Problem Set 4: Dequantizing QSVT

Before you begin, recall the definitions of sampling and query access for vectors and matrices ($\text{SQ}(v), \text{SQ}(A)$) and *oversampling* and query access ($\text{SQ}_\phi(v), \text{SQ}_\phi(A)$) [CGLLTW22, Definition 3.2]. Below, time complexities are in the word RAM model: basically, assume that reading input numbers, and performing operations on those numbers, cost $\mathcal{O}(1)$.

Problem 4.1 (Errare humanum est...). Suppose we have $\text{SQ}_{\phi_u}(u), \text{SQ}_{\phi_v}(v)$ for vectors u, v . Show that we have $\text{SQ}_\phi(A)$ for $A := uv^\dagger$ and $\phi = \phi_u \phi_v$ with cost $\text{sq}_\phi(A) = \text{sq}_{\phi_u}(u) + \text{sq}_{\phi_v}(v)$.

Problem 4.2 (...sed perseverare (non?) diabolicum.). Suppose we are given a matrix $A \in \mathbb{C}^{m \times m}$ with at most s non-zero entries per row, and suppose all entries are bounded by c . We are given this matrix as a list of non-zero entries $(i, j, A(i, j))$. Show how to perform $\text{SQ}_\phi(A)$ queries for $\phi = c^2 \frac{sm}{\|A\|_F^2}$ with $\text{sq}_\phi(A) = s$.¹⁵ This means that we can run “dequantized” algorithms on sparse matrices with condition number κ ; why doesn’t this imply that QSVT admits no exponential speedup for sparse matrices?

Problem 4.3 (The alias method [Vos91]). Let $p = (p_1, \dots, p_m)$ be a set of probabilities, so $p_i \geq 0$ and $\sum p_i = 1$. Suppose also that all of the p_i ’s are described in binary with $\mathcal{O}(1)$ bits.

1. Suppose we are given a uniformly random number $x \in [0, 1]$ as a stream of random bits. Show how to sample $i \in [m]$ such that $\Pr[\text{sample } \ell] = p_\ell$ in $\mathcal{O}(m)$ operations.
2. Suppose we are given $p = (p_1, \dots, p_m)$ in the following form: we get a list of m probability distributions d_1, \dots, d_m such that $\frac{1}{m}(d_1 + \dots + d_m) = p$ and every d_i is supported on at most two outcomes. Show that we can sample $i \in [m]$ according to p in $\mathcal{O}(1)$ time.
3. Prove that we can convert any distribution p into the form described above. Prove that we can do this in $\mathcal{O}(m)$ time.¹⁶

¹⁵Hint: We immediately have query access to A . What’s a good upper bound that’s easy to sample from?

¹⁶This implies that, if we get time to pre-process, we can get a data structure such that we can respond to $\text{SQ}(v)$ queries in $\mathcal{O}(1)$ time (in the word RAM access model).

5 Quantum-inspired algorithms: sketching and beyond

Recall our definition of oversampling and query access.

Definition 4.4 (Oversampling and query access to a vector). We have ϕ -oversampling and query access to a vector $v \in \mathbb{C}^n$, $\text{SQ}_\phi(v)$, if:

1. we can query for entries of v , $\text{Q}(v)$, and;
2. we have sampling and query access to an “entry-wise upper bound” vector \tilde{v} , $\text{SQ}(\tilde{v})$, where $\|\tilde{v}\|^2 = \phi\|v\|^2$ and $|\tilde{v}(i)| \geq |v(i)|$ for all indices $i \in [n]$.

Let $\text{sq}_\phi(v)$ denote the time cost of any query.

Intuitively speaking, estimators that use \mathcal{D}_v can also use $\mathcal{D}_{\tilde{v}}$ via rejection sampling at the expense of a factor ϕ increase in the number of utilized samples. From this observation we can prove that oversampling access implies an approximate version of the usual sampling access:

Lemma 4.5 (Sampling from oversampling). *Suppose we are given $\text{SQ}_\phi(v)$ and some $\delta \in (0, 1]$. Denote $\overline{\text{sq}}(v) := \phi \text{sq}_\phi(v) \log \frac{1}{\delta}$. We can sample from \mathcal{D}_v with probability $\geq 1 - \delta$ in $\mathcal{O}(\overline{\text{sq}}(v))$ time. We can also estimate $\|v\|$ to ν multiplicative error for $\nu \in (0, 1]$ with probability $\geq 1 - \delta$ in $\mathcal{O}(\frac{1}{\nu^2} \overline{\text{sq}}(v))$ time.*

Lemma 4.6 (Summing SQ-accessible vectors [Tan19, Proposition 4.3]). *Given $\text{SQ}_{\varphi_t}(v_t) \in \mathbb{C}^n$ and $\lambda_t \in \mathbb{C}$ for all $t \in [\tau]$, we have $\text{SQ}_\phi(\sum_{t=1}^\tau \lambda_t v_t)$ for $\phi = \tau \frac{\sum_{t=1}^\tau \varphi_t \|\lambda_t v_t\|^2}{\|\sum_{t=1}^\tau \lambda_t v_t\|^2}$ and $\text{sq}_\phi(\sum_{t=1}^\tau \lambda_t v_t) = \sum_{t=1}^\tau \text{sq}(v_t)$ (after paying $\mathcal{O}(\sum_{t=1}^\tau \text{sq}_{\varphi_t}(v_t))$ one-time pre-processing cost to query for norms).*

5.1 Another vignette: Tools for the matrix-vector product

Assuming A and b are in appropriate data structures in QRAM, we can implement a block-encoding of $A/\|A\|_{\mathbb{F}}$ and prepare copies of $|b\rangle$ efficiently, so we can quantumly produce a sample from Ab in $\mathcal{O}(\frac{\|A\|_{\mathbb{F}}^2 \|b\|^2}{\|Ab\|^2})$ time. We can dequantize this algorithm! Classically, under identical assumptions, we can produce a sample from a v such that $\|v - Ab\| \leq \varepsilon \|Ab\|$ in $\mathcal{O}(\frac{\|A\|_{\mathbb{F}}^4 \|b\|^4}{\varepsilon^2 \|Ab\|^4})$ time, only polynomially slower than quantum.

We note here that a dependence on error ε appears here where it does not in the quantum setting. However, this is not a realizable quantum speedup (except possibly for sampling tasks) since the output is a quantum state: estimating some statistic of the quantum state requires incurring a polynomial dependence on ε . For example, if the goal is to estimate $|\langle v | Ab \rangle|^2$, where v is a given vector, then this can be done with $1/\varepsilon^2$ invocations of a swap test (or $1/\varepsilon$ if one uses amplitude amplification). More generally, distinguishing a state from one ε -far in trace distance requires $\Omega(1/\varepsilon)$ additional overhead, even when given an oracle efficiently preparing that state, so estimating quantities to this sensitivity requires polynomial dependence on ε .

To see this, we first consider a simple case: where b is a constant-sized vector, so $n = \mathcal{O}(1)$. Then we simply wish to sample from a linear combination of columns of A , since $Ab = \sum_{t=1}^n b(t)A(\cdot, t)$. If A is in the QRAM data structure (i.e. storing A^\dagger in Fig. 7), then this means its columns are in the vector QRAM data structures (Fig. 6), so classically

we have sampling and query access to the columns of A , $\text{SQ}(A(\cdot, t))$ for all $t \in [n]$. This implies we have sampling and query access to Ab , up to some overhead.

Given access to a constant number of vectors $\text{SQ}(A(\cdot, 1)), \dots, \text{SQ}(A(\cdot, n))$, we have access to linear combinations $\text{SQ}_\phi(Ab)$ with $\phi = n \frac{\sum_{t=1}^n |b(t)|^2 \|A(\cdot, t)\|^2}{\|Ab\|^2} \leq \frac{n \|A\|_{\mathbb{F}}^2 \|b\|^2}{\|Ab\|^2}$ and $\mathbf{sq}_\phi(Ab) = \mathcal{O}(n)$ (Lemma 4.6; the inequality follows from Cauchy-Schwarz). Finally, from $\text{SQ}_\phi(Ab)$ we can perform approximate versions of all the queries of $\text{SQ}(Ab)$ with a factor ϕ of overhead (Lemma 4.5). This is possible with rejection sampling: given $\text{SQ}_\phi(v)$, pull a sample i from \tilde{v} ; accept it with probability $|v(i)|^2/|\tilde{v}(i)|^2$, and restart otherwise; the output will be a sample from v . In particular, we can sample from Ab in $\phi n = \mathcal{O}(n^2 \frac{\|A\|_{\mathbb{F}}^2 \|b\|^2}{\|Ab\|^2})$ time in expectation, which is good when $n = \mathcal{O}(1)$.

Now, consider when n is too large to iterate over in our linear combination of vectors. In this setting, we can use the *approximate matrix product* property of importance sampling to reduce the number of vectors under consideration. Consider pulling a sample $s \in [n]$ where we sample i with probability $p(i)$. Then $\frac{1}{p(s)} b(s) A(\cdot, s)$, a rescaled random column of A , has expectation $\sum_i b(i) A(\cdot, i) = Ab$. If the sampling distribution is chosen to be $p(i) = \frac{|b(i)|^2}{\|b\|^2}$, an importance sample from $\text{SQ}(b)$, then a variance computation shows that the average of $\tau = \Theta(\frac{\|A\|_{\mathbb{F}}^2}{\varepsilon^2 \|A\|^2})$ copies of this random vector is $\varepsilon \|A\| \|b\|$ -close to Ab with probability ≥ 0.9 (Lemma 5.6). This average, which we denote v , is now a linear combination of only τ columns of A , each of which we have sampling and query access. So, we can use the closure properties mentioned before to get $\text{SQ}_\phi(v)$ for $\phi = \mathcal{O}(\frac{\|A\|_{\mathbb{F}}^2 \|b\|^2}{\|v\|^2})$ and $\mathbf{sq}_\phi(v) = \mathcal{O}(\tau)$, and a sample from v in $\phi \tau = \mathcal{O}(\frac{\|A\|_{\mathbb{F}}^4 \|b\|^2}{\varepsilon^2 \|A\|^2 \|v\|^2})$ time in expectation. Rescaling ε by a factor of $\frac{\|Ab\|}{\|A\| \|b\|}$ gives the result stated above.

5.2 Oversampling and query access to matrices

Now, we formalize the above discussion. We begin by defining (over)sampling and query access to a matrix, and showing some basic properties.

Definition 5.1 (Oversampling and query access to a matrix). For a matrix $A \in \mathbb{C}^{m \times n}$, we have $\text{SQ}(A)$ if we have $\text{SQ}(A(i, \cdot))$ for all $i \in [m]$ and $\text{SQ}(a)$ for $a \in \mathbb{R}^m$ the vector of row norms ($a(i) := \|A(i, \cdot)\|$).

We have $\text{SQ}_\phi(A)$ if we have $\text{Q}(A)$ and $\text{SQ}(\tilde{A})$ for $\tilde{A} \in \mathbb{C}^{m \times n}$ satisfying $\|\tilde{A}\|_{\mathbb{F}}^2 = \phi \|A\|_{\mathbb{F}}^2$ and $|\tilde{A}(i, j)|^2 \geq |A(i, j)|^2$ for all $(i, j) \in [m] \times [n]$.

Let $\mathbf{sq}_\phi(A)$ denote the cost of every query. We omit subscripts if $\phi = 1$.

Lemma 5.2 (Taking outer products of SQ-accessible vectors). *Given vectors $\text{SQ}_{\varphi_u}(u) \in \mathbb{C}^m$ and $\text{SQ}_{\varphi_v}(v) \in \mathbb{C}^n$, we have $\text{SQ}_\phi(A)$ for their outer product $A := uv^\dagger$ with $\phi = \varphi_u \varphi_v$ and $\mathbf{sq}_\phi(A) = \mathbf{sq}_{\varphi_u}(u) + \mathbf{sq}_{\varphi_v}(v)$.*

Proof. We can query an entry $A(i, j) = u(i)v(j)^\dagger$ by querying once from u and v . Our choice of upper bound is $\tilde{A} = \tilde{u}\tilde{v}^\dagger$. Clearly, this is an upper bound on uv^\dagger and $\|\tilde{A}\|_{\mathbb{F}}^2 = \|\tilde{u}\|^2 \|\tilde{v}\|^2 = \varphi_u \varphi_v \|A\|_{\mathbb{F}}^2$. We have $\text{SQ}(\tilde{A})$ in the following manner: $\tilde{A}(i, \cdot) = \tilde{u}(i)\tilde{v}^\dagger$, so we have $\text{SQ}(\tilde{A}(i, \cdot))$ from $\text{SQ}(\tilde{v})$ after querying for $\tilde{u}(i)$, and $\tilde{a} = \|\tilde{v}\|^2 \tilde{u}$, so we have $\text{SQ}(\tilde{a})$ from $\text{SQ}(\tilde{u})$ after querying for $\|\tilde{v}\|$. \square

Using the same ideas as in Lemma 4.6, we can extend sampling and query access of input matrices to linear combinations of those matrices.

Lemma 5.3 (Summing SQ-accessible matrices). *Given $\text{SQ}_{\varphi^{(t)}}(A^{(t)}) \in \mathbb{C}^{m \times n}$ and $\lambda_t \in \mathbb{C}$ for all $t \in [\tau]$, we have $\text{SQ}_{\phi}(A) \in \mathbb{C}^{m \times n}$ for $A := \sum_{t=1}^{\tau} \lambda_t A^{(t)}$ with $\phi = \tau \frac{\sum_{t=1}^{\tau} \varphi^{(t)} \|\lambda_t A^{(t)}\|_{\mathbb{F}}^2}{\|A\|_{\mathbb{F}}^2}$ and $\mathbf{sq}_{\phi}(A) = \sum_{t=1}^{\tau} \mathbf{sq}_{\varphi^{(t)}}(A^{(t)})$ (after paying $\mathcal{O}(\sum_{t=1}^{\tau} \mathbf{sq}_{\varphi^{(t)}}(A^{(t)}))$ one-time pre-processing cost).*

5.3 Sketching to estimate matrix products

We now introduce the workhorse of our algorithms: the matrix sketch. Using sampling and query access, we can generate these sketches efficiently, and these allow us to reduce the dimensionality of a problem, up to some approximation. Most of the results presented in this section are known in the classical sketching literature.

Definition 5.4. For a distribution $p \in \mathbb{R}^m$, we say that a matrix $S \in \mathbb{R}^{s \times m}$ is *sampled according to p* if each row of S is independently chosen to be $e_i / \sqrt{s \cdot p(i)}$ with probability $p(i)$, where e_i is the vector that is one in the i th position and zero elsewhere.

We call S an *importance sampling sketch* for $A \in \mathbb{C}^{m \times n}$ if it is sampled according to A 's row norms a , and we call S a ϕ -*oversampled importance sampling sketch* if it is sampled according to the bounding row norms from $\text{SQ}_{\phi}(A)$, \tilde{a} (or, more generally, from a ϕ -oversampled importance sampling distribution of a).

One should think of S as a description of how to sketch A down to SA . In the standard algorithm setting, computing an importance sampling sketch requires reading all of A , since we need to sample from \mathcal{D}_a . If we have $\text{SQ}_{\phi}(A)$, though, we can efficiently create a ϕ -oversampling sketch S in $\mathcal{O}(s \mathbf{sq}_{\phi}(A))$ time: for each row of S , we pull a sample from \tilde{a} , and then compute $\sqrt{\tilde{a}(i)}$. After finding this sketch S , we have an implicit description of SA : it is a normalized multiset of rows of A , so we can describe it with the row indices and corresponding normalization, $(i_1, c_1), \dots, (i_s, c_s)$.

Further, we can chain sketches using the lemma below, which shows that from $\text{SQ}_{\phi}(A)$, we have $\text{SQ}_{\leq 2\phi}((SA)^{\dagger})$, under a mild assumption on the size of the sketch S . This can be used to find a sketch T^{\dagger} of $(SA)^{\dagger}$. The resulting expression SAT is small enough that we can compute functions of it in time independent of dimension, and so will be key for us. When we discuss sketching A down to SAT , we are referring to the below lemma for the method of sampling T .

Lemma 5.5. *Consider $\text{SQ}_{\varphi}(A) \in \mathbb{C}^{m \times n}$ and $S \in \mathbb{R}^{r \times m}$ sampled according to \tilde{a} , described as pairs $(i_1, c_1), \dots, (i_r, c_r)$. If $r \geq 2\varphi^2 \ln \frac{2}{\delta}$, then with probability $\geq 1 - \delta$, we have $\text{SQ}_{\phi}(SA)$ and $\text{SQ}_{\phi}((SA)^{\dagger})$ for some ϕ satisfying $\phi \leq 2\varphi$. If $\varphi = 1$, then for all r , we have $\text{SQ}(SA)$ and $\text{SQ}((SA)^{\dagger})$.*

The runtimes for $\text{SQ}_{\phi}(SA)$ are $\mathbf{sq}(SA) = \mathbf{sq}(A)$. The runtimes for $\text{SQ}_{\phi}((SA)^{\dagger})$ are $\mathbf{sq}((SA)^{\dagger}) = r \mathbf{sq}(A)$.

Proof. We will only prove this for $\varphi = 1$. We have $\text{SQ}(SA)$. Because the rows of SA are rescaled rows of A , we have SQ access to them from SQ access to A . Because $\|SA\|_{\mathbb{F}}^2 = \|A\|_{\mathbb{F}}^2$ and $\|[SA](i, \cdot)\|^2 = \|A\|_{\mathbb{F}}^2 / r$, we have SQ access to the vector of row norms of SA (pulling samples simply by pulling samples from the uniform distribution).

We have $\text{SQ}((SA)^{\dagger})$. (This proof is similar to one from [FKV04].) Since the rows of $(SA)^{\dagger}$ are length r , we can respond to SQ queries to them by reading all entries of the row and performing some linear-time computation. $\|(SA)^{\dagger}\|_{\mathbb{F}}^2 = \|A\|_{\mathbb{F}}^2$, so we can respond to a norm query by querying the norm of A . Finally, we can sample according to the row

norms of $(SA)^\dagger$ by first querying an index $i \in [r]$ uniformly at random, then outputting the index $j \in [n]$ sampled from $[SA](i, \cdot)$ (which we can sample from because it is a row of A). The distribution of the samples output by this procedure is correct: the probability of outputting j is

$$\frac{1}{r} \sum_{i=1}^r \frac{|[SA](i, j)|^2}{\|[SA](i, \cdot)\|_F^2} = \sum_{i=1}^r \frac{|[SA](i, j)|^2}{\|SA\|_F^2} = \frac{\|[SA](\cdot, j)\|_F^2}{\|SA\|_F^2}. \quad \square$$

We will show below that SA can be used in place of A in matrix products. We begin with a fundamental observation: given sampling and query access to a matrix A , we can approximate the matrix product $A^\dagger B$ by a sum of rank-one outer products. We formalize this with two variance bounds, which we can use together with Chebyshev's inequality.

Lemma 5.6 (Asymmetric matrix multiplication to Frobenius norm error, [DKM06, Lemma 4]). *Consider $X \in \mathbb{C}^{m \times n}$, $Y \in \mathbb{C}^{m \times p}$, and take $S \in \mathbb{R}^{r \times m}$ to be sampled according to $p \in \mathbb{R}^m$ a ϕ -oversampled importance sampling distribution from X or Y . Then,*

$$\begin{aligned} \mathbb{E}[\|X^\dagger S^\dagger SY - X^\dagger Y\|_F^2] &\leq \frac{\phi}{r} \|X\|_F^2 \|Y\|_F^2 \\ \text{and } \mathbb{E} \left[\sum_{i=1}^r \|[SX](i, \cdot)\|_F^2 \|[SY](i, \cdot)\|_F^2 \right] &\leq \frac{\phi}{r} \|X\|_F^2 \|Y\|_F^2. \end{aligned}$$

Proof. To show the first equation, we use that $\mathbb{E}[\|X^\dagger S^\dagger SY - X^\dagger Y\|_F^2]$ is a sum of variances, one for each entry (i, j) , since $\mathbb{E}[X^\dagger S^\dagger SY - X^\dagger Y]$ is zero in every entry. Furthermore, for every entry (i, j) , the matrix expression is the sum of r independent, mean-zero terms, one for each row of S :

$$[X^\dagger S^\dagger SY - X^\dagger Y](i, j) = \sum_{s=1}^r \left([SX](s, i)^\dagger [SY](s, j) - \frac{1}{r} [X^\dagger Y](i, j) \right).$$

So, we can use standard properties of variances to conclude that

$$\begin{aligned} \mathbb{E}[\|X^\dagger S^\dagger SY - X^\dagger Y\|_F^2] &= r \cdot \mathbb{E}[\|[SX](1, \cdot)^\dagger [SY](1, \cdot) - \frac{1}{r} X^\dagger Y\|_F^2] \\ &\leq r \cdot \mathbb{E}[\|[SX](1, \cdot)^\dagger [SY](1, \cdot)\|_F^2] = r \sum_{i=1}^m p(i) \frac{\|X(i, \cdot)^\dagger Y(i, \cdot)\|_F^2}{r^2 p(i)^2} \\ &= \frac{1}{r} \sum_{i=1}^m \frac{\|X(i, \cdot)\|_F^2 \|Y(i, \cdot)\|_F^2}{p(i)} \leq \frac{\phi}{r} \|X\|_F^2 \|Y\|_F^2. \end{aligned}$$

The second other inequality follows by the same computation:

$$\mathbb{E} \left[\sum_{i=1}^r \|[SX](i, \cdot)\|_F^2 \|[SY](i, \cdot)\|_F^2 \right] = r \cdot \mathbb{E}[\|[SX](1, \cdot)\|_F^2 \|[SY](1, \cdot)\|_F^2] \leq \frac{\phi}{s} \|X\|_F^2 \|Y\|_F^2. \quad \square$$

The above result shows that, given $\text{SQ}(X)$, $X^\dagger Y$ can be approximated by a sketch with constant failure probability. If we have $\text{SQ}(X)$ and $\text{SQ}(Y)$, we can make the failure probability exponentially small.¹⁷

¹⁷There are also versions of this statement for operator norm [BT24, Theorem 5.5].

Lemma 5.7 (Approximating matrix multiplication to Frobenius norm error; corollary of [DKM06, Theorem 1]). Consider $X \in \mathbb{C}^{m \times n}$, $Y \in \mathbb{C}^{m \times p}$, and take $S \in \mathbb{R}^{r \times m}$ to be sampled according to $q := \frac{q_1 + q_2}{2}$, where $q_1, q_2 \in \mathbb{R}^m$ are ϕ_1, ϕ_2 -oversampled importance sampling distributions from x, y , the vector of row norms for X, Y , respectively. Then S is a $2\phi_1, 2\phi_2$ -oversampled importance sampling sketch of X, Y , respectively. Further,

$$\Pr \left[\|X^\dagger S^\dagger S Y - X^\dagger Y\|_F < \sqrt{\frac{8\phi_1\phi_2 \log 2/\delta}{r}} \|X\|_F \|Y\|_F \right] > 1 - \delta.$$

Remark 5.8 (Taking products of SQ-accessible matrices). Lemma 5.7 implies that, given $\text{SQ}_{\phi_1}(X)$ and $\text{SQ}_{\phi_2}(Y)$, we can get $\text{SQ}_\phi(M)$ for M a sufficiently good approximation to $X^\dagger Y$, with $\phi \leq \phi_1\phi_2 \frac{\|X\|_F^2 \|Y\|_F^2}{\|M\|_F^2}$. This is an approximate extensibility property for oversampling and query access under matrix products.

Given the above types of accesses, we can compute the sketch S necessary for Lemma 5.7 by taking $p = \mathcal{D}_{\tilde{x}}$ and $q = \mathcal{D}_{\tilde{y}}$, thereby finding a desired $M := X^\dagger S^\dagger S Y$. We can compute entries of M with only r queries each to X and Y , so all we need is to get $\text{SQ}(\tilde{M})$ for \tilde{M} the appropriate bound. We choose $|\tilde{M}(i, j)|^2 := r \sum_{\ell=1}^r |[S\tilde{X}](\ell, i)^\dagger [S\tilde{Y}](\ell, j)|^2$; showing that we have $\text{SQ}(M)$ follows from the proofs of Lemmas 5.2 and 5.3, since M is simply a linear combination of outer products of rows of \tilde{X} with rows of \tilde{Y} . Finally, this bound has the appropriate norm. Notating the rows sampled by the sketch as s_1, \dots, s_r , we have

$$\begin{aligned} \|\tilde{M}\|_F^2 &= r \sum_{\ell=1}^r \|[S\tilde{X}](\ell, \cdot)\|^2 \|[S\tilde{Y}](\ell, \cdot)\|^2 = r \sum_{\ell=1}^r \frac{\|\tilde{X}(s_\ell, \cdot)\|^2 \|\tilde{Y}(s_\ell, \cdot)\|^2}{r^2 \left(\frac{\|\tilde{X}(s_\ell, \cdot)\|^2}{2\|\tilde{X}\|_F^2} + \frac{\|\tilde{Y}(s_\ell, \cdot)\|^2}{2\|\tilde{Y}\|_F^2} \right)^2} \\ &\leq \sum_{\ell=1}^r \frac{\|\tilde{X}(s_\ell, \cdot)\|^2 \|\tilde{Y}(s_\ell, \cdot)\|^2}{r \left(\frac{\|\tilde{X}(s_\ell, \cdot)\| \|\tilde{Y}(s_\ell, \cdot)\|}{\|\tilde{X}\|_F \|\tilde{Y}\|_F} \right)^2} = \|\tilde{X}\|_F^2 \|\tilde{Y}\|_F^2 = \phi_1\phi_2 \|X\|_F^2 \|Y\|_F^2. \end{aligned}$$

5.4 General singular value transformation

So far, we have shown extensibility properties of SQ access like that of the block-encoding. However, as we saw in a problem set, this does not necessarily mean we can efficiently implement all polynomials. We will now discuss the broad approach for producing $p^{(\text{SV})}(A)b$ from $\text{SQ}(A)$ and $\text{SQ}(b)$.

Recall our goal of simulating QSVT: given a matrix $\text{SQ}(A) \in \mathbb{C}^{m \times n}$, a vector $\text{SQ}(b) \in \mathbb{C}^n$, and a polynomial $p : [-1, 1] \rightarrow \mathbb{R}$, compute a description of a vector y such that $\|y - p(A)b\| \leq \varepsilon \|p\|_{[-1, 1]} \|b\|$. Specifically, we aim for our algorithm to run in $\text{poly}(\|A\|_F, \frac{1}{\varepsilon}, d)$ time, and our description to be some sparse vector x such that $y = Ax$, since this allows us to get $\text{SQ}_\phi(y)$.

Given a degree- d polynomial p given in terms of its Chebyshev coefficients a_ℓ (i.e. $p(x) = \sum_{\ell=0}^d a_\ell T_\ell(x)$, where $T_\ell(x)$ is the degree ℓ Chebyshev polynomial) and a value $x \in [-1, 1]$, the Clenshaw recurrence computes $p(x)$. Concretely, our recurrence for p odd (so that $a_\ell = 0$ for ℓ even), is the following.

$$\begin{aligned} q_{(d-1)/2} &= q_{(d+1)/2} = 0 \\ q_k &= 2(2x^2 - 1)q_{k+1} - q_{k+2} + 2a_{2k+1}x \\ p(x) &= \frac{1}{2}(q_0 - q_1) \end{aligned}$$

The scalar recurrences we discuss lift to computing matrix polynomials in a natural way:

$$\begin{aligned} u_{(d-1)/2} &= u_{(d+1)/2} = 0 \\ u_k &= 2(2AA^\dagger - I)u_{k+1} - u_{k+2} + 2a_{2k+1}Ab \\ p(A)b &= \frac{1}{2}(u_0 - u_1) \end{aligned}$$

Each iteration (to get u_k from u_{k+1} and u_{k+2}) can be performed in $\mathcal{O}(\text{nnz}(A))$ arithmetic operations, so this can be used to compute $p(A)b$ in $\mathcal{O}(d \text{nnz}(A))$ operations. We would like to do this approximately in time independent of $\text{nnz}(A)$ and n . We begin by sketching down our matrix and vector: we show that it suffices to maintain a sparse description of u_k of the form $u_k = Av_k$ where v_k is sparse. In particular, we produce sketches $S \in \mathbb{C}^{n \times s}$ and $T \in \mathbb{C}^{t \times m}$ such that

1. $\|AS(AS)^\dagger - AA^\dagger\| \leq \varepsilon$,
2. $\|ASS^\dagger b - Ab\| \leq \varepsilon\|b\|$,
3. $\|TAS(TAS)^\dagger - AS(AS)^\dagger\| \leq \varepsilon$

In the pre-processing phase, we can produce these sketches of size $s, t = \tilde{\mathcal{O}}(\frac{\|A\|_F^2}{\varepsilon^2} \log \frac{1}{\delta})$, and then compute TAS . Since we assume that the input is in the SQ access model, this can be done in time independent of dimension. The approximations hold because of improved versions of Lemma 5.7.

Using these guarantees we can sketch the iterates as follows:

$$\begin{aligned} u_k &= 2(2AA^\dagger - I)u_{k+1} - u_{k+2} + 2a_{2k+1}Ab \\ &= 4AA^\dagger Av_{k+1} - 2Av_{k+1} - Av_{k+2} + 2a_{2k+1}Ab \\ &\approx AS[4(TAS)^\dagger(TAS)v_{k+1} - 2v_{k+1} - v_{k+2} + 2a_{2k+1}S^\dagger b]. \end{aligned} \tag{38}$$

Therefore, we can interpret Clenshaw iteration as the recursion on the dimension-independent term $v_k \approx 4(TAS)^\dagger(TAS)v_{k+1} - 2v_{k+1} - v_{k+2} + 2a_{2k+1}S^\dagger b$, and then applying AS on the left to lift it back to m dimensional space. As desired, we can perform the iteration to produce v_k in $\mathcal{O}(st) = \tilde{\mathcal{O}}(\frac{\|A\|_F^4}{\varepsilon^4} \log^2 \frac{1}{\delta})$ time, which is independent of dimension, at the cost of incurring $\mathcal{O}(\varepsilon(\|v_{k+1}\| + \|v_{k+2}\| + a_{2k+1}\|b\|))$ error. To bound the effect of these per-iteration errors on the final output, we need a stability analysis of the Clenshaw recurrence. We can prove that this gives a $d^3\varepsilon\|p\|_{[-1,1]}\|b\|$ scaling on the final bound, so we rescale ε by a factor of d^2 to get a final runtime of

$$d \frac{\|A\|_F^4}{(\varepsilon/d^3)^4} \log^2 \frac{1}{\delta} = d^{13} \frac{\|A\|_F^4}{\varepsilon^4} \log^2 \frac{1}{\delta}.$$

If we allow linear-time pre-processing, this can be improved further by a factor of ε^2/d^2 [BT24].

Problem Set 5: The power of classical

For this problem set, you'll need the following result about importance sampling sketches, strengthening Lemma 5.7 from the lecture notes.

Lemma 5.9 (Approximating matrix multiplication to spectral norm error [RV07, Theorem 3.1]). *Suppose we are given $A \in \mathbb{R}^{m \times n}$, $\varepsilon > 0$, $\delta \in [0, 1]$, and $S \in \mathbb{R}^{r \times n}$ a ϕ -oversampled importance sampling sketch of A . Then*

$$\Pr \left[\|A^\dagger S^\dagger S A - A^\dagger A\| \lesssim \sqrt{\frac{\phi^2 \log r \log 1/\delta}{r}} \|A\| \|A\|_F \right] > 1 - \delta.$$

Problem 5.1 ([CGLLTW22, Lemma 4.9]). Given $\text{SQ}(A) \in \mathbb{C}^{m \times n}$ and $\varepsilon \in (0, 1]$, we can form importance sampling sketches $S \in \mathbb{R}^{r \times m}$ and $T^\dagger \in \mathbb{R}^{c \times n}$ in $\mathcal{O}(rc \text{sq}(A))$ time. Let σ_i and $\hat{\sigma}_i$ denote the singular values of A and SAT , respectively (where $\hat{\sigma}_i = 0$ for $i > \min(r, c)$). How big does our sketch ($r \times c$) need to be for the following property to hold with probability 0.9?

$$\left(\sum_{i=1}^{\min(m,n)} (\hat{\sigma}_i^2 - \sigma_i^2)^2 \right)^{1/2} \leq \varepsilon \|A\|_F^2. \quad (\star)$$

Problem 5.2 ([CGLLTW22, Corollary 6.12]). We now show that the previous problem implies a dequantization of QPCA [LMR14]. Given a matrix $\text{SQ}(X) \in \mathbb{C}^{m \times n}$ such that $X^\dagger X$ has top k eigenvalues $\{\lambda_i\}_{i=1}^k$, along with a lower bound ν such that $\lambda_1, \dots, \lambda_k \geq \nu$, compute eigenvalue estimates $\{\hat{\lambda}_i\}_{i=1}^k$ such that, with probability 0.9,

$$\sum_{i=1}^k |\hat{\lambda}_i - \lambda_i| \leq \varepsilon \text{tr}(X^\dagger X). \quad (39)$$

What is the runtime of this classical algorithm?

Bonus: how would you design a quantum algorithm to solve this task? Suppose we are given a state prep unitary that prepares a purification of $\rho = X^\dagger X$ (i.e. the vectorized version of X), which implies both the ability to prepare ρ and a 1-block encoding of ρ .

Problem 5.3 ([Van11; GL22]). Suppose we are given SQ access to the vector corresponding to the n -qubit state $|\psi\rangle$ and a description of $H = \frac{1}{s} \sum_{i=1}^s \lambda_a E_a$, where $\lambda_a \in [-1, 1]$ and E_a are Pauli matrices. Show how to estimate $\langle \psi | H^k | \psi \rangle$ to ε error in $\text{poly}(n, s^k, 1/\varepsilon)$ time.

Bonus: prove you can still perform the above estimate if $|\psi\rangle$ is given as a matrix product state with polynomial bond dimension, meaning that, for some $2n \text{poly}(n) \times \text{poly}(n)$ matrices $A_i[0], A_i[1]$, $\psi_{b_1 \dots b_n} = \text{tr}(A_1[b_1] \cdots A_n[b_n])$. Here, $b_1 \cdots b_n$ are bits.

References

- [Aar15] Scott Aaronson. “Read the fine print”. In: *Nature Physics* 11.4 (2015), pp. 291–293. DOI: [10.1038/nphys3272](https://doi.org/10.1038/nphys3272) (page 28).
- [ACQ22] Dorit Aharonov, Jordan Cotler, and Xiao-Liang Qi. “Quantum algorithmic measurement”. In: *Nature Communications* 13.1 (Feb. 2022). DOI: [10.1038/s41467-021-27922-0](https://doi.org/10.1038/s41467-021-27922-0). arXiv: [2101.04634](https://arxiv.org/abs/2101.04634) [quant-ph] (page 32).
- [BCKKS17] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. “Exponential improvement in precision for simulating sparse Hamiltonians”. In: *Forum of Mathematics, Sigma* 5 (2017), e8. DOI: [10.1017/fms.2017.2](https://doi.org/10.1017/fms.2017.2). arXiv: [1312.1414](https://arxiv.org/abs/1312.1414) [quant-ph] (pages 3, 10, 19).
- [BCWW01] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. “Quantum fingerprinting”. In: *Physical Review Letters* 87.16 (Sept. 2001), p. 167902. DOI: [10.1103/physrevlett.87.167902](https://doi.org/10.1103/physrevlett.87.167902). arXiv: [quant-ph/0102001](https://arxiv.org/abs/quant-ph/0102001) [quant-ph] (page 29).
- [BT24] Ainesh Bakshi and Ewin Tang. “An improved classical singular value transformation for quantum machine learning”. In: *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, Jan. 2024, pp. 2398–2453. ISBN: 9781611977912. DOI: [10.1137/1.9781611977912.86](https://doi.org/10.1137/1.9781611977912.86). arXiv: [2303.01492](https://arxiv.org/abs/2303.01492) [quant-ph] (pages 1, 39, 41).
- [CGLLTW22] Nai-Hui Chia, András Pal Gilyén, Tongyang Li, Han-Hsuan Lin, Ewin Tang, and Chunhao Wang. “Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning”. In: *Journal of the ACM* 69.5 (Oct. 2022), pp. 1–72. DOI: [10.1145/3549524](https://doi.org/10.1145/3549524). arXiv: [1910.06151](https://arxiv.org/abs/1910.06151) [cs.DS] (pages 1, 35, 42).
- [CHM21] Jordan Cotler, Hsin-Yuan Huang, and Jarrod R. McClean. “Revisiting dequantization and quantum advantage in learning tasks”. 2021. DOI: [10.48550/ARXIV.2112.00811](https://doi.org/10.48550/ARXIV.2112.00811). arXiv: [2112.00811](https://arxiv.org/abs/2112.00811) [quant-ph] (page 32).
- [Cil+18] Carlo Ciliberto, Mark Herbster, Alessandro Davide Ialongo, Massimiliano Pontil, Andrea Rocchetto, Simone Severini, and Leonard Wossnig. “Quantum machine learning: a classical perspective”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2209 (Jan. 2018), p. 20170551. DOI: [10.1098/rspa.2017.0551](https://doi.org/10.1098/rspa.2017.0551). arXiv: [1707.08561](https://arxiv.org/abs/1707.08561) (page 32).
- [CS19] Andrew M. Childs and Yuan Su. “Nearly optimal lattice simulation by product formulas”. In: *Physical Review Letters* 123.5 (Aug. 2019), p. 050503. ISSN: 1079-7114. DOI: [10.1103/physrevlett.123.050503](https://doi.org/10.1103/physrevlett.123.050503). arXiv: [1901.00564](https://arxiv.org/abs/1901.00564) [quant-ph] (page 3).
- [CSTWZ21] Andrew M. Childs, Yuan Su, Minh Tran, Nathan Wiebe, and Shuchen Zhu. “Theory of Trotter error with commutator scaling”. In: *Physical Review X* 11.1 (Feb. 2021), p. 011020. ISSN: 2160-3308. DOI: [10.1103/PhysRevX.11.011020](https://doi.org/10.1103/PhysRevX.11.011020). arXiv: [1912.08854](https://arxiv.org/abs/1912.08854) [quant-ph] (page 3).

- [CW12] Andrew M. Childs and Nathan Wiebe. “Hamiltonian simulation using linear combinations of unitary operations”. In: *Quantum Info. Comput.* 12.11–12 (Nov. 2012), pp. 901–924. ISSN: 1533-7146. arXiv: [1202.5822 \[quant-ph\]](#) (page 3).
- [DKM06] P. Drineas, R. Kannan, and M. Mahoney. “Fast Monte Carlo algorithms for matrices I: approximating matrix multiplication”. In: *SIAM Journal on Computing* 36.1 (Jan. 2006), pp. 132–157. DOI: [10.1137/s0097539704442684](#) (pages 32, 39, 40).
- [EJ23] Alan Edelman and Sungwoo Jeong. “Fifty three matrix factorizations: a systematic approach”. In: *SIAM Journal on Matrix Analysis and Applications* 44.2 (Apr. 2023), pp. 415–480. DOI: [10.1137/21m1416035](#). arXiv: [2104.08669 \[math.NA\]](#) (page 15).
- [FKV04] Alan Frieze, Ravi Kannan, and Santosh Vempala. “Fast Monte-Carlo algorithms for finding low-rank approximations”. In: *Journal of the ACM* 51.6 (Nov. 2004), pp. 1025–1041. DOI: [10.1145/1039488.1039494](#) (page 38).
- [GL22] Sevag Gharibian and François Le Gall. “Dequantizing the quantum singular value transformation: hardness and applications to quantum chemistry and the quantum pcp conjecture”. In: *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2022. Rome, Italy: Association for Computing Machinery, 2022, pp. 19–32. ISBN: 9781450392648. DOI: [10.1145/3519935.3519991](#) (page 42).
- [GLM08] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. “Quantum random access memory”. In: *Physical Review Letters* 100.16 (2008), p. 160501. DOI: [10.1103/PhysRevLett.100.160501](#). arXiv: [0708.1879](#) (page 30).
- [GSLW19] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. “Quantum singular value transformation and beyond: Exponential improvements for quantum matrix arithmetics”. In: *Proceedings of the 51st ACM Symposium on the Theory of Computing (STOC)*. ACM, June 2019, pp. 193–204. DOI: [10.1145/3313276.3316366](#). arXiv: [1806.01838](#) (pages 1, 3, 4, 7, 8, 10–12, 14, 16, 23, 24, 26, 30, 33).
- [HHKL21] Jeongwan Haah, Matthew B. Hastings, Robin Kothari, and Guang Hao Low. “Quantum algorithm for simulating real time evolution of lattice Hamiltonians”. In: *SIAM Journal on Computing* 52.6 (Jan. 2021), FOCS18-250-FOCS18–284. ISSN: 1095-7111. DOI: [10.1137/18m1231511](#). arXiv: [1801.03922 \[quant-ph\]](#) (page 3).
- [HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. “Quantum algorithm for linear systems of equations”. In: *Physical Review Letters* 103 (15 Oct. 2009), p. 150502. DOI: [10.1103/PhysRevLett.103.150502](#) (pages 24, 28, 29).
- [JR23] Samuel Jaques and Arthur G. Rattew. “QRAM: A survey and critique”. May 17, 2023. arXiv: [2305.10310 \[quant-ph\]](#) (page 30).
- [JW06] Dominik Janzing and Pawel Wocjan. “Estimating diagonal entries of powers of sparse symmetric matrices is BQP-complete”. June 27, 2006. arXiv: [quant-ph/0606229 \[quant-ph\]](#) (page 29).

- [KP17] Jordanis Kerenidis and Anupam Prakash. “Quantum recommendation systems”. In: *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*. 2017, 49:1–49:21. DOI: [10.4230/LIPIcs.ITCS.2017.49](https://doi.org/10.4230/LIPIcs.ITCS.2017.49). arXiv: [1603.08675](https://arxiv.org/abs/1603.08675) (page 28).
- [LC17] Guang Hao Low and Isaac L. Chuang. “Optimal Hamiltonian simulation by quantum signal processing”. In: *Physical Review Letters* 118.1 (Jan. 2017), p. 010501. DOI: [10.1103/PhysRevLett.118.010501](https://doi.org/10.1103/PhysRevLett.118.010501). arXiv: [1606.02685](https://arxiv.org/abs/1606.02685) [quant-ph] (page 3).
- [LC19] Guang Hao Low and Isaac L. Chuang. “Hamiltonian simulation by qubitization”. In: *Quantum* 3 (July 2019), p. 163. DOI: [10.22331/q-2019-07-12-163](https://doi.org/10.22331/q-2019-07-12-163) (pages 3, 10).
- [LGZ16] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. “Quantum algorithms for topological and geometric analysis of data”. In: *Nature Communications* 7.1 (Jan. 2016), p. 10138. DOI: [10.1038/ncomms10138](https://doi.org/10.1038/ncomms10138). arXiv: [1408.3106](https://arxiv.org/abs/1408.3106) (page 28).
- [Llo96] Seth Lloyd. “Universal quantum simulators”. In: *Science* 273.5278 (Aug. 1996), pp. 1073–1078. DOI: [10.1126/science.273.5278.1073](https://doi.org/10.1126/science.273.5278.1073) (page 3).
- [LMR13] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. “Quantum algorithms for supervised and unsupervised machine learning”. 2013. arXiv: [1307.0411](https://arxiv.org/abs/1307.0411) [quant-ph] (page 30).
- [LMR14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. “Quantum principal component analysis”. In: *Nature Physics* 10.9 (July 2014), pp. 631–633. DOI: [10.1038/nphys3029](https://doi.org/10.1038/nphys3029). arXiv: [1307.0401](https://arxiv.org/abs/1307.0401) [quant-ph] (page 42).
- [MRTC21] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. “Grand unification of quantum algorithms”. In: *PRX Quantum* 2 (4 Dec. 2021), p. 040203. DOI: [10.1103/PRXQuantum.2.040203](https://doi.org/10.1103/PRXQuantum.2.040203). arXiv: [2105.02859](https://arxiv.org/abs/2105.02859) [quant-ph] (pages 10, 11, 29).
- [OD21] Davide Orsucci and Vedran Dunjko. “On solving classes of positive-definite quantum linear systems with quadratically improved runtime in the condition number”. In: *Quantum* 5 (Nov. 2021), p. 573. DOI: [10.22331/q-2021-11-08-573](https://doi.org/10.22331/q-2021-11-08-573). arXiv: [2101.11868](https://arxiv.org/abs/2101.11868) [quant-ph] (page 26).
- [Pra14] Anupam Prakash. “Quantum algorithms for linear algebra and machine learning”. PhD thesis. University of California at Berkeley, 2014. URL: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/ECS-2014-211.pdf> (pages 30, 32).
- [PW94] C. C. Paige and M. Wei. “History and generality of the CS decomposition”. In: *Linear Algebra and Its Applications* 208/209 (1994), pp. 303–326. ISSN: 0024-3795. DOI: [10.1016/0024-3795\(94\)90446-4](https://doi.org/10.1016/0024-3795(94)90446-4) (page 15).
- [Ral20] Patrick Rall. “Quantum algorithms for estimating physical quantities using block encodings”. In: *Physical Review A* 102.2 (Aug. 2020), p. 022408. DOI: [10.1103/physreva.102.022408](https://doi.org/10.1103/physreva.102.022408). arXiv: [2004.06832](https://arxiv.org/abs/2004.06832) [quant-ph] (pages 4, 10, 11).

- [RV07] Mark Rudelson and Roman Vershynin. “Sampling from large matrices: an approach through geometric functional analysis”. In: *Journal of the ACM* 54.4 (July 2007), 21–es. ISSN: 0004-5411. DOI: [10.1145/1255443.1255449](https://doi.org/10.1145/1255443.1255449). URL: <https://doi.org/10.1145/1255443.1255449> (page 42).
- [Sch41] A. C. Schaeffer. “Inequalities of A. Markoff and S. Bernstein for polynomials and related functions”. In: *Bull. Amer. Math. Soc.* 47 (1941), pp. 565–579. ISSN: 0002-9904. DOI: [10.1090/S0002-9904-1941-07510-5](https://doi.org/10.1090/S0002-9904-1941-07510-5) (page 25).
- [SV14] Sushant Sachdeva and Nisheeth K. Vishnoi. “Faster algorithms via approximation theory”. In: *Foundations and Trends in Theoretical Computer Science* 9.2 (2014), pp. 125–210. ISSN: 1551-305X. DOI: [10.1561/04000000065](https://doi.org/10.1561/04000000065) (page 23).
- [Tan19] Ewin Tang. “A quantum-inspired classical algorithm for recommendation systems”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC 2019*. ACM Press, 2019, pp. 217–228. DOI: [10.1145/3313276.3316310](https://doi.org/10.1145/3313276.3316310). arXiv: [1807.04271 \[cs.IR\]](https://arxiv.org/abs/1807.04271) (pages 33, 36).
- [Tan22] Ewin Tang. “Dequantizing algorithms to understand quantum advantage in machine learning”. In: *Nature Reviews Physics* 4.11 (Sept. 2022), pp. 692–693. DOI: [10.1038/s42254-022-00511-w](https://doi.org/10.1038/s42254-022-00511-w) (page 1).
- [Tan23] Ewin Tang. “Quantum machine learning without any quantum”. English. PhD thesis. 2023, p. 169. ISBN: 9798380327909. URL: <https://www.proquest.com/dissertations-theses/quantum-machine-learning-without-any/docview/2864097401/se-2> (page 1).
- [TB97] Lloyd N. Trefethen and David Bau III. *Numerical linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997, pp. xii+361. ISBN: 0-89871-361-7. DOI: [10.1137/1.9780898719574](https://doi.org/10.1137/1.9780898719574) (page 26).
- [Tre19] Lloyd N. Trefethen. *Approximation theory and approximation practice, extended edition*. Extended edition [of 3012510]. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2019, pp. xi+363. ISBN: 978-1-611975-93-2. DOI: [10.1137/1.9781611975949](https://doi.org/10.1137/1.9781611975949) (pages 1, 20–22, 27).
- [TT24] Ewin Tang and Kevin Tian. “A CS guide to the quantum singular value transformation”. In: *2024 Symposium on Simplicity in Algorithms (SOSA)*. Society for Industrial and Applied Mathematics, Jan. 2024, pp. 121–143. ISBN: 9781611977936. DOI: [10.1137/1.9781611977936.13](https://doi.org/10.1137/1.9781611977936.13). arXiv: [2302.14324 \[quant-ph\]](https://arxiv.org/abs/2302.14324) (pages 1, 14, 15, 17, 23, 24).
- [Van11] Maarten Van den Nest. “Simulating quantum computers with probabilistic methods”. In: *Quantum Information and Computation* 11.9&10 (Sept. 2011), pp. 784–812. ISSN: 1533-7146. DOI: [10.26421/qic11.9-10-5](https://doi.org/10.26421/qic11.9-10-5). arXiv: [0911.1624 \[quant-ph\]](https://arxiv.org/abs/0911.1624) (page 42).
- [Vos91] Michael D. Vose. “A linear algorithm for generating random numbers with a given distribution”. In: *IEEE Transactions on Software Engineering* 17.9 (1991), pp. 972–975. DOI: [10.1109/32.92917](https://doi.org/10.1109/32.92917) (page 35).

[Woo14] David P. Woodruff. “Sketching as a tool for numerical linear algebra”. In: *Foundations and Trends® in Theoretical Computer Science* 10.1–2 (2014), pp. 1–157. ISSN: 1551-305X. DOI: [10.1561/04000000060](https://doi.org/10.1561/04000000060) (page 32).