Quantum Algorithms 3/21/2023

Lecture 15: Quantum Linear System Solvers

Instructor: Dieter van Melkebeek

Today's lecture is about solving a system of linear equations, which is important for many applications including machine learning and other engineering disciplines, and also an example of using Hamiltonian simulation as a tool. Classically, we can solve these systems in time polynomial in the dimension of the system, but we show that we do better in the quantum setting, even an exponential speedup. We first state the classical problem and the known results, followed by the quantum problem, which is similar except in how the input is given and how we can access the output. For the quantum algorithm, we first discuss the original algorithm for solving a linear system in a quantum setting [Harrow-Hassidim-Lloyd '09], which is not quite as efficient due to its use of eigenvalue estimation, and then we discuss a more recent algorithm with improved accuracy, which makes use of techniques connecting back to the previous lecture on quantum walks.

1 Classical problem and results

Consider an $N \times N$ matrix A and an N-dimensional column vector b over some field $\mathbb{F}$, i.e., $A \in \mathbb{F}^{N\times N}$ and $b \in \mathbb{F}^N$. We are looking for a column vector $x \in \mathbb{F}^N$ such that the equation $Ax = b$ holds. For this discussion, assume that A is invertible so that there is always a unique solution.

1.1 Results assuming infinite precision

Assuming infinite precision (or finite fields), this system can be solved in $O(N^3)$ time using standard methods such as Gaussian elimination. As solving a system of linear equations is computationally equivalent to matrix multiplication, the problem can actually be solved in time $O(N^\omega)$, where ω denotes the exponent for matrix multiplication. Currently, the best known upper bound on ω is about 2.373. Progress on ω is slow, the algorithms become more and more complicated, and the best that can be hoped for is $\omega = 2$ (since all entries need to be read). One example is Strassen's algorithm, which runs in $O(N^{\log_2 7}) \approx O(N^{2.8074})$ time.

In most branches of engineering where linear systems of equations need to be solved, for example civil or chemical engineering, the systems can be very large, but their matrices are usually **sparse**, i.e., most coefficients are 0. More precisely, a matrix A is said to be s-sparse if every row and column contain at most s non-zero entries. Then, as a function of this sparseness s, the problem can instead be solved in $O(N^2 \cdot s)$ time for s-sparse A. Therefore, in practice, real-world systems with hundreds or thousands of components, many of which are zero, can be solved efficiently.

1.2 Results assuming finite precision over $\mathbb{F} = \mathbb{R}$

Over the reals with finite precision ($\mathbb{F} = \mathbb{R}$), however, the number of bits of precision needed has to be taken into account. In this case, the problem can be solved in time $O(N^2 \cdot s \cdot \kappa \cdot \log(1/\epsilon))$ to obtain an approximate solution $\tilde{x}$ with $\|\tilde{x} - x\|_2 \leq \epsilon \|x\|_2$, where ϵ is the maximum relative error, $\log(1/\epsilon)$ is the number of bits of accuracy, and κ is an upper bound for the **condition number** of

A, defined as

$$\kappa(A) \doteq \|A\|_2 \cdot \|A^{-1}\|_2 = \sigma_1/\sigma_N, \tag{1}$$

where σ_1 denotes the largest singular value of A, and σ_N the smallest. Note that in the case of a Hermitian matrix A, $\kappa(A) = |\lambda|_{max}/|\lambda|_{min}$, where $|\lambda|_{max}$ denotes the largest absolute value of an eigenvalue of A, and $|\lambda|_{min}$ the smallest.

In general, the condition number of a numerical problem bounds how much the solution can change when the inputs are changed. If the condition number is high, then small perturbations can be amplified and cause wide variations in the output. However, even with a small condition number, the chosen algorithm can still end up losing a lot of precision. For example, subtracting two large numbers that are almost equal may cause a significant loss in accuracy. From this perspective, some algorithms are better than others.

To gain some intuition on how $\kappa(A)$ defines the condition of a problem, consider a matrix A where the difference between the largest and smallest singular values is large. Solving the system conceptually means computing $A^{-1}b$. When A^{-1} is applied to a vector, some dimensions may be expanded a lot, while others may shrink by a lot. Suppose that the correct right-hand side b lies along a direction of smallest expansion of A^{-1}, and the measurement error on b along a direction of largest expansion of A^{-1}. Since the smallest expansion of A^{-1} is σ_1^{-1}, and the largest expansion σ_N^{-1}, the relative error can get multiplied by a factor of about σ_1/σ_N when going from the right-hand side b to the solution x. The best matrix we can get from this perspective is with the smallest condition number $\kappa = 1$, i.e., with $\sigma_1 = \sigma_N$. These are exactly the unitary matrices and scaled versions.

2 Quantum problem

We consider a variation of the classical problem in the quantum setting. More specifically, we compact the inputs and outputs into superpositions. The invertible matrix $A \in \mathbb{R}^{N\times N}$ is given as a block encoding U_A using ℓ ancilla qubits. The right-hand side b is given as a superposition that can be generated from the known basis state $|0^n\rangle$by a unitary U_b:

$$U_b : |0^n\rangle \mapsto |b\rangle \doteq \frac{1}{\|b\|_2}\sum_{i=0}^{N-1} b_i \, |i\rangle\,, \tag{2}$$

where each b_i is a coefficient of b and $\frac{1}{\|b\|_2}$ is the normalizing factor. We represent this as a unitary U_b because we need multiple copies, and providing it as a unitary operation yields an easy way to generate $|b\rangle$ without requiring multiple copies of it to be supplied. Supplying a U_b also allows us to reflect around $|b\rangle$. Besides our encodings of A and b, we also require an accuracy parameter $\epsilon > 0$.

Likewise, the output is given as a superposition $|\tilde{x}\rangle$ such that $\| \, |\tilde{x}\rangle - |x\rangle \, \|_2 \leq \epsilon$ where x denotes the exact solution the system $Ax = b$. Again, ϵ is the maximum relative error between the exact (normalized) solution $|x\rangle \doteq \frac{1}{\|x\|_2}\sum_{i=0}^{N-1} x_i \, |i\rangle$ and the actual output $|\tilde{x}\rangle$. (Since $|x\rangle$ has 2-norm one, relative and absolute error are the same.) Note that the output of the quantum problem is very different from the classical version; with $|\tilde{x}\rangle$ being a superposition, one can, for example, observe a particular basis state $|i\rangle$ with probability of the absolute value of the corresponding coefficient, squared. However, one cannot just read out the parts of $|\tilde{x}\rangle$, so it is not equivalent to solving the system of equations classically. It means that one needs to run it many times and use some statistics to get the full solution, or even a single component, and even then one can only get estimates for the absolute value squared of the components. In general, this representation is very restrictive, but there are some settings where it suffices. For example, in machine learning, when $|\tilde{x}\rangle$ represents (the square roots of) the weights of a probability distribution and our work is to sample from the distribution, the representation is very appropriate, because all we need to do is just to measure the superposition as this gives us the sample we need.

Note that, because A is a block encoding, we must have that $\|A\|_2 \leq 1$. If this is not the case, we can re-scale A. Additionally, we can assume without loss of generality that A is Hermitian. This is because $Ax = b$ if and only if there exists a y such that

$$A^* y = 0 \tag{3}$$

$$A x = b \tag{4}$$

which can be rewritten as

$$\begin{bmatrix} 0 & A^* \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix} \tag{5}$$

Since A is invertible, y must be 0. Therefore, solving these equations is equivalent to solving $Ax = b$. But note that since the coefficient matrix of equation (5) is Hermitian, and since solving that is equivalent to solving $Ax = b$, solving a system with coefficient matrix A can be reduced to solving one with a Hermitian coefficient matrix.

Theorem 1. *There exists a black-box algorithm with success indicator that solves the quantum linear system for Hermitian A with the following expected cost when provided with a bound $\tilde{\kappa}$ such that $\tilde{\kappa} \geq \sigma_N^{-1} = \|A^{-1}\|_2$:*

- ◦ $q = O(\tilde{\kappa} \operatorname{poly}\log(\tilde{\kappa}/\epsilon))$ controlled applications for U_A and its inverse,
- ◦ $O(\tilde{\kappa}\log(\tilde{\kappa}))$ applications of U_b,
- ◦ $O(\ell q)$ other quantum gates, where ℓ is the number of ancilla qubits used in the block encoding of A.

Note that, since A is given in block encoding form, the largest singular value σ_1 is at most one. This means that $\kappa(A) \doteq \sigma_1/\sigma_N \leq \sigma_1\tilde{\kappa} \leq \tilde{\kappa}$, so $\tilde{\kappa}$ gives an upper bound on the condition number. Additionally, the applications of U_b can be replaced by $O((\tilde{\kappa}\log(1/\epsilon))^2)$ copies of $|b\rangle$, at a multiplicative cost of $O((\tilde{\kappa}^2\log(1/\epsilon))^2)$ for q. Note that $|b\rangle$ is a quantum superposition, so the multiple copies the algorithm needs cannot be produced from a single copy (as per the No Cloning Theorem). We either need a certain number of copies, or else a unitary U_b that allows us to generate ourselves as many copies as we need.

2.1 Sparse Case

As we will see next lecture, in the case of s-sparse A with $\|A\|_{\max} \doteq \max_{i,j}|A_{ij}| \leq 1$, we can construct a block encoding of A/s using $O(1)$ so-called sparse accesses to A, and $O(\log N)$ ancillas and other quantum gates. Then, the construction from the theorem can be run on $A' \doteq A/s$, yielding an algorithm for A using:

- ◦ $O(s \cdot \tilde{\kappa} \cdot \operatorname{poly}\log(s\tilde{\kappa}/\epsilon))$ sparse accesses to A

- $O(s \cdot \tilde{\kappa} \cdot \log(s\tilde{\kappa}))$ applications of U_b
- $O(\log N \cdot s \cdot \tilde{\kappa} \cdot \operatorname{poly}\log(s\tilde{\kappa}/\epsilon))$ other quantum gates.

Recall that the classical algorithm for the classical problem runs in time $O(N^2 \cdot s \cdot \kappa \cdot \log(1/\epsilon))$. This is very similar to the last bullet point above, but notably has a factor of N^2 instead of $\log N$ in the last bullet point. In the case where both the sparseness s and condition number $\tilde{\kappa}$ are small, of order $\operatorname{poly}\log N$, then the quantum algorithm represents an exponential speedup over the classical version. However, do keep in mind that the classical and quantum outputs are not comparable, in that the quantum output is a superposition.

3 Harrow-Hassidim-Lloyd Approach

We start by discussing the Harrow-Hassidim-Lloyd (HHL) approach. This approach ends up with a dependency on the accuracy that is polynomial in $1/\epsilon$ instead of polylogarithmic, which is not good, but we will look at different approaches later.

Recall that A is Hermitian, so it has a full orthonormal basis of eigenvectors: $A\,|\varphi_j\rangle = \lambda_j\,|\varphi_j\rangle$ with $\lambda_j \in \mathbb{R}$. So we can think of the right-hand-side as a linear combination of eigenvectors:

$$|b\rangle = \sum_j \beta_j\,|\varphi_j\rangle\,. \tag{6}$$

The solution can also be written as a linear combination of the same vectors, namely as follows:

$$|x\rangle = \frac{\sum_j \beta_j \lambda_j^{-1}\,|\varphi_j\rangle}{\sqrt{\sum_j |\beta_j \lambda_j^{-1}|^2}} \tag{7}$$

3.1 Harrow-Hassidim-Lloyd Subroutine

We want to transform the register that initially contains (6) into one that contains (7). In order to do so, we will use two ancilla registers, which start and end in the all-zero state:

- One in which we compute approximations to the eigenvalues λ_j. It consists of a qubits, where a denotes the number of bits of absolute accuracy to which we compute the eigenvalues.
- Another one is a register consisting of a single qubit that will aid to transfer our estimates of the eigenvalues to the amplitudes (as inverse factors).

To execute the transformation, HHL performs the following:

1. Eigenvalue estimation:

$$|b\rangle\,|0^a\rangle\,|0\rangle = \sum_j \beta_j\,|\varphi_j\rangle\,|0^a\rangle\,|0\rangle \mapsto \sum_j \beta_j\,|\varphi_j\rangle\,|\lambda_j\rangle\,|0\rangle$$

   Recall that our expression for equation (7) requires the inverse of the eigenvalues λ_j of A. This requires first finding the eigenvalues, which can be done through eigenvalue estimation. After finding each λ_j (more precisely, an approximation to within a bits of absolute accuracy), we use its inverse to scale $|\varphi_j\rangle$.

2. Scaling / Quantum rejection sampling. We would like to perform the following rotation to the last ancilla qubit:

$$|\varphi_j\rangle |\lambda_j\rangle |0\rangle \mapsto |\varphi_j\rangle |\lambda_j\rangle \left(\frac{1}{\lambda_j} |0\rangle + \sqrt{1 - (\frac{1}{\lambda_j})^2} |1\rangle \right)$$

   However, the coefficient $1/\lambda_j$ will always be greater than 1, which is invalid for a 2-norm one vector. Thus the $\tilde{\kappa}$ in the denominator is introduced, as $\tilde{\kappa}$ is exactly defined such that $1/\tilde{\kappa}\lambda_j$ is less than one.

$$|\varphi_j\rangle |\lambda_j\rangle \left(\frac{1}{\tilde{\kappa}\lambda_j} |0\rangle + \sqrt{1 - (\frac{1}{\tilde{\kappa}\lambda_j})^2} |1\rangle \right)$$

   This results in

$$\frac{1}{\tilde{\kappa}} \sum_j \beta_j \lambda_j^{-1} |\varphi_j\rangle |\lambda_j\rangle |0\rangle + |\text{garbage}\rangle |1\rangle$$

   Up to the $1/\tilde{\kappa}$ scaling factor, the $\sum_j \beta\lambda_j^{-1} |\varphi_j\rangle$ is what we are looking for. However, note the presence of $|\lambda_j\rangle |0\rangle$ above; this entanglement may prevent interference, so it needs to be removed. This can be done by running the eigenvalue estimation procedure in reverse.

3. Undo eigenvalue estimation, resulting in

$$\frac{1}{\tilde{\kappa}} \sum_j \beta_j \lambda_j^{-1} |\varphi_j\rangle |0^a\rangle |0\rangle + |\text{other garbage}\rangle |1\rangle$$

   This resets the ancilla qubits back to the zero state.

3.2 Eigenvalue estimation

Eigenvalue estimation cannot be done on A itself if A is not unitary. However, we can instead do it on $U = \exp(iA)$, which can be constructed using Hamiltonian simulation. This U is a unitary matrix and has the same eigenvectors as A. The eigenvalues have been transformed, namely from λ_j to $e^{i\lambda_j}$.

It is sufficient to estimate each eigenvalue of A to within relative error at most ϵ. This means that the eigenvalues of A that are smallest in value must be estimated to within an absolute error of at most $\epsilon/\tilde{\kappa}$. To get this level of accuracy, eigenvalue estimation requires $O(\tilde{\kappa}/\epsilon)$ applications of U, or of a block encoding of an approximation of U to within $\Theta(\epsilon/\tilde{\kappa})$.

This results in a running time cost for the subroutine of $\tilde{O}(\tilde{\kappa}/\epsilon)$ applications of U_A and its inverse, one application of U_b, and $\tilde{O}(\ell\tilde{\kappa}/\epsilon)$ other quantum gates. Note the dependency on $1/\epsilon$ here; this is what ends up causing the polynomial dependency on accuracy for the entire HHL algorithm. To do better, we will have to avoid this eigenvalue estimation.

3.3 Harrow-Hassidim-Lloyd Algorithm

The subroutine described above yields:

$$\frac{1}{\tilde{\kappa}} \sum_j \beta_j \lambda_j^{-1} |\varphi_j\rangle |0\rangle + |\text{something}\rangle |1\rangle$$

As we only care about the first term, we measure the last qubit and hope for a 0. Our probability of success is:

$$\Pr[\text{success}] = \sum_j \left|\frac{\beta_j}{\tilde{\kappa}\lambda_j}\right|^2 \geq \min_j \left|\frac{1}{\tilde{\kappa}\lambda_j}\right| \geq \frac{1}{\tilde{\kappa}^2}$$

Recall that $\sum_j |\beta_j|^2 = 1$ and that $|\lambda_j| \leq 1$.

By using amplitude amplification, running the subroutine $O(\tilde{\kappa})$ times suffices to get a high probability of success (with success indicator). To use amplitude amplification, we need to be able to reflect around the start state $|b\rangle$, which is why we need the unitary U_b that generates the start state. Using this algorithm, the overall expected running time is $\tilde{O}(\tilde{\kappa}^2/\epsilon)$ applications of U_A and its inverse, $O(\tilde{\kappa})$ applications of U_b, and $\tilde{O}(\ell\tilde{\kappa}^2/\epsilon)$ other quantum gates. If we ran this subroutine as a classical Bernoulli experiment, we would have to run the subroutine $O(\tilde{\kappa}^2)$ times to get a high probability of success. Running the subroutine $O(\tilde{\kappa}^2)$ times results in an overall expected running time of $\tilde{O}(\tilde{\kappa}^3/\epsilon)$ applications of U_A and its inverse, $O(\tilde{\kappa}^2)$ applications of U_b, and $\tilde{O}(\ell\tilde{\kappa}^3/\epsilon)$ other quantum gates.

4 Approximate Matrix Inversion Approach

In the previous approach, the part that most greatly impacted the run time was the use of eigenvalue estimation. Eigenvalue estimation had a run-time dependent on $1/\epsilon$. By using approximation methods we are able to improve on this to get a dependence of $\operatorname{poly}\log(1/\epsilon)$. To do this we need to compute a good approximation M of A^{-1}, then compute Mb to get an approximation of x.

We cannot hope to construct M directly within the block encoding framework. Recall that for a block encoding of A to exist, we need that $\|A\|_2 \leq 1$. This implies that $\|M\|_2 \geq 1$, which makes it impossible to block encode within a unitary matrix. To get around this we will construct a block encoding for M/c, for some small c. We want c to be small as it will inversely affect the probability of success of applying the block encoding. A successful application of this block encoding to $|b\rangle$ yields:

$$\frac{M|b\rangle}{\|M|b\rangle\|_2}.$$

Exercise. Show that if $\|M - A^{-1}\|_2 \leq \epsilon$ then $\left\|\frac{M|b\rangle}{\|M|b\rangle\|_2} - |x\rangle\right\|_2 \leq 4\epsilon$.

Note that by the triangle inequality

$$\|M|b\rangle\|_2 \geq \|A^{-1}|b\rangle\|_2 - \|A^{-1}|b\rangle - M|b\rangle\|_2.$$

The probability of a successful application of the block encoding is equal to

$$\left\|\frac{1}{c}M|b\rangle\right\|_2^2 \geq \left(\frac{1}{c}(\|A^{-1}|b\rangle\|_2 - \epsilon)\right)^2 \geq \left(\frac{1-\epsilon}{c}\right)^2$$

By using amplitude amplification we can obtain an approximate solution $|\tilde{x}\rangle$ at an expected cost of $O(c)$ times the combined cost of generating the block encoding for M/c and the cost of U_b.

4.1 Approximate Block Functions

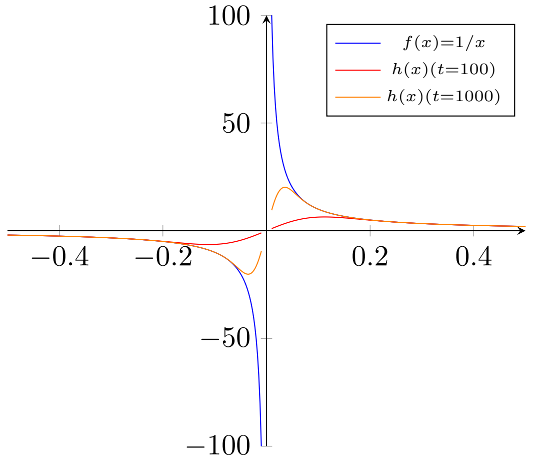
In the general setting for approximating a block function, we are given the block encoding U_A of a Hermitian matrix A and want to find an efficient block encoding for an ϵ-approximation of $f(A)$, which possibly needs to be re-scaled.

The Hamiltonian simulation problem discussed in the previous lecture was equivalent to the approximate block function problem for $f(x) = \exp(ixt)$. This problem was solved with the aid of a function $g(x)$ that approximates $f(x)$ well and such that a block encoding of $g(A)$ can be easily computed from a block encoding of A. It suffices to for g to approximate f well at all the eigenvalues of A, which in the case of a Hermitian matrix are all real, and in the case of a block-encoded matrix are at most 1 in absolute value (as the largest singular is at most 1 and singular values coincide with absolute values of eigenvalues in the case of Hermitian matrices). Thus, for a generic block-encoded Hermitian matrix A, it suffices for $g(x)$ to satisfy $|f(x) - g(x)| \leq \epsilon$ for $x \in [-1, 1]$ to guarantee that $\|g(A) - f(A)\|_2 \leq \epsilon$. In the case of Hamiltonian simulation, our $g(x)$ was the truncated Taylor expansion of $f(x) = \exp(ixt)$ expressed as a linear combination of Chebyshev polynomials. For this $g(x)$ we were able to efficiently construct a block encoding of $g(A)$ given the block encoding of A, by combining techniques from quantum walks with the technique for Linear Combinations of Unitaries (LCU).

Our matrix inversion problem is equivalent to the approximate block function problem of $f(x) = \frac{1}{x}$. It is impossible to get a good approximation for $f(x) = 1/x$ around $x = 0$. However, since we are given a lower bound of $\tilde{\kappa}$ on σ_N^{-1}, we know that the eigenvalues of A are at least $\tilde{\kappa}^{-1}$ in absolute value. For this reason, we can relax the approximation requirement for $g(x)$ from the interval $[-1, 1]$ to $[-1, 1]\backslash(-1/\tilde{\kappa}, 1/\tilde{\kappa})$, i.e., we only need $|f(x) - g(x)| \leq \epsilon$ for $x \in [-1, 1]\backslash(-1/\tilde{\kappa}, 1/\tilde{\kappa})$.

Some possible candidates for $g(x)$ in this scenario are a polynomial approximation expressed as a linear combination of Chebyshev polynomials or a linear combination of harmonics (Fourier expansion). We will be using the polynomial approximation approach.

4.2 Polynomial Approximation



Let us consider $f(x) = \frac{1}{x}$ and $h(x) = \frac{1-(1-x^2)^t}{x}$ for a positive integer t. Note that $h(x)$ is a polynomial of degree $2t - 1$ in x. Note also that for nonzero $x \in [-1, 1]$, $1 - x^2 \in [0, 1)$. Thus, when t grows, the term $(1 - x^2)^t$ goes to zero, which leaves us with just $\frac{1}{x}$. This provides some of the intuition of why this $h(x)$ gives us a good enough approximation of $f(x) = \frac{1}{x}$. Plot above is an example of $f(x)$ and $h(x)$ by assigning 100 and 1000 to t. As t grows, the curve for $h(x)$ becomes more similar with the one for $f(x)$.

Now we need to find a t that gives us a good approximation of $f(x)$.

$$\begin{aligned}
|f(x) - h(x)| &= \left|\frac{(1-x^2)^t}{x}\right| \\
&\le \tilde{\kappa}(1-x^2)^t && [x \ge \tilde{\kappa}^{-1}] \\
&\le \tilde{\kappa}\exp(-x^2)^t && [1+y \le \exp(y) \text{ for } y = -x^2] \\
&= \tilde{\kappa}\exp(-tx^2) \\
&\le \epsilon/2,
\end{aligned}$$

where the last step holds for $x \in [-1,1]\backslash(-1/\tilde{\kappa}, 1/\tilde{\kappa})$, provided $t = \Omega(\tilde{\kappa}^2\log(\tilde{\kappa}/\epsilon))$.

Recall from the previous lecture that $x^k = E_{s_k}[T_{s_k}(x)]$, where s_k is the sum of k independent, uniform ± 1 variables and T_{s_k} is the s_k-th Chebyshev polynomial. Using the binomial theorem we can re-express h as a sum of monomials and plug in the above Chebyshev expansion for each of them:

$$\begin{aligned}
h(x) &= \sum_{k=1}^{t}\binom{t}{k}(-1)^{k+1}x^{2k-1} = \ldots \quad \text{(some manipulations not shown)} \\
&= \sum_{k=1}^{t} c_k T_{2k-1}(x) \quad \text{where } c_k = 4(-1)^{k+1}\Pr[s_{2t} \ge 2k]
\end{aligned}$$

As the coefficient c_k starts to become negligibly small as k exceeds $t/2 + O(\log t)$, we would like to see if we can drop those higher terms. To do that we must determine the error of dropping the higher terms.

$$\begin{aligned}
\left|f(x) - \sum_{k=1}^{d-1} c_k T_{2k-1}(x)\right| &= \left|\sum_{k=d}^{t} c_k T_{2k-1}(x)\right| && (8) \\
&\le \sum_{k=d}^{t} |c_k| && (9) \\
&\le 4t\Pr[s_{2t} \ge 2d] && (10) \\
&\le 4t\exp\left(-\frac{4d^2}{4t}\right) && (11) \\
&\le \epsilon/2 \quad \text{for } d = \Omega(\sqrt{t\log(1/\epsilon)}) && (12)
\end{aligned}$$

In (9) we used the fact that $T_d(x) \in [-1,1]$ for $x \in [-1,1]$, in (10) the above expression for the coefficients c_k, and in (11) the Chernoff bound.

From this we can conclude that $g(x) \doteq \sum_{k=1}^{d} c_k T_{2k-1}(x)$ for $d = \Omega(\tilde{\kappa}\log(\tilde{\kappa}/\epsilon))$, satisfies the requirement $|f(x) - g(x)| \le \epsilon \; \forall x \in [-1,1]\backslash(-1/\tilde{\kappa}, 1/\tilde{\kappa})$.

4.3 Improved Algorithm

So far, we have shown that $g(x)$ satisfies the first requirement, namely that $g(x)$ is a good enough approximation for $f(x)$. What remains to be shown is that a block encoding of $g(A)$ can be efficiently computed from a block encoding of A. To do this we use the LCU method described last lecture to construct the block encoding for M/c where $M \doteq g(A) = \sum_{k=1}^{d} c_k T_{2k-1}(A)$ and $c = \sum_{k=1}^{d} |c_k| \leq 4d$. For the bound for c, recall that each $|c_k|$ term is four times a probability and there are d terms. Each individual term $T_{2k-1}(A)$ is implemented using the quantum walk method described last lecture. Using this polynomial approximation for the block encoding of A^{-1}, we are able to construct a more efficient algorithm for solving a linear system in a quantum setting.

The resulting block encoding uses $O(d) = O(\tilde{\kappa} \log(\tilde{\kappa}/\epsilon))$ applications of U_A and its inverse, and $O(\ell d)$ other quantum gates. The resulting quantum linear system solver has an expected cost of $O(c)$ times the combined cost of the block encoding for M/c and the cost of U_b. This yields an overall cost of $q = O(cd) = O(d^2) = O(\tilde{\kappa}^2 \log^2(\tilde{\kappa}/\epsilon))$ applications of U_A and its inverse, $O(c) = O(\tilde{\kappa} \log(\tilde{\kappa}/\epsilon))$ applications of U_b, and $O(\ell q)$ other quantum gates. One factor of $\tilde{\kappa}$ in q can be shaved off by using so-called variable-time amplitude amplification at only a polylog cost in $1/\epsilon$. This technique (which we will not cover) leverages the fact that the time bounds are generated using the worst case scenarios for the largest eigenvalue and smallest eigenvalue. As both of these situations cannot actually occur simultaneously, we can exploit this leeway algorithmically.