On the Problem of Approximating the Eigenvalues of Undirected Graphs in Probabilistic Logspace

Dean Doron * and Amnon Ta-Shma **

The Blavatnik School of Computer Science,
Tel-Aviv University, Israel 69978
deandoron@mail.tau.ac.il, amnon@tau.ac.il

Abstract. We introduce the problem of *approximating* the eigenvalues of a given stochastic/symmetric matrix in the context of classical space-bounded computation. The problem can be *exactly* solved in $\mathsf{DET} \subseteq \mathsf{NC}^2$. Recently, it has been shown that the approximation problem can be solved by a *quantum* logspace algorithm. We show a *probabilistic* logspace algorithm that solves the problem with *constant* accuracy. The result we obtain falls short of achieving the polynomially-small accuracy that the quantum algorithm achieves. Thus, at our current state of knowledge, we can achieve polynomially-small accuracy with quantum logspace algorithms, constant accuracy with probabilistic logspace algorithms, and no non-trivial result is known for deterministic logspace algorithms.
Our work raises two challenges. First, a derandomization challenge, trying to achieve a deterministic algorithm approximating eigenvalues with some non-trivial accuracy. Second, a de-quantumization challenge trying to decide whether the quantum logspace model is strictly stronger than the classical probabilistic one or not. We therefore believe the problem of approximating the eigenvalues of an undirected graph is not only natural and important by itself, but also important for understanding the relative power of deterministic, probabilistic and quantum logspace computation.

1 Introduction

One of the most basic questions in complexity theory is whether randomness buys extra computational power or not. In the space-bounded model, Nisan [1] constructed a pseudo-random generator (PRG) against logarithmic space-bounded non-uniform algorithms that uses seed length $O(\log^2 n)$. Using that he showed BPL is contained in the class having simultaneously polynomial time and $O(\log^2 n)$ space. Saks and Zhou [2] showed BPL is contained in $\mathsf{DSPACE}(\log^{1.5} n)$.

* Supported by the Israel science Foundation grant no. 994/14 and by the United States – Israel Binational Science Foundation grant no. 2010120.
** Supported by the Israel science Foundation grant no. 994/14 and by the United States – Israel Binational Science Foundation grant no. 2010120.

Reingold [3] showed undirected st-connectivity (which was shown to be in RL by [4]) already belongs to L. These results seem to indicate that randomness does not add additional power to the model and many conjecture that in fact BPL = L. Yet, we currently do not know a PRG with seed length $o(\log^2 n)$, nor a general derandomization result that simultaneously uses $o(\log^2 n)$ space and polynomial time.

One can look up and ask which upper bounds we know on BPL. We then know the following:

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{RL} \subseteq \mathsf{NL} \subseteq \mathsf{DET} \subseteq \mathsf{NC}^2 \subseteq \mathsf{DSPACE}(O(\log^2 n)),$$

where DET is the class of languages that are NC^1 Turing–reducible to the problem *intdet* of computing the determinant of an integer matrix (see [5] for a definition of DET). As it turns out, many important problems in linear algebra, such as inverting a matrix, or equivalently, solving a set of linear equations are in DET, and often complete for it (see, e.g., [5]). The fact that $\mathsf{NL} \subseteq \mathsf{DET}$ is due to [5] who showed that the directed connectivity problem, STCON is reducible to *intdet*. $\mathsf{DET} \subseteq \mathsf{NC}^2$ follows from Csansky's algorithm [6] for the parallel computation of the determinant. In addition to the above we also know that $\mathsf{BPL} \subseteq \mathsf{DET}$ (e.g., using the fact that matrix powering is DET complete).

While matrix powering is complete for DET, *approximating* matrix powering of *stochastic* matrices is in BPL. To see that, assume A represents a stochastic matrix. Then one can approximate $A^k[s,t]$ by estimating the probability a random walk over A starting at s reaches t after k steps. Conversely, it is possible to convert a BPL machine to a stochastic operator A such that the probability the machine moves from s to t in k steps is $A^k[s,t]$, see, e.g., [1]. Thus, in a sense, approximating matrix-powering of stochastic operators is complete for BPL.

In 1999, Watrous [7] defined the model of quantum logspace computation, and proved several facts on it. The definition was modified several times, see, [8]. Roughly speaking, a language is in BQL if there exists an L–uniform family of quantum circuits solving the language with only $O(\log n)$ qubits. The quantum circuits are over some universal basis of gates (e.g., CNOT, HAD, T) plus intermediate measurements (that in particular may simulate a stream of random coins). For details we refer the reader to [8, 9]. The works of Watrous, van Melkebeek and Watson showed that BQL is also contained in NC^2.

Recently, it was shown in [9], building on an earlier work by [10], that it is possible to *approximate* the singular value decomposition (SVD) of a given linear operator in BQL. This also implies that it is possible to approximately invert a matrix in BQL. In a sense, this is an analogue of what is known regarding the permanent. Computing the permanent exactly is #P complete but approximating it is in BPP. Similarly, computing the SVD exactly is DET hard while approximating it is in BQL. Given the above picture, it is natural to ask whether the above is already true in the classical setting. Namely,

Question 1. (Main question) Is it possible to approximate the SVD of an arbitrary linear operator in BPL?

A slightly less ambitious question is whether it is possible to approximate the SVD of a specific class of operators, e.g., Hermitian operators.

In this work we take a first step on this question. One way of doing this is "de-quantumizing" the quantum algorithm. Ben-Or and Eldar [11] recently de-quantumized the SVD quantum algorithm and obtained a classical probabilistic algorithm for inverting matrices that achieves the state of the art running time, using a completely new approach that is derived from the quantum algorithm. We would like to do the same in the space-bounded model. Motivated by this we try to de-quantumize the quantum algorithm and get a classical probabilistic low-space algorithm. For simplicity, we restrict the discussion in the introduction to Hermitian operators only. We define the following promise problem:

Definition 1. *(The promise problem $EV_{\alpha,\beta}$) The input is a stochastic, Hermitian matrix A, $\lambda \in [-1,1]$ and $\alpha < \beta$.*

Yes instances *: There is an eigenvalue λ_i of A such that $|\lambda_i - \lambda| \leq \alpha$.*
No instances *: All eigenvalues of A are β–far from λ.*

The BQL algorithm solves the above problem for any Hermitian operator A whose eigenvalues are τ–separated, for, say, $\tau = n^{-c}$, $\alpha = \frac{\tau}{4}$ and $\beta = 2\alpha$. That is, the quantum algorithm can handle any polynomially small accuracy. With such accuracy one can turn the solution of the promise problem to a procedure approximating the whole spectrum.

We develop a BPL algorithm that follows the main idea of the quantum algorithm, and in that sense we de-quantumize the quantum algorithm, but we achieve much worse parameters. Specifically, we prove that the promise problem $EV_{\alpha,\beta}$ belongs to BPL, for *constant* parameters $\alpha < \beta$. Thus, on the one hand the result is disappointing because the quantum algorithm does so much better and can handle polynomially small gaps. On the other hand, we remark that we do not know how to achieve even constant approximation with a deterministic logspace algorithm. We are not aware of many natural promise problems in BPL that are not known to be in L. This paper shows $EV_{\alpha,\beta}$ is such a promise problem.

1.1 Our technique

The usual way of describing the quantum algorithm is that the algorithm applies quantum phase estimation on the completely mixed state. The completely mixed state is a uniform mixture of the pure states that are formed from the eigenvectors of A, and on each such eigenvector, the quantum phase estimation estimates the corresponding eigenvalue. Thus, if the procedure can be run in (quantum) logarithmic space, we essentially sample a random eigenvector/eigenvalue pair, and from that we can approximately get the SVD decomposition of A.

Another (less standard) way of viewing the quantum algorithm is that it manipulates the eigenvalues of an input matrix A without knowing the decomposition of A to eigenvectors and eigenvalues. This can be done using the simple fact that if $\lambda_1, \ldots, \lambda_n$ are the roots of the characteristic polynomial of A, and if p is an arbitrary univariate polynomial, then $p(\lambda_1), \ldots, p(\lambda_n)$ are the roots of

the characteristic polynomial of the matrix $p(A)$. The probability the algorithm measures λ is proportional to Tr $(p(A))$, where p is a shift of the Fejér kernel by λ (see Section 5). Applying p on A amplifies the eigenvalues that are close to λ to a value close to 1, and damps eigenvalues far from λ close to 0. Thus, Tr $(p(A))$ approximately counts the number of eigenvalues close to λ.

We would like to follow the same approach but with a probabilistic algorithm rather than a quantum one. We say a matrix A is *simulatable* if a probabilistic logspace algorithm can approximate $A^k[s,t]$ for any k polynomial in n and with polynomially-small accuracy (see Definition 3 for the exact details). From the discussion above it is clear that if A is the transition matrix of a (directed or undirected) graph then A is simulatable (see Lemma 1). We first ask what other matrices are simulatable? We show that even non-stochastic matrices A, even with negative or complex entries, are simulatable as long A has infinity norm at most 1, namely, those matrices A for which all rows $i \in [n]$ have ℓ_1 norm at most 1, $\sum_j |A[i,j]| \leq 1$.

If A is simulatable and the coefficients of $p(x) = \sum_i c_i x^i$ are not too large (i.e., only polynomially large in n), then we can approximate in BPL the matrix $p(A) = \sum_i c_i A^i$. In particular, we can also approximate Tr $(p(A))$. By taking p to be a threshold polynomial with degree logarithmic in n (that guarantees the size of the coefficients c_i is polynomial in n) and a threshold around λ, we can solve $EV_{\alpha,\beta}(A)$ for constants $\alpha < \beta$ (see Section 4).

There are many other possible candidate functions for a threshold polynomial p. However, we prove in Theorem 2 that no polynomial can do significantly better than a threshold polynomial. The reason the quantum algorithm works better is because it is able to take p up to some polynomial degree (rather than logarithmic degree) not worrying about the (quite large) size of the coefficients, thus leading to much better accuracy. The quantum algorithm also has the advantage that it works for any normal operator A, not necessarily stochastic or simulatable.

Thus, the algorithm we give for $EV_{\alpha,\beta}$ is simple: Approximate Tr $(p(A))$ to a simple logarithmic degree polynomial p. Nevertheless, we believe it features a new component that has not been used before by probabilistic space-bounded algorithms. An algorithm that takes a random walk on a graph and takes a decision based on the walk length and connectivity properties of the graph (as, e.g., [4]) works with some power of the input matrix A. More generally, such an algorithm can work with a convex combination of powers of the input matrix (by probabilistically choosing which power to take). The algorithm we present utilizes *arbitrary* (positive or *negative*) combinations of matrix powers and we believe it is a crucial feature of the solution. We are not aware of previous BPL algorithms using such a feature.

1.2 A short discussion

We believe the problem of approximating the eigenvalues of an undirected graph is natural and important. Also, at our current state of knowledge, it simultaneously separates deterministic, probabilistic and quantum complexity: In BQL we

can solve it with polynomially-small accuracy, in BPL with constant accuracy and in L we do not know how to solve it at all. Thus it poses several challenges:

- First, there is the natural question of whether one can approximate eigenvalues in BPL with better accuracy. A positive answer would imply BPL approximations to many important linear algebra problems that are currently only known to be in NC^2. A negative answer would imply a separation between BQL and BPL.
- Second, it raises the natural question of derandomization. Can one design a *deterministic* algorithm approximating eigenvalues to constant accuracy?

We believe the solution of this problem is not only important by itself, but may also shed new light on the strengths and weaknesses of the space-bounded model, and the relative strengths of the deterministic, probabilistic and quantum models of space-bounded computation.

2 Preliminaries

A deterministic space-bounded Turing machine has three semi-infinite tapes: an *input tape* (that is read-only); a *work tape* (that is read-write) and an *output tape* (that is write-only and uni-directional). The space complexity of the machine is the number of cells on the work tape. The running time of a space-bounded Turing machine with $s(n) \geq \log n$ space complexity is bounded by $2^{O(s(n))}$ time. A *probabilistic* space-bounded Turing machine is similar to the deterministic machine (and in particular we require it always halts within $2^{O(s(n))}$ time) except that it can also toss random coins. One convenient way to formulate this is by adding a fourth semi-infinite tape, the *random-coins tape*, that is read-only, unidirectional and is initialized with perfectly uniform bits. We are only concerned with bounded-error computation: We say a language is accepted by a probabilistic Turing machine if for every input in the language the acceptance probability is at least 2/3, and for every input not in the language it is at most 1/3. As usual, the acceptance probability can be amplified as long as there is some non-negligible gap between the acceptance probability of yes and no instances.

Definition 2. *A language is in* $\mathsf{BPSPACE}(s(n))$ *if it is accepted by a probabilistic space bounded TM with space complexity* $s(n)$. $\mathsf{BPL} = \cup_c \mathsf{BPSPACE}(c \log n)$.

Often we are interested in computing a *value* (e.g., an entry in a matrix with integer values or the whole matrix) and are only able to approximate it with a probabilistic machine. More precisely, assume there exists some value $u = u(x) \in \mathbb{R}$ that is determined by the input $x \in \{0,1\}^n$. We say a probabilistic TM $M(x,y)$ (ε, δ)–approximates $u(x)$ if:

$$\forall_{x \in \{0,1\}^n} \quad \Pr_y \left[|M(x,y) - u(x)| \geq \varepsilon \right] \leq \delta$$

For a positive integer n, we denote $[n] = \{1, \ldots, n\}$. For a matrix $A \in \mathbb{C}_{n \times n}$, the operator norm corresponding to the ℓ_p norm is $\|A\|_p = \max_{\|v\|_p = 1} \|Av\|_p$.

The special case of $p = 2$, the spectral norm, is also the largest singular value of the operator A. Also, $\|A\|_\infty$ is

$$\|A\|_\infty = \max_r \sum_j |A_{r,j}|.$$

We can view a directed or undirected graph $G = (V, E)$ over n vertices, as a linear operator that describes the transition probabilities of a random walk on G. Specifically, let $\tilde{A}$ be the adjacency matrix of the graph. Let D be a diagonal matrix D with $D(i,i) = d_{\text{out}}(v_i)$ for every $i \in [n]$. We always work with graphs that have no sinks, so $d_{\text{out}}(v_i) \neq 0$ for every $i \in [n]$. Then the *transition matrix* of G is the linear operator $A = D^{-1}\tilde{A}$. Notice that A is stochastic and corresponds to a random walk on G. It is well known that if A is a stochastic matrix then all its eigenvalues have absolute value at most 1. Also, if G is an undirected (possibly irregular) graph then its transition matrix A is diagonalizable with real eigenvalues in the range $[-1, 1]$. [1]

3 Simulatable matrices

A random walk on a graph G (or its transition matrix A) can be simulated by a probabilistic logspace machine. As a consequence, a probabilistic logspace machine can approximate powers of A well. Here we try to extend this notion to arbitrary linear operators A, not necessarily stochastic. We say a matrix A is *simulatable* if any power of it can be approximated by a probabilistic algorithm running in small space. Formally:

Definition 3. *We say that a family of matrices $\mathcal{A}$ is* simulatable *if there exists a probabilistic algorithm that on input $A \in \mathcal{A}$ of dimension n with $\|A\| \leq poly(n)$, $k \in \mathbb{N}$, $s, t \in [n]$, runs in space $O(\log \frac{nk}{\varepsilon\delta})$ and (ε, δ)–approximates $A^k[s,t]$.*

As expected,

Lemma 1. *The family of transition matrices of (directed or undirected) graphs is simulatable.*

Proof. Let $G = (V, E)$ be a graph with n vertices and let A be its transition matrix. Let $k \in \mathbb{N}$, $s, t \in [n]$ and $\delta, \varepsilon > 0$. Consider the algorithm that on input k, s, t, takes T independent random walks of length k over G starting at vertex s. The algorithm outputs the ratio of walks that reach vertex t. Let Y_i be the

[1] To see that notice that the adjacency matrix $\tilde{A}$ is symmetric (because the graph is undirected) but the transition matrix $A = D^{-1}\tilde{A}$ is not symmetric when the graph is irregular. Yet, consider the matrix $L = D^{-1/2}\tilde{A}D^{-1/2}$. L is symmetric and thus has an eigenvector basis with real eigenvalues. $A = D^{-1/2}LD^{1/2}$ is conjugate to L and thus is diagonalizable and has the same eigenvalues. As A is stochastic its eigenvalues are in the range $[-1, 1]$.

random value that is 1 if the i-th trial reached t and 0 otherwise. Then, for every i, $\mathbb{E}[Y_i] = A^k[s,t]$. Also, $Y_1, \ldots, Y_T$ are independent. By Chernoff,

$$\Pr[|\frac{1}{T}\sum_{i=1}^{T} Y_i - A^k[s,t]| \geq \varepsilon] \leq 2e^{-2\varepsilon^2 T}$$

Taking $T = \text{poly}(\varepsilon^{-1}, \log \delta^{-1})$, the error probability (i.e., getting an estimate that is ε far from the correct value) is at most δ. Altogether, the algorithm runs in space $O(\log(Tnk|E|)) = O(\log(nk\varepsilon^{-1}) + \log\log\delta^{-1})$, assuming $|E| = \text{poly}(n,k)$.

Intuitively, any stochastic matrix corresponds to a walk on some directed graph. A technical issue is that the entries of the matrix might have high precision beyond our small space capabilities. We prove:

Lemma 2. *The family of stochastic matrices is simulatable.*

In fact, this can be further generalized to any real matrix A (with possibly negative entries) with infinity norm at most 1.

Lemma 3. *The family of real matrices with infinity norm at most* 1 *is simulatable.*

The proofs of both lemmas are omitted.

4 Approximating eigenvalues with constant accuracy

In this section we prove:

Theorem 1. *There exists a probabilistic algorithm that on input a stochastic matrix B with real eigenvalues in $[0,1]$, constants $\beta > \alpha > 0$ and $\lambda \in [0,1]$ such that:*

- *There are d eigenvalues λ_i satisfying $|\lambda - \lambda_i| \leq \alpha$,*
- *All other eigenvalues λ_i satisfy $|\lambda - \lambda_i| \geq \beta$,*

outputs d with probability at least $2/3$. Furthermore the algorithm runs in probabilistic space $O(\log n)$.

We remark that Theorem 1 covers the case of transition matrices of undirected graphs. As mentioned earlier, a transition matrix A of an undirected graph has an eigenvector basis with real eigenvalues in the range $[-1,1]$. Taking $B = \frac{1}{2}A + \frac{1}{2}I_{n\times n}$ we get a stochastic matrix with eigenvalues in the range $[0,1]$, and whose eigenvectors are in a natural one-to-one correspondence with A's eigenvalues.

Proof. (Of Theorem 1) The input to the algorithm is $n, B, \lambda, \alpha, \beta$. We assume a univariate polynomial $p(x) = \sum_{i=0}^{M} c_i x^i$ with the following properties:

– p has a sharp peak around λ, i.e., $p(x) \geq 1 - \eta$ for $x \in [\lambda - \alpha, \lambda + \alpha]$ and $p(x) \leq \eta$ for $x \in [0,1] \setminus (\lambda - \beta, \lambda + \beta)$, where $\eta = \eta(n) = n^{-2}$.
– p can be computed in L. Formally, $M = \deg(p)$ and $|c_i|$ are at most $\mathrm{poly}(n)$ and for every i, c_i can be computed (exactly) by a deterministic Turing machine that uses $O(\log n)$ space.

In the next subsection we show how to obtain such a polynomial p with $M = 32(\beta - \alpha)^{-2} \log n$ and $|c_i| \leq 2^{O(M)}$.

Choose $\varepsilon = \frac{1}{n}$ and $\delta = \frac{1}{3}$. Set $\varepsilon' = \varepsilon \cdot 2^{-2M}$ and $\delta' = \delta \cdot 2^{-M}$. The output of the algorithm is the integer closest to

$$R = \sum_{i=0}^{M} c_i \cdot \mathrm{TP}(B, n, i, \varepsilon', \delta')$$

where TP is the probabilistic algorithm guaranteed by Lemma 2 that (ε', δ')–approximates Tr (B^i).

It is easy to check that:

Claim. $\Pr[|R - \mathrm{Tr}\ (p(B))| \geq \varepsilon] \leq \delta$.

As Tr $(p(B)) = \sum_{i=1}^{n} p(\lambda_i)$, $\Pr[|R - \sum_{i=1}^{n} p(\lambda_i)| \geq \varepsilon] \leq \delta$. However, $p(\lambda_i)$ is large when λ_i is α–close to λ and small when it is β–far from λ, and we are promised that *all* eigenvalues λ_i are either α–close or β–far from λ. Thus,

$$|\mathrm{Tr}\ (p(B)) - d| \leq n\eta.$$

Altogether, except for probability δ, $|R - d| \leq \varepsilon + n\eta \leq \frac{1}{3}$, and the nearest integer closest to R is d. The correctness follows. It is also straightforward to check that the space complexity is $O(\log(n\varepsilon^{-1}\delta^{-1})) = O(\log n)$.

The constant accuracy we achieve is far from being satisfying. The matrix B has n eigenvalues in the range $[0,1]$, so the average distance between two neighboring eigenvalues is $1/n$. Thus, the assumption that there is an interval of length $\beta - \alpha$ with no eigenvalue is often not true. The desired accuracy we would like to get is $o(1/n)$. Having such accuracy would enable outputting an approximation of the whole spectrum of B, using methods similar to those in [9], thus getting a true classical analogue to the quantum algorithm in [9]. However, we do not know how to achieve subconstant accuracy. The question whether better accuracy is possible in BPL is the main problem raised by this work.

4.1 Using the symmetric threshold functions

There are several natural candidates for the function p above. In this subsection we use the threshold function to obtain such a function p. For $\lambda = \frac{k}{M}$ for some integers k and M, define:

$$p_\lambda(x) = \sum_{i=k}^{M} \binom{M}{i} x^i (1-x)^{M-i}.$$

p_λ approximates well the threshold function $\mathbf{Th}_\lambda(x) : [0,1] \to \{0,1\}$ that is one for $x \geq \lambda$ and zero otherwise. Specifically, using the Chernoff bound, we obtain:

Lemma 4. *Let $x \in [0,1]$. $p_\lambda(x)$ approximates $\mathbf{Th}_\lambda(x)$ over $[0,1]$ with accuracy $(\xi(\varepsilon))^{Mx}$, where $\varepsilon = \frac{\lambda - x}{x}$ and $\xi(\varepsilon) = \frac{e^\varepsilon}{(1+\varepsilon)^{1+\varepsilon}}$.*

As a polynomial in x, $p_\lambda(x) = \sum_{i=0}^{M} c_i x^i$ with $c_i = (-1)^i \sum_{j=\lambda M}^{i} \binom{M}{j}\binom{M-j}{i-j}(-1)^j$ and therefore $|c_i| \leq \sum_{j=\lambda M}^{i} \binom{M}{j}\binom{M-j}{i-j} \leq M\binom{M}{M/2}^2 = 2^{O(M)}$. Furthermore, c_i can be computed (exactly) by a deterministic Turing machine that uses $O(M)$ space by simply running through the loop over j, each time updating the current result by $(-1)^j\binom{M}{j}\binom{M-j}{i-j}$.

To obtain our polynomial p, define p as the difference between the threshold polynomial around $\lambda + \Delta$ and the threshold polynomial around $\lambda - \Delta$,

$$p(x) = p_{\lambda-\Delta}(x) - p_{\lambda+\Delta}(x)$$

where $M = 32(\beta - \alpha)^{-2} \log n$ and $\Delta = (\alpha + \beta)/2$. It is easy to check that

Lemma 5. *$p(x) \geq 1 - n^{-2}$ for every x that is α–close to λ (i.e., $|x - \lambda| < \alpha$) and $p(x) \leq n^{-2}$ for every x that is β–far from λ (i.e., $|x - \lambda| \geq \beta$).*

4.2 The limitation of the technique

In this section, we prove the accuracy of the above technique cannot be enhanced merely by choosing a different polynomial p. Approximating threshold functions by a polynomial is well-studied and well understood (see, for example, [12–14] and references therein). However, we need to adapt this work to our needs because we have an additional requirement that the magnitude of the polynomial's coefficients is small.

We start by formalizing the properties of p that were useful to us. We say that $\mathcal{P} = \{p_{\lambda,n}\}_{\lambda\in[0,1],n\in\mathbb{N}}$ is a family of polynomials if for every $\lambda \in [0,1]$ and $n \in \mathbb{N}$, $p_{\lambda,n}$ is a univariate polynomial with coefficients in $\mathbb{R}$.

Definition 4. *(Small family) Let $\mathcal{P}$ be a family of polynomials and fix $\lambda \in [0,1]$. For every $n \in \mathbb{N}$, write $p_{\lambda,n}(x) = \sum_{i=0}^{\deg(p_{\lambda,n})} c_{\lambda,n,i} x^i$. We say the family is $s(n)$–small if,*

- $\deg(p_{\lambda,n}) \leq 2^{s(n)}$,
- *For every $0 \leq i \leq \deg(p_{\lambda,n})$, $|c_{\lambda,n,i}| \leq 2^{s(n)}$, and*
- *There exists a deterministic Turing machine running in space $s(n)$ that outputs $c_{\lambda,n,0}, \ldots, c_{\lambda,n,\deg(p_{\lambda,n})}$.*

Definition 5. *(Distinguisher family) Let $\mathcal{P}$ be a family of polynomials and fix $n \in \mathbb{N}$. Given $\alpha < \beta$ in $(0,1)$ and $\eta < 1/2$, we say the family is (α, β, η)–distinguisher for $\lambda \in [0,1]$ if,*

- *For every $x \in [0,1]$ that is α–close to λ, $p_{\lambda,n}(x) \in [1-\eta, 1]$, and*

– *For every $x \in [0,1]$ that is β–far from λ, $p_{\lambda,n}(x) \in [0,\eta]$.*

Theorem 2. *Let $\alpha, \beta, \lambda, \eta$ be such that $\alpha \leq \beta$, $\beta = o(1)$, $\eta = o(n^{-1})$ and $\lambda + \beta \leq \frac{1}{2}$. Then there is no (α, β, η)–distinguisher family for λ that is $O(\log n)$–small.*

Proof. Assume there exists such a family $\{p_{\lambda,n}\}_{\lambda\in[0,1],n\in\mathbb{N}}$ with $s(n) = c' \log n$. We first show that without loss of generality p has logarithmic degree. Let $r_{\lambda,n}(x)$ be the residual error of truncating $p_{\lambda,n}(x)$ after $c \log n$ terms, for c that will soon be determined. Also, w.l.o.g., assume $x \in [0,1)$ is bounded away from 1. Then:

$$r_{\lambda,n}(x) \leq \sum_{i=c\log n+1}^{\deg(p_{\lambda,n})} |c_{\lambda,n,i}| \cdot x^i \leq n^{c'} \cdot \frac{x^{c\log n}}{1-x} \leq \frac{1}{1-x} n^{c'-c\log(1/x)}.$$

So, by taking $c = \lceil \frac{c'+2-\log(1-x)}{\log(1/x)} \rceil$ we obtain $r_{\lambda,n}(x) \leq n^{-2}$.

We now show that $O(\log n)$–degree polynomials cannot decay around λ fast enough. Assume to the contrary that there exists such a distinguisher family, so $|p_{\lambda,n}(x)| < n^{-1}$ for $x \in [\lambda+\beta, 1]$. The following lemma states that if a function has a small value on an interval, than it cannot be too large outside it. Namely,

Lemma 6. *[15, Theorem 2.9.11] Let $T_n(x)$ be the Chebyshev polynomial (of the first kind) of degree n. Then, if the polynomial $P_n(x) = \sum_{i=0}^n c_i x^i$ satisfies the inequality $|P_n(x)| \leq L$ on the segment $[a,b]$ then at any point outside the segment we have*

$$|P_n(x)| \leq L \cdot \left| T_n\left(\frac{2x-a-b}{b-a}\right)\right|.$$

For properties of the Chebyshev polynomials see [16, Chapter 1.1]. We mention a few properties that we use. An explicit representation of $T_n(x)$ is given by $T_n(x) = \frac{\left(x-\sqrt{x^2-1}\right)^n + \left(x+\sqrt{x^2-1}\right)^n}{2}$. $|T_n(-x)| = |T_n(x)|$ and T_n is monotonically increasing for $x > 1$. Also,

$$|T_n(1+\delta)| \leq \left(1+\delta+\sqrt{(1+\delta)^2-1}\right)^n \leq \left(1+4\sqrt{\delta}\right)^n \leq e^{4n\sqrt{\delta}} \leq 2^{8n\sqrt{\delta}} \quad (1)$$

for $0 \leq \delta \leq 1$. Then:

$$\begin{aligned}
|p_{\lambda,n}(\lambda)| &\leq n^{-1} \cdot \left| T_{c\cdot\log n}\left(\tfrac{\lambda-\beta-1}{-\lambda-\beta+1}\right)\right| \\
&= n^{-1} \cdot \left| T_{c\cdot\log n}\left(1+\tfrac{2\beta}{1-\lambda-\beta}\right)\right| && \text{By } |T_n(x)| = |T_n(-x)| \\
&\leq n^{-1} \cdot |T_{c\cdot\log n}(1+4\beta)| && \text{By the monotonicity of } T_n(x) \text{ for } x > 1 \text{ and } \lambda+\beta \leq \tfrac{1}{2}
\end{aligned}$$

By Equation (1) $|p_{\lambda,n}(\lambda)| \leq n^{-1} 2^{32c\sqrt{\beta}\log n} \leq n^{-1+32c\sqrt{\beta}}$. As $\beta = o(1)$ for n large enough we have $|p_{\lambda,n}(\lambda)| \leq n^{-1/2}$, contradicting the fact that $|p_{\lambda,n}(\lambda)| \geq 1 - n^{-1}$.

We note that for values very close to 1, polynomials of higher degrees are useful, and indeed better approximations are possible. In particular, one can separate a 1 eigenvalue from $1 - \frac{1}{n}$ by using the polynomial x^{n^2}.

5 A comparison with the quantum phase estimation

The quantum algorithm of [10, 9] relies on the quantum phase estimation algorithm. We do not explain this algorithm here, and we refer the interested reader to [17, Chapter 7] or [18, Chapter 5]. The quantum phase estimation algorithm has access to a Hermitian operator B with eigenvalues $\lambda_1, \ldots, \lambda_n$. Given an accuracy parameter T the quantum phase estimation algorithm returns answer $\frac{\ell}{T}$ with probability

$$\frac{1}{n}\sum_k \frac{1}{T^2}\frac{\sin^2\left(\pi T(\lambda_k - \frac{\ell}{T})\right)}{\sin^2\left(\pi(\lambda_k - \frac{\ell}{T})\right)} = \frac{1}{n}\sum_k f_T\left(2\pi\left(\lambda_k - \frac{\ell}{T}\right)\right)$$

where f_T is the Fejér kernel – a function that plays a central role in Fourier analysis (see [19, Chapter 2]). So, in fact, although not stated that way, the quantum algorithm uses the same approach of computing Tr $(p(B))$ with $p(x)$ being the (shifted, scaled) Fejér kernel.

One big advantage of the quantum algorithm is that it can simulate the Fejér kernel with polynomially-small accuracy up to any *polynomial* degree, whereas the classical technique we employ only works for $T = O(\log n)$. Another, perhaps inherent, difference is that the quantum algorithm works over arbitrary Hermitian operators, whereas our classical algorithm requires stochastic operators, or at least operators with infinity norm at most 1.

Hence, several natural questions arise. E.g.,

Question 2.

- Can we approximate (even with constant accuracy) the eigenvalues of bounded norm Hermitian operators in BPL? Currently we can handle only operators with infinity norm at most 1.
- Can we approximate the eigenvalues of a stochastic operator with subconstant accuracy in BPL?

Question 3.

- Given an Hermitian matrix A of dimension n with eigenvalues that are well-separated and in $[0, 1]$, and an integer $T = \text{poly}(n)$, can the entries of e^{iTA} be approximated in $O(\log n)$ space, by a probabilistic algorithm, with high probability? More generally,
- Is the family of unitary matrices simulatable?

Solving Question 3 would show that $f_T(A - \lambda I)$ can be approximated for every $\lambda \in [0, 1]$ and T that is $\text{poly}(n)$ and would essentially show that the results that the quantum algorithm achieves can also be obtained in BPL.

References

1. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica **12** (1992) 449–461

2. Saks, M.E., Zhou, S.: $BP_HSPACE(S) \subseteq DSPACE(S^{3/2})$. J. Comput. Syst. Sci. **58** (1999) 376–403
3. Reingold, O.: Undirected connectivity in log-space. J. ACM **55** (2008)
4. Aleliunas, R., Karp, R.M., Lipton, R., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Foundations of Computer Science, 1979., 20th Annual Symposium on. (1979) 218–223
5. Cook, S.A.: A taxonomy of problems with fast parallel algorithms. Information and Control **64** (1985) International Conference on Foundations of Computation Theory.
6. Csansky, L.: Fast parallel matrix inversion algorithms. SIAM Journal of Computing **5** (1976) 618–623
7. Watrous, J.: Space-bounded quantum complexity. Journal of Computer and System Sciences **59** (1999) 281 – 326
8. van Melkebeek, D., Watson, T.: Time-space efficient simulations of quantum computations. Electronic Colloquium on Computational Complexity (ECCC) **17** (2010) 147
9. Ta-Shma, A.: Inverting well conditioned matrices in quantum logspace. In: Proceedings of the 45th annual ACM symposium on Symposium on theory of computing. STOC '13, New York, NY, USA, ACM (2013) 881–890
10. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103** (2009) 150502
11. Ben-Or, M., Eldar, L.: Optimal algorithms for linear algebra by quantum inspiration. CoRR **abs/1312.3717** (2013)
12. Saff, E.B., Totik, V.: Polynomial approximation of piecewise analytic functions. Journal of the London Mathematical Society **s2-39** (1989) 487–498
13. Eremenko, A., Yuditskii, P.: Uniform approximation of sgn(x) by polynomials and entire functions. Journal d'Analyse Mathématique **101** (2007) 313–324
14. Diakonikolas, I., Gopalan, P., Jaiswal, R., Servedio, R.A., Viola, E.: Bounded independence fools halfspaces. SIAM Journal on Computing **39** (2010) 3441–3462
15. Timan, A.: Theory of Approximation of Functions of a Real Variable. Dover books on advanced mathematics. Pergamon Press (1963)
16. Rivlin, T.: The Chebyshev polynomials. Pure and applied mathematics. Wiley (1974)
17. Kaye, P., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Oxford University Press (2007)
18. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Series on Information and the Natural Sciences. Cambridge University Press (2000)
19. Hoffman, K.: Banach Spaces of Analytic Functions. Dover Books on Mathematics Series. Dover Publications, Incorporated (2007)