Week 4

The Polynomial Method

4.1 Representing functions by polynomials

This week, we will study a technique for proving quantum lower bounds in query complexity: that is, showing that there are no fast quantum query algorithms for computing some Boolean function. This technique is known as the *polynomial method*, and relies on the observation that fast quantum query algorithms give rise to certain low-degree polynomials.

The study of polynomial approximation for Boolean functions has by now grown into its own sub-field. It has applications to other parts of complexity theory (including things like AC_0 lower bounds and PAC-learning). It is also closely related to a measure in communication complexity known as the "approximate gamma-2 norm" or "approximate logrank", which we will define later in the course.

We begin our study of polynomials with the following observation, which says that any real-valued function f acting on the Boolean hypercube $\{0,1\}^n$ can be written as an n-variate polynomial in the bits of the input string x. At first, it may seem weird to treat the bits of the string x as real numbers that we can add and multiply – aren't the symbols of the string x arbitrary characters? However, this turns out to be a natural thing to do, as we'll soon see.

Lemma 4.1. *Let $f : \{0,1\}^n \to \mathbb{R}$ be any real-valued function defined on the Boolean hypercube. Then f can be uniquely represented as a multivariate polynomial $p(x_1, x_2, \ldots, x_n)$ with real coefficients, such that $p(x) = f(x)$ for all $x \in \{0,1\}^n$.*

Moreover, the polynomial p will be multilinear, which means that each variable x_i only occurs with degree 0 or 1 in each monomial; i.e.

$$p(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i,$$

with $c_S \in \mathbb{R}$ being the coefficients.

Proof. First, note that for $x_i \in \{0,1\}$, we have $x_i^2 = x_i$, so terms like x_i^2 or x_i^3 don't help – they are equivalent to x_i. With this in mind, there are 2^n different subsets of the set $[n]$, and each subset $S \subseteq [n]$ defines a corresponding monomial monomial $m_S = \prod_{i \in S} x_i$. Any polynomial p over $\{0,1\}^n$ is therefore a linear combination of these 2^n monomials m_S for different subsets S.

Consider the monomials m_S as functions; that is, for each S let $m_S(x)$ denote the product $\prod_{i \in S} x_i$, so that $m_S : \{0,1\}^n \to \{0,1\}$ is a Boolean function. Further, associate with this function m_S a long vector v_S representing the truth table of m_S with one entry for each $x \in \{0,1\}^n$, i.e. the vector with $v_S[x] = m_S(x)$. We claim that the 2^n vectors v_S for $S \subseteq [n]$ are all linearly independent. To see this,

consider a linear combination of them that equals the all-zero vector. This linear combination is some polynomial q in the variables $x_1, x_2, \ldots, x_n$ such that $q(x) = 0$ for all $x \in \{0,1\}^n$. Let m_S be a monomial of q that has non-zero coefficient and such that $|S|$ is as small as possible. Consider the string x with $x_i = 1$ for $i \in S$ and $x_i = 0$ otherwise, and consider the value of $q(x)$. Each monomial of q other than m_S must use some variable outside of S, by the minimality of the choice of S; hence each monomial of q other than m_S evaluates to 0 on x. However, $m_S(x) = 1$, and since m_S has a non-zero coefficient in q, we have $q(x) \neq 0$. This is a contradiction, which means that the monomials are linearly independent, as desired.

Finally, consider the arbitrary function $f\colon \{0,1\}^n \to \mathbb{R}$. The truth table of this function is a vector in $\mathbb{R}^{2^n}$ with one entry for each $x \in \{0,1\}^n$. Since $\mathbb{R}^{2^n}$ has dimension 2^n, and since the vectors v_S are 2^n independent vectors, they are a basis. This means the truth table of f can be uniquely written as a linear combination of the vectors v_S, which means f can be uniquely represented as a real polynomial p. □

Polynomials of this form, in which each variable occurs with degree 0 or 1, are called *multilinear*. We now define the degree of a Boolean function, as follows.

Definition 4.2. *For a function $f\colon \{0,1\}^n \to \mathbb{R}$, define its degree $\deg(f)$ to be the degree of its corresponding real polynomial. That is, the maximum value of $|S|$ for monomials m_S that have a non-zero coefficient in the unique real polynomial p computing f.*

Note that the degree of a function on n bits is always at most n. Also note that although we allowed f to output real numbers, this definition also works perfectly well for Boolean functions that have outputs in $\{0,1\}$. The input and output alphabets should be Boolean for the degree to make sense, at least as we've currently defined it.

We can also extend this definition to partial functions. There are actually two ways of doing so.

Definition 4.3. *Let f be a possibly partial Boolean function on n bits. Then $\deg(f)$ is the minimum degree of a real polynomial p satisfying $p(x) = f(x)$ for $x \in \mathrm{Dom}(f)$.*

An alternative definition is to take the minimum degree of p only over polynomials p satisfying both $p(x) = f(x)$ for all $x \in \mathrm{Dom}(f)$, and also $p(x) \in [0,1]$ for all $x \in \{0,1\}^n$.

So far we have taken polynomial degree where the inputs and outputs are $\{0,1\}$-valued. It is often useful to consider polynomials that take inputs in $\{+1,-1\}^n$ and give outputs in $\{+1,-1\}$ instead. As it turns out, the degree of a function when represented over $\{+1,-1\}$ is the same as when it is represented over $\{0,1\}$. We will associate $+1$ with 0 and -1 with 1, so that we can convert from $\{0,1\}$ to $\{+1,-1\}$ by taking $(-1)^b$ or $1-2b$, and by taking $(1-b)/2$ to convert back from $\{+1,-1\}$ to $\{0,1\}$.

We refer these as different *bases*: the $\{0,1\}$ basis and the $\{+1,-1\}$ basis. The word "basis" comes from the fact that, as we've seen, the monomials m_S form a basis when considered as vectors over $\mathbb{R}^{2^n}$ (with these vectors representing the truth table of the function m_S, i.e. the vectors $v_S[x] = m_S(x)$). It turns out that monomials m_S over $\{+1,-1\}$ variables also form a basis for $\mathbb{R}^{2^n}$. Note that these are different functions now: the function $m_S(x) = \prod_{i \in S} x_i$ is an AND function when x is a $\{0,1\}^n$ vector, but it is a PARITY function when x is a $\{+1,-1\}^n$ vector.

Lemma 4.4. *A polynomial p in the $\{0,1\}$ basis can be converted into a polynomial q in the $\{+1,-1\}$ basis, and vice versa. These conversions preserve the behavior of the polynomials as functions, and also the degree of the polynomials.*

Proof. If p expects $\{0,1\}$ inputs, we can make it take $\{+1,-1\}$ inputs instead by plugging in $(1-x_i)/2$ into each variable x_i of p. Since $(1-x_i)/2$ converts $\{+1,-1\}$ into $\{0,1\}$, this preserves

the behavior of p. When we expand and simplify, it's not hard to see that the degree of p cannot increase, since we are plugging in degree-1 terms into p. We can also change the output of p from $\{0,1\}$ to $\{+1,-1\}$ by applying $1-2p$ to the polynomial, which does not affect its degree.

To convert back, we similarly plug in $(1-2x_i)$ into x_i in q, and apply $(1-q)/2$ on the outside. Once again, this transformation can only decrease the degree. However, since applying both transformations gets us back to exactly the same function (and hence exactly the same polynomial, since each function has a unique polynomial), we conclude that neither transformation can actually decrease the degree, and both must preserve it. □

Note that polynomials taking in $\{+1,-1\}$ variables can still be assumed to be multilinear, since now $x_i^2 = 1$, $x_i^3 = x_i$, etc., and there is still no need for x_i to have degrees other than 0 or 1.

4.2 Approximating polynomials

Having defined the degree of a Boolean function, we now define a related notion called the approximate degree.

Definition 4.5 (Approximating polynomial)**.** *We say that a polynomial p approximates a (possibly partial) Boolean function f to error ϵ if $|p(x) - f(x)| \leq \epsilon$ for all $x \in \mathrm{Dom}(f)$. This is for $\{0,1\}$ outputs; for $\{+1,-1\}$ outputs, the values get stretched by a factor of 2, so we require instead that $|p(x) - f(x)| \leq 2\epsilon$.*

Definition 4.6 (Bounded polynomial)**.** *We say that a real polynomial p in n variables is bounded if $p(x) \in [0,1]$ for all $x \in \{0,1\}^n$. This is when dealing with $\{0,1\}$ outputs; in the context of $\{+1,-1\}$ outputs, we will say p is bounded if $|p(x)| \leq 1$ for all x in the Boolean hypercube.*

With these definitions in hand, we can define the approximate degree of a Boolean function.

Definition 4.7. *Let f be a (possibly partial) Boolean function on n bits. The approximate degree of f to error ϵ, denoted $\widetilde{\deg}_\epsilon(f)$, is the minimum degree of a bounded polynomial p which approximates f to error ϵ. When $\epsilon = 1/3$, we omit it and write $\widetilde{\deg}(f)$.*

It turns out that approximate polynomial degree lower bounds quantum query complexity. Before we see this, let's see why it lower bounds deterministic and randomized query complexities.

Theorem 4.8. *Let f be a (possibly partial) Boolean function. Then $\mathrm{D}(f) \geq \deg(f)$.*

Proof. We will show that each decision tree D has a corresponding polynomial p_D which computes the same function as D and that has degree at most the height of D. We proceed by induction over the height of D. If the height of D is 0, meaning D is a constant, we can represent it by a constant polynomial (either the constant 0 or the constant 1). These polynomials have degree 0, completing the base case.

Suppose all decision trees of height at most k can be represented by polynomials in this fashion, and let D be a decision tree of height $k+1$. Then D starts by querying some position x_i of the input x, and if $x_i = 0$ it goes to the root of subtree D_0, while if $x_i = 1$ it goes to the root of subtree D_1. These subtrees have height at most k, so they are represented by polynomials p_{D_0} and p_{D_1} of degree at most k. We claim that $p = (1-x_i)p_{D_0} + x_i p_{D_1}$ is a polynomial computing the same function as D. To see this, note that if $x_i = 0$, p outputs the same as p_{D_0}, which is the same as the output of D_0; on the other hand, if $x_1 = 1$, p outputs the same as p_{D_1}, which is the same as the output of D_1. In both cases, $p(x) = D(x)$. Moreover, the degree of p is at most $k+1$, since p_{D_0} and p_{D_1} have degree at most k. This completes the induction argument.

Finally, recall that $\mathrm{D}(f)$ is the minimum height of a decision tree computing f. This minimum decision tree converts into a polynomial computing f, and this polynomial has degree at most $\mathrm{D}(f)$, so $\mathrm{D}(f) \geq \deg(f)$. □

While the exact degree of a Boolean function lower bounds $\mathrm{D}(f)$, to lower bound $\mathrm{R}(f)$ we will need the approximate degree.

Theorem 4.9. *Let f be a (possibly partial) Boolean function. Then $\mathrm{R}_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f)$.*

Proof. Recall that a randomized query algorithm is a probability distribution over decision trees. Let R be the one that minimizes $\mathrm{R}_\epsilon(f)$, so that each decision tree in the support of R has height at most $\mathrm{R}_\epsilon(f)$. Let $p = \sum_D \Pr[D] \cdot p_D$, where the sum ranges over decision trees D in the support of R, $\Pr[D]$ denotes the probability of D in R, and p_D is the polynomial computing the same function as the decision tree D. Then p_D has degree at most $\mathrm{R}_\epsilon(f)$ for all D, and p is a linear combination of such polynomials, so it has degree at most $\mathrm{R}_\epsilon(f)$.

Moreover, for each $x \in \mathrm{Dom}(f)$, we have $\Pr[R(x) \neq f(x)] \leq \epsilon$, which means $\Pr_{D\sim R}[p_D(x) \neq f(x)] \leq \epsilon$. It is not hard to see that $\Pr_{D\sim R}[p_D(x) \neq f(x)] = |p(x) - f(x)|$ when $f(x)$ is $\{0,1\}$-valued, so we have $|p(x) - f(x)| \leq \epsilon$ for all $x \in \mathrm{Dom}(f)$. Finally, it is clear that $p(x) \in [0,1]$ for all $x \in \{0,1\}^n$, because each $p_D(x)$ is in $\{0,1\}$ and $p(x)$ is a convex combination of $p_D(x)$ terms. □

As it turns out, the approximate degree also lower bounds quantum query complexity, not just randomized query complexity.

Theorem 4.10. *Let f be a (possibly partial) Boolean function. Then $\mathrm{Q}_\epsilon(f) \geq \widetilde{\deg}_\epsilon(f)/2$.*

Proof. Let Q be a T-query quantum algorithm computing f to error ϵ, where $T = \mathrm{Q}_\epsilon(f)$. Then Q is a sequence of unitaries $U_0, U_1, \ldots, U_T$, and the output of $Q(x)$ is the result of querying the output register of

$$U_T U^x U_{T-1} U^x \ldots U^x U_1 U^x U_0 \left|\psi\right\rangle$$

for some initial state $|\psi\rangle$. We assume the queries U^x are made in phase. We will also let the string x be represented using the $\{+1,-1\}$ alphabet; this way, U^x maps $|i\rangle|b\rangle$ to $x_i^b |i\rangle|b\rangle$ for $b \in \{0,1\}$.

Let $|\psi_t^x\rangle$ be the state of the algorithm right after the unitary U_t, so $|\psi_t^x\rangle = U_t U^x U_{t-1} U^x \ldots U^x U_0 |\psi\rangle$. Consider the amplitudes of $|\psi_t^x\rangle$, that is, the entries of the vector. When $t = 0$, no queries have been made yet, and so these entries have no dependence on x; they are constants relative to x. When a query gets made, some of the amplitudes (those corresponding to basis elements with a 1 in the B register) get multiplied by one of the x_i variables; in other words, every time U^x gets applied, the polynomial degree of each amplitude in the vector is increased by 1. Also, note that the U_t unitaries have no dependence on x; after applying U_t, the amplitudes of the vector get linearly transformed (so each amplitude in the resulting vector is a linear combination of the amplitudes in the input vector), which does not change their degree.

We conclude that each amplitude in $|\psi_t^x\rangle$ is a polynomial in x of degree at most t. Hence the final state $|\psi_T^x\rangle$ has amplitudes that are polynomials in x, each of degree at most T. Note that these polynomials might have complex coefficients; we will write $|\psi_T^x\rangle = |\phi_1\rangle + i|\phi_2\rangle$, where $|\phi_1\rangle$ and $|\phi_2\rangle$ have real coefficients. Then the entries of $|\phi_1\rangle$ and $|\phi_2\rangle$ are real polynomials of degree at most T.

The acceptance probability of the algorithm Q when run on x is the probability that we see a 1 when we measure some register in the final state $|\psi_T^x\rangle$; this can be expressed as a sum of squares of entries from $|\phi_1\rangle$ and $|\phi_2\rangle$. Squaring the entries increases their degree by at most a factor of 2, while adding up polynomials does not increase their degree; hence we conclude that the final acceptance probability is a real polynomial in x of degree at most $2T$. Denote this polynomial by $p(x)$.

We conclude that $\deg(p) \leq 2T$ and that $p(x) = \Pr[Q(x) = 1]$ for all $x \in \{+1, -1\}^n$. Since p computes an acceptance probability, we have $p(x) \in [0, 1]$ for all $x \in \{+1, -1\}^n$, even when x is not in $\mathrm{Dom}(f)$ (since in all cases the quantum algorithm does *something* and has *some* probability of accepting). Moreover, since Q computes f to error ϵ, we have $|p(x) - f(x)| \leq \epsilon$, which means that p approximates f to error ϵ. This shows that $\widetilde{\deg}_\epsilon(f) \leq 2T$, as desired. □

4.3 Symmetrization

Now that we've seen that $\widetilde{\deg}(f)$ lower bounds $\mathrm{Q}(f)$, it remains to get a handle on determining the approximate degree of Boolean functions. This is often tricky to do, but it becomes much easier if the function is symmetric, due to a trick called symmetrization.

Theorem 4.11. *Let p be an n-variate multilinear polynomial. Then there is a single-variate polynomial q such that for all $t \in \{0, 1, 2, \ldots, n\}$, $q(t)$ evaluates to the average of $p(x)$ over all strings $x \in \{0, 1\}^n$ of Hamming weight t. Further, the degree of q is at most the degree of p.*

Proof. The trick is to "symmetrize" the polynomial p by averaging over all permutations of the variables. That is, for each permutation $\sigma \in S_n$, let p_σ be the polynomial $p_\sigma(x) = p(x_\sigma)$, where x_σ is the string $(x_\sigma)_i = x_{\sigma(i)}$. That is, p_σ is the polynomial we get if we permute the input string x according to σ before we applying p to the result. Then let

$$p_{sym}(x) = \frac{1}{n!} \sum_{\sigma \in S_n} p_\sigma(x).$$

Note that each p_σ has the same degree as p, so the degree of p_{sym} is at most the degree of p (it might be lower due to cancellations). Moreover, the degree p_{sym} is symmetric in its variables; that is, $p_{sym}(x_\sigma) = p_{sym}(x)$ for all σ.

It turns out that symmetric multilinear polynomials can always be written as a linear combination of the elementary symmetric polynomials. The first elementary symmetric polynomial is $J_0 = 1$, the second is $J_1 = \sum_i x_i$, the third is $J_2 = \sum_{i<j} x_i x_j$, and so on. Note that the degree of J_t is t. Since p_{sym} is symmetric, we can write

$$p_{sym} = \sum_{t=0}^{\deg(p)} c_t J_t$$

for some real coefficients c_t. Next, note that $J_1^2 = J_2 + \sum_i x_i^2 = J_2 + J_1$, since $x_i^2 = x_i$. Hence $J_2 = J_1^2 - J_1$. We similarly have $J_3 = J_2 J_1 - 2J_2 = J_1^3 - 3J_1^2 + 2J_1$. More generally, each J_t can be written as a degree-t polynomial in J_1. Hence p_{sym} can be written as $q(J_1)$ with q being the polynomial $q(t) = \sum_{j=0}^{\deg(p)} d_j t^j$ for some real coefficients d_j. We conclude that $q(t)$ computes the average of $p(x)$ over strings of Hamming weight t, as desired. □

This theorem is primarily useful when analyzing the approximate degree of symmetric functions. For example, let's use it to lower bound the approximate degree of PARITY.

Lemma 4.12. $\widetilde{\deg}_\epsilon(\textsc{Parity}) = n$ *for all* $\epsilon < 1/2$.

Proof. Suppose p approximates PARITY, and consider its outputs in $\{+1, -1\}$ form. Let q be the single-variate symmetrization of p, so that $q(t)$ is the average of $p(x)$ over $x \in \{0, 1\}^n$ with Hamming weight t. Note that for x with Hamming weight t, $p(x)$ must approximate $(-1)^t$, so $p(x)(-1)^t \in [1 - 2\epsilon, 1]$. Hence the average of $p(x)$ over x of Hamming weight t still has this property,

so $q(t)(-1)^t \geq [1-2\epsilon]$ for all $t = 0, 1, \ldots n$. In particular, $q(t)$ must be positive when t is even and negative when t is odd. This means that in the range $[0, n]$, $q(t)$ must cross the x-axis at least n times. It follows that $q(t)$ must have degree at least n, and since its degree is at most that of p, p must also have degree at least n. Hence $\widetilde{\deg}_\epsilon(\textsc{Parity}) = n$. □

Note that this lemma implies that $Q_\epsilon(\textsc{Parity}) \geq n/2$ for all $\epsilon < 1/2$; that is, a quantum algorithm cannot even achieve a small bias towards computing the parity of a string in fewer than $n/2$ queries (note that $n/2$ can be achieved by an exact quantum algorithm when n is even, using Deutsch–Jozsa; hence this lower bound is exactly tight).

Next, we consider the approximate degree of PromiseOR. Recall that PromiseOR is the function which outputs 0 when the Hamming weight of the string is 0, outputs 1 when the Hamming weight of the string is 1, and is undefined elsewhere. A polynomial p which approximates PromiseOR, and which is bounded, turns into a single-variate polynomial $q(t)$ with $q(0) \leq \epsilon$, $q(1) \geq 1-\epsilon$, and $q(t) \in [0,1]$ for all $t = 0, 1, 2, \ldots, n$. This polynomial q moves quickly between $q(0)$ and $q(1)$ (going up by $1-2\epsilon$ in the y-axis for a shift of 1 in x-axis, meaning its derivative was at least $1-2\epsilon$ at some point). However, this polynomial is also stuck inside $[0,1]$ on all the points $\{0, 1, 2, \ldots, n\}$.

The following property of single-variate real polynomials will come in handy.

Theorem 4.13 (Markov Brothers' inequality)**.** *Let $p\colon \mathbb{R} \to \mathbb{R}$ be a real polynomial of degree d. If $p(x) \in [b, b']$ for all $x \in [a, a']$, then $|p'(x)| \leq \frac{b'-b}{a'-a} d^2$ for all $x \in [a, a']$.*

This theorem says that if a polynomial is stuck in a rectangular box, which is $a'-a$ wide and $b'-b$ high, then it cannot have a high derivative unless it has high degree. In particular, if the width of the box is n and the height is 1, then the derivative of the polynomial in the box must be at most d^2/n, which means that if the polynomial moves sharply—getting derivative $\Omega(1)$—then it must have degree at least $\Omega(\sqrt{n})$.

Markov brothers' inequality was proven in the 1800s, though its proof is too technical to include here. Instead, we will see how it can be used to imply a lower bound on the approximate degree of PromiseOR. First, we will need the following corrolary, first shown by [EZ64; RC66]. This corollary gives a lower bound on the degree of a polynomial even if it is only known to stay in the box on integer points, rather than on all points.

Corollary 4.14. *Let $p\colon \mathbb{R} \to \mathbb{R}$ be a real polynomial. Let $n \in \mathbb{N}$ be a positive integer, and suppose $p(x) \in [0,1]$ for all $x \in \{0, 1, 2, \ldots n\}$. Then $\deg(p) \geq \sqrt{nc/(1+c)}$, where c is the maximum value of $|p'(x)|$ for $x \in [0, n]$.*

Proof. Let c be the maximum value of $|p'(x)|$ for $x \in [0, n]$. Note that if $p(x)$ is ever outside of $[0,1]$ by more than δ for some $x \in [0, n]$, then x has distance at most $1/2$ to some $i \in \{0, 1, 2, \ldots, n\}$, which means that p changed by at least δ from $p(i)$ to $p(x)$ (with an x-axis change of at most $1/2$). By the mean value theorem, we have $|p'(y)| \geq 2\delta$ for some y between x and i, so $y \in [0, n]$. This means that $c \geq 2\delta$, or $\delta \leq c/2$. In other words, we must have $p(x) \in [-c/2, 1+c/2]$ for all $x \in [0, n]$.

We now have bounds on $p(x)$, so we use Markov brothers' inequality. This tells us that $|p'(x)| \leq (1+c)\deg(p)^2/n$ for $x \in [0, n]$, so $c \leq (1+c)\deg(p)^2/n$. Rearranging, we get $\deg(p) \geq \sqrt{nc/(1+c)}$, as desired. □

With this tool in hand, we can prove a lower bound on $\widetilde{\deg}(\textsc{PromiseOR})$.

Theorem 4.15. *For all $\epsilon \in [0, 1/2)$ and all $n \in \mathbb{N}$, we have*

$$\widetilde{\deg}_\epsilon(\textsc{PromiseOR}_n) \geq \sqrt{\frac{1-2\epsilon}{2(1-\epsilon)} n}.$$

In particular, $\widetilde{\deg}(\textsc{PromiseOR}_n) \geq \sqrt{n}/2$.

Proof. Let p be a bounded polynomial approximating $\textsc{PromiseOR}$ to error ϵ, and let q be the single-variate symmetrization polynomial with degree at most that of p. Then $q(t) \in [0,1]$ for $t \in \{0,1,2,\ldots,n\}$. Let d be the degree of q, so $d \leq \widetilde{\deg}_\epsilon(\textsc{PromiseOR})$. By the above corollary, we have $d \geq \sqrt{nc/(1+c)}$, where c is the maximum value of $q'(t)$ for $t \in [0,n]$. Note that $q(0) \leq \epsilon$ and $q(1) \geq 1-\epsilon$, so by the mean value theorem, $q'(t) \geq 1-2\epsilon$ for some $t \in [0,1]$, from which the desired result follows. □

Recall from the first week that if a measure M satisfies some simple properties, then for all f we have $M(f) \geq M(\textsc{PromiseOR}_{\mathrm{bs}(f)})$. Because of this, we get the following corollary, first observed by Beals, Buhrman, Cleve, Mosca, and de Wolf [BBC+01].

Corollary 4.16. *Let f be a (possibly partial) Boolean function. Then $\widetilde{\deg}(f) \geq \sqrt{\mathrm{bs}(f)}/2$.*

We also have a corollary for quantum algorithms, which has a better dependence on ϵ when ϵ is close to $1/2$.

Corollary 4.17. *Let f be a (possibly partial) Boolean function. Then*

$$\mathrm{Q}_\epsilon(f) \geq \sqrt{\frac{1-2\epsilon}{8(1-\epsilon)}\,\mathrm{bs}(f)} \geq \sqrt{(1-2\epsilon)\,\mathrm{bs}(f)/8}.$$

Next, let's lower bound the approximate degree of majority. We have $\mathrm{bs}(\mathrm{MAJ}_n) = (n+1)/2$ (where we assume n is odd), so we know that $\widetilde{\deg}(\mathrm{MAJ}_n) = \Omega(\sqrt{n})$. Can we improve this?

If we start with a polynomial p approximating MAJ_n and symmetrize it, we get a polynomial q in one variable with $q(t) \in [0,\epsilon]$ for $t \in \{0,1,\ldots,(n-1)/2\}$ and $q(t) \in [1-\epsilon,1]$ for $t \in \{(n+1)/2,\ldots,n\}$. Using Corollary 4.14 on this polynomial will only give an $\Omega(\sqrt{n})$ lower bound on the degree of q. To improve this, we will use yet another old theorem from approximation theory.

Theorem 4.18 (Bernstein's inequality)**.** *Let $p\colon \mathbb{R} \to \mathbb{R}$ be a real polynomial of degree d. If $p(x) \in [b,b']$ for all $x \in [a,a']$, then for all $x \in [a,a']$, we have*

$$|p'(x)| \leq \frac{b'-b}{2\sqrt{(x-a)(a'-x)}}d.$$

As before, we need a version of this theorem in which the polynomial is only known to be bounded on integer points. This was shown by Paturi [Pat92] (the proof is quite tricky).

Theorem 4.19 (Paturi)**.** *Let $p\colon \mathbb{R} \to \mathbb{R}$ be a real polynomial. Let $n \in \mathbb{N}$ be a positive integer, and suppose $p(x) \in [0,1]$ for all $x \in \{0,1,2,\ldots,n\}$. Then for all $x \in [0,n]$, we have*

$$\deg(p) \geq \frac{|p'(x)|}{C(1+|p'(x)|)}\sqrt{x(n-x)}$$

where C is some universal constant.

Note that Paturi's theorem improves on Corollary 4.14 when x is near $n/2$; if $|p'(x)|$ is at least a constant for such x, then Paturi gives $\deg(p) = \Omega(n)$ instead of $\Omega(\sqrt{n})$. In other words, we knew from before that if a polynomial stays bounded in a box of width n and height 1, and if at some point it has constant derivative, then it must have degree at least $\Omega(\sqrt{n})$; we now know that unless the constant derivative only happens near the left/right edges of the box, the polynomial must in fact have degree at least $\Omega(n)$. Further, this holds even when the polynomial only stays in the box on integer points.

Using Paturi's theorem, we can prove an approximate degree lower bound for majority.

Theorem 4.20. $\widetilde{\deg}(\mathrm{MAJ}_n) = \Omega(n)$.

Proof. Let p be a polynomial approximating MAJ_n to error ϵ, and let q be the symmetrization of p. Then $q(t) \in [0,1]$ for all $t \in \{0,1,2,\ldots,n\}$. Now, we know that $q((n-1)/2) \leq \epsilon$ and $q((n+1)/2) \geq 1-\epsilon$, so by the mean value theorem, some point t in between them has $q'(t) \geq 1-2\epsilon$. Paturi's theorem then gives $\deg(q) = \Omega((1-2\epsilon)n)$, which is $\Omega(n)$ when $\epsilon = 1/3$. □

4.4 Larger alphabets

The polynomial degree measures we've defined so far are specific to functions that have Boolean inputs and outputs. We can generalize this to a notion of polynomials that can handle larger input and output alphabets.

To handle larger output alphabets, we will use a collection of polynomials $\{p_c\}$ for each symbol c in the output alphabet. The polynomial $p_c(x)$ will represent the probability that a quantum algorithm outputs the symbol c when run on x. Such a collection will be required to have $\sum_c p_c(x) = 1$ for all $x \in \{0,1\}^n$. It will also be required to have $p_c(x) \geq 1-\epsilon$ if $f(x) = c$. Its degree will be the maximum degree of p_c for any c.

More interesting is the question of how to handle larger input alphabets. It turns out that the natural way to do so is as follows. Instead of defining a polynomial in n variables $x_1, x_2, \ldots, x_n$ where $x \in \{0,1\}^n$ is the input string, we will instead define a polynomial in nm variables, where m is the size of the input alphabet. The variable y_{ij} for $i \in [n]$ and $j \in [m]$ will be in indicator variable for whether $x_i = j$, where $x \in [m]^n$ is the input. That is, $p(y)$ will be a polynomial in nm variables, and applying p to x works by evaluating $p(y_x)$, where y_x is the string with $(y_x)_{ij} = 1$ if $x_i = j$ and $(y_x)_{ij} = 0$ otherwise. As before, we require that $|p(y_x) - f(x)| \leq \epsilon$ for all $x \in \mathrm{Dom}(f)$ (assuming f has Boolean outputs; see above if not). For boundedness, we will require that $p(y_x) \in [0,1]$ for all $x \in [m]^n$, but not necessarily for all $y \in \{0,1\}^{nm}$.

Actually, when dealing with large input alphabets, it is often convenient to consider an extra symbol $\perp$ which is promised never to occur. When $\perp$ does occur in an input, a quantum algorithm accepting that input may output anything. We will replace the above boundedness requirement with the stronger requirement that $p(y_x) \in [0,1]$ for all $x \in ([m] \cup \{\perp\})^n$, where $(y_x)_{ij} = 0$ for all j when $x_i = \perp$.

The above generalizations allow us to write $\widetilde{\deg}(f)$ for functions f with non-Boolean inputs and/or outputs. We will primarily care about the case of non-Boolean inputs, so we will often assume f has Boolean outputs for simplicity. It turns out that $\mathrm{Q}(f) \geq \widetilde{\deg}(f)/2$ still holds for these generalized notions of approximate degree, so we can still use polynomials to lower bound quantum algorithms in this setting.

One thing that will come in handy is a notion of symmetrization for polynomials with non-Boolean input alphabets.

Theorem 4.21. *Let p be a polynomial in the variables y_{ij} for $i \in [n]$, $j \in [m]$. Then there is a polynomial $q(z)$ in m variables such that $\deg(q) \leq \deg(p)$ and for all $z \in [n]^m$ with $\sum_{j=1}^m z_j \leq n$, the value of $q(z)$ equals the average of $p(y_x)$ over all strings $x \in ([m] \cup \{\perp\})^n$ such that z_j is the number of times the symbol j occurs in x.*

This theorem statement is a little confusing. Think of it as follows: instead of having a single new variable t corresponding to the Hamming weight, and have a new polynomial $q(t)$ act only in t, we now have m different "Hamming weights", each for a different alphabet symbol. These Hamming weights are $z_1, z_2, \ldots, z_m$, and must sum to at most n, since there are n total symbols in a string of length n (they need not sum to exactly n due to the possibility of the symbol $\perp$ showing up). The

new polynomial q will act in the m variables $z_1, z_2, \ldots, z_m$, and will evaluate to the average of $p(y_x)$ over all strings x that are of type z (i.e. all strings x for which z correctly specifies the number of times each alphabet symbol in $[m]$ occurs). It turns out that q is still a low-degree polynomial in the variables z_j (with degree at most that of p), though it does not need to be multilinear.

4.5 Duality for polynomials

We have defined $\widetilde{\deg}_\epsilon(f)$ as the minimum degree of a polynomial p which approximates f to error ϵ. Consider the flip side of this question: the task of minimizing the error ϵ to which a polynomial p of fixed degree d can approximate f. This flipped question is closely related to the original one; if we knew, for each d, the smallest error ϵ_d to which polynomials of degree at most d may approximate f, we could simply define $\widetilde{\deg}_\epsilon(f) = \min\{d \in \{0, 1, \ldots, n\} : \epsilon_d \leq \epsilon\}$.

It turns out that this flipped question can be viewed as a linear program. This is perhaps most cleanly expressed when f (and the polynomial approximating it) have $\{+1, -1\}$ outputs, in which case we want to approximate f to error 2ϵ. Indeed, it is the linear program

$$\begin{array}{lll} \min & \epsilon & \\ \text{s.t.} & f(x)\cdot\sum_{m\in M_d} m(x)c_m \geq 1-2\epsilon & \forall x \in \mathrm{Dom}(f) \\ & \sum_{m\in M_d} m(x)c_m \leq 1 & \forall x \in \{+1,-1\}^n \\ & \sum_{m\in M_d} m(x)c_m \geq -1 & \forall x \in \{+1,-1\}^n. \end{array}$$

In the above program, the variables are ϵ and c_m for all $m \in M_d$, where M_d is the set of all monomials which have degree at most d. The monomials $m(x)$ should be treated as constants. What is going on above is that we wish to minimize the variable ϵ subject to constraints that effectively say there is a polynomial of degree at most d that approximates f to error ϵ. To see that this is what the constraints say, note that the coefficients c_m define a polynomial $p(x) = \sum_{m\in M_d} c_m m(x)$; the last two constraints say that $p(x) \leq 1$ and $p(x) \geq -1$ for all x, meaning that p is bounded in $[-1, 1]$ on the Boolean hypercube. The first constraint says that $f(x)p(x) \geq 1 - 2\epsilon$; since we have $f(x) \in \{+1, -1\}$, this is equivalent to the condition $|f(x) - p(x)| \leq 2\epsilon$. Hence the optimal solution to this program is the minimum ϵ such that there is a polynomial p of degree at most d which approximates f to error ϵ. The key observation – and the reason we wrote things this way in the first place – is that all the constraints are linear in the variables ϵ and c_m.

One key property of linear programs is that we can take the *dual* of them. This is a way of turning a minimization program into a related maximization program, such that their optimal solutions have the same objective value. In the dual, we have one variable for each constraint; this gives us a variable μ_x for each $x \in \mathrm{Dom}(f)$, as well as variables ν_x^+ and ν_x^- for each $x \in \{+1, -1\}^n$. The dual turns out to be

$$\begin{array}{lll} \max & (1/2)\sum_x \mu_x - \sum_x \nu_x^+ - \sum_x \nu_x^- & \\ \text{s.t.} & \sum_x m(x)(f(x)\mu_x/2 - \nu_x^+ + \nu_x^-) = 0 & \forall m \in M_d \\ & \sum_x \mu_x = 1 & \\ & \mu_x \geq 0 & \forall x \in \mathrm{Dom}(f) \\ & \nu_x^+, \nu_x^- \geq 0 & \forall x \in \{+1,-1\}^n. \end{array}$$

We can simplify this. First, we can substitute $\sum_x \mu_x = 1$ in the objective function, since this is a constraint. Next, observe that if for any $x \in \{+1, -1\}^n$ it holds that $\nu_x^+ > 0$ and $\nu_x^- > 0$, we can decrease both by the same amount, until one of them becomes 0; this only increases the objective value, and does not break any constraints. Hence we can assume that for each x, either $\nu_x^+ = 0$ or

$\nu_x^- = 0$. We can set $\nu_x = \nu_x^+$ if $\nu_x^+ \geq 0$ and $\nu_x = -\nu_x^-$ if $\nu_x^- \geq 0$. This way, we will get

$$\begin{array}{lrll} \max & 1/2 - \|\nu\|_1 & & \\ \text{s.t.} & \sum_x m(x)(f(x)\mu_x/2 - \nu_x) & = 0 & \forall m \in M_d \\ & \sum_x \mu_x & = 1 & \\ & \mu_x & \geq 0 & \forall x \in \mathrm{Dom}(f). \end{array}$$

Note that we can always achieve objective value at least 0 by taking $\nu_x = f(x)\mu_x/2$. Now, note that for any x, if $f(x)\mu_x/2$ and ν_x are both positive in the optimal solution, we can decrease them both by the same amount (say $\delta/2$), preserving the first constraint; we can then scale μ and ν up by the same factor of $1/(1-\delta)$, which restores the second constraint. This new solution has objective value $1/2 - (\|\nu\|_1 - \delta/2)/(1-\delta)$, which is at least $1/2 - \|\nu\|_1$ when the latter is at least 0. Hence we can assume that either $f(x)\mu_x/2$ or ν_x is not positive in the optimal solution. Similarly, if they are both negative, we can add $\delta/2$ weight and the same calculation goes through; hence they are not both negative. Finally, if one is negative and the other is positive, we can add $\delta/2$ to one and subtract $\delta/2$ from the other, and once again scale ν and μ by $1/(1-\delta)$. This lets us conclude that either $\mu_x = 0$ or $\nu_x = 0$ for all $x \in \{+1, -1\}^n$.

Let $v_x = f(x)\mu_x/2 - \nu_x$. Then we now know that $\|v\|_1 = \|\mu/2\|_1 + \|\nu\|_1 = 1/2 + \|\nu\|_1$. We also have $\sum_x v_x f(x) = \sum_x f(x)^2\mu_x/2 - \sum_x f(x)\nu_x = 1/2 - \sum_x f(x)\nu_x \geq 1/2 - \|\nu\|_1$, and for each $m \in M_d$, $\sum_x m(x)v_x = 0$. It is also not hard to get back μ and ν if someone gives us v satisfying these conditions. Letting $\gamma = \|\nu\|_1$, the program can be written

$$\begin{array}{lrll} \max & 1/2 - \gamma & & \\ \text{s.t.} & \sum_x m(x)v_x & = 0 & \forall m \in M_d \\ & \sum_x f(x)v_x & \geq 1/2 - \gamma & \\ & \|v\|_1 & = 1/2 + \gamma. & \end{array}$$

Finally, let $\delta = 1/2 - \gamma$, and let $u = v/(1/2+\gamma)$. Then $\|u\|_1 = 1$, and $\sum_x f(x)u_x \geq \delta/(1-\delta)$. The program is then

$$\begin{array}{lrll} \max & \delta & & \\ \text{s.t.} & \sum_x m(x)u_x & = 0 & \forall m \in M_d \\ & \sum_x f(x)u_x & \geq \delta/(1-\delta) & \\ & \|u\|_1 & = 1. & \end{array}$$

How shall we interpret this? Well, since $\|u\|_1 = 1$, we can split it up into a probability distribution μ with $\mu_x = |u_x|$ and a function $f' \colon \{+1,-1\}^n \to \{+1,-1\}$ specifying the sign, so that $u_x = f'(x)\mu_x$ (note that we reused the variable μ_x, which has a different meaning now than in the first linear program). In terms of μ and f', the constraints are

$$\mathop{\mathbb{E}}_{x\sim\mu}[m(x)f'(x)] = 0 \quad \forall m \in M_d$$

$$\mathop{\mathbb{E}}_{x\sim\mu}[f(x)f'(x)] \geq \delta/(1-\delta).$$

In other words, we have the following conclusion.

Theorem 4.22. *To show that there is no bounded polynomial of degree d which approximates f to error ϵ, it suffices to show that there is a total function f' and a distribution μ over the hypercube such that*

1. *f' has correlation greater than $\epsilon/(1-\epsilon)$ with f when measured against distribution μ (i.e. $\mathbb{E}_{x\sim\mu}[f(x)f'(x)] > \epsilon/(1-\epsilon)$ in the $\{+1,-1\}$ basis, where we set $f(x) = 0$ if $x \notin \mathrm{Dom}(f)$),*

2. *f' has zero correlation with all monomials m of degree at most d when measured against μ (i.e. $\mathbb{E}_{x\sim\mu}[f'(x)m(x)] = 0$ for all such monomials m in the $\{+1,-1\}$ basis).*

Moreover, this technique is tight: if there is no bounded polynomial of degree d approximating f to error ϵ, then such f' and μ must exist.

This theorem is not too easy to use in practice, but it is often employed for proving polynomial lower bounds when symmetrization does not work.

4.6 Composition theorems

Recall that if f and g are (possibly partial) Boolean functions defined on n and m bits respectively, then $f \circ g$ is a (possibly partial) Boolean function on nm bits. We can get a polynomial for $f \circ g$ by composing a polynomial p_f for f with n copies of a polynomial p_g for g. However, there is a problem if p_g only approximates g: in this case, there is no guarantee that $p_f(p_g, p_g, \ldots, p_g)$ approximates $f \circ g$, because p_f is only known to approximates f on the Boolean hypercube (not when fed in noisy input bits). It is possible to overcome this issue by amplifying the polynomial p_g before composing, increasing its degree by a factor of $O(\log \widetilde{\deg}(f))$ and decreasing its approximation error to $1/\operatorname{poly}(\widetilde{\deg}(f))$. This gives a polynomial approximating $f \circ g$ of degree $O(\widetilde{\deg}(f)\,\widetilde{\deg}(g)\log\widetilde{\deg}(f))$.

It turns out that the extra log factor is not necessary when composing polynomials, just as it was not necessary for quantum algorithms. This was shown by Sherstov [She13]. This means that $\widetilde{\deg}(f \circ g) = O(\widetilde{\deg}(f)\,\widetilde{\deg}(g))$. On the other hand, the other direction is still open.

Conjecture 4.23. *For all Boolean functions f and g, $\widetilde{\deg}(f \circ g) = \Omega(\widetilde{\deg}(f)\,\widetilde{\deg}(g))$.*

While the conjecture is still open, some partial results are known. Sherstov [She12] showed the following using the dual characterization of approximate degree.

Theorem 4.24. *Let f act on n bits and let g act on m bits. Then*

$$\widetilde{\deg}(f \circ g) = \Omega\left(\frac{\widetilde{\deg}(f)^2\,\widetilde{\deg}(g)}{n}\right).$$

In particular, this theorem gives a tight composition theorem when $\widetilde{\deg}(f) = \Omega(n)$, but it gets weaker the smaller $\widetilde{\deg}(f)$ gets.

Other partial progress towards a composition theorem includes work on composition with AND and OR.

References

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM* (2001). Previous version in FOCS 1998. DOI: 10.1145/502090.502097. arXiv: quant-ph/9802049 (p. 7).

[EZ64] Hartmut Ehlich and Karl Zeller. Schwankung von Polynomen zwischen Gitterpunkten. *Mathematische Zeitschrift* (1964). DOI: 10.1007/BF01111276 (p. 6).

[Pat92] Ramamohan Paturi. On the degree of polynomials that approximate symmetric Boolean functions (preliminary version). *Proceedings of the 24th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. 1992. DOI: 10.1145/129712.129758 (p. 7).

[RC66] Theodore J. Rivlin and Elliott Ward Cheney. A comparison of uniform approximations on an interval and a finite subset thereof. *SIAM Journal on Numerical Analysis* (1966). DOI: 10.1137/0703024 (p. 6).

[She12] Alexander A. Sherstov. Strong Direct Product Theorems for Quantum Communication and Query Complexity. *SIAM Journal on Computing* (2012). Previous version in STOC 2011. DOI: 10.1137/110842661. arXiv: 1011.4935 (p. 11).

[She13] Alexander A. Sherstov. Making Polynomials Robust to Noise. *Theory of Computing* (2013). Previous version in STOC 2012. DOI: 10.4086/toc.2013.v009a018. ECCC: 2012/037 (p. 11).