CS294-4: Fourier Transforms and Theoretical Computer Science	Fall 1996
Lecture 6: March 5, 1999	
Lecturer: Madhu Sudan	*Scribe: Ashwin Nayak*

This lecture will cover the following topics: Linearity Testing, testing the Long Code and Probabilistically Checkable Proofs.

6.1 Linearity Testing

Let G and H be two finite groups. We say that a function $f : G \to H$ is *linear* if for all $x, y \in G$, we have $f(x+y) = f(x) + f(y)$. (This is in fact the same as saying that f is a *group homomorphism*.)

The problem of interest to us is the following: given an oracle for a function f, can we check "quickly" whether f is linear? Clearly, any check for linearity would have to probe the function at *all* the points. So we relax our requirement a little: can we check quickly whether the function is *approximately* linear? By a function f being "approximately" linear, we mean that there is a linear function g with which f agrees at "most" points. Even with this relaxed requirement, the task seems to be hard. Another, perhaps more natural, and easily checkable definition of "approximately linear" is that for "most" pairs x, y, the function f satisfy $f(x+y) = f(x) + f(y)$. However, for the applications we will be looking at, the first definition will prove to be more useful. The question that then arises is what the relationship between the two notions is. We formalize this question below.

Let $\mathcal{F}_\ell$ be the set of linear functions from G to H. Define ϵ_f as:

$$\epsilon_f \stackrel{\text{def}}{=} \min_g \mathbf{P}_x \left[f(x) \neq g(x) \right].$$

Clearly, when $\epsilon_f = 0$, the function f is linear. Now, define δ_f as:

$$\delta_f \stackrel{\text{def}}{=} \mathbf{P}_{x,y} \left[f(x+y) \neq f(x) + f(y) \right].$$

Again, if $\delta_f = 0$, the function is linear (i.e., $\epsilon_f = 0$). What can we say about ϵ_f if δ_f is a small non-zero constant, say 0.1? In other words, how close to linear would f be in the first sense, if it is close to linear in the second sense?

The goal of linearity testing is to study ϵ_f as a function of δ_f. This problem was first considered by Blum, Luby and Rubinfeld [?] for general groups. With Coppersmith, they gave an almost complete picture [?]:

$$\delta_f \geq \min \left\{ \frac{\epsilon_f}{2}, \frac{2}{9} \right\}.$$

If G and H are vector spaces, we can use Fourier analysis to solve this problem. Kiwi [?] gave the following relation in the case $G = \mathbb{F}^n$ and $H = \mathbb{F}$, where $\mathbb{F}$ is a finite field: $\delta_f \geq \epsilon_f - o(1)$. In this inequality, the 'little-oh' is with respect to n, the dimension of G.

Today, we will show that $\delta_f \geq \epsilon_f$ when $G = \mathbb{F}_2^n$ and $H = \mathbb{F}_2$, $\mathbb{F}_2$ being the field of two elements.

The field $\mathbb{F}_2$ of two elements is usually represented by $\{0, 1\}$ with the addition being the exclusive OR operation. A particularly useful alternative representation for it as an additive group is $\{+1, -1\}$ with "addition" being the multiplication operation. The class $\mathcal{F}_\ell$ of linear functions may then be described as:

$$\mathcal{F}_\ell = \{f : \mathbb{F}_2^n \to \mathbb{F}_2 \mid \forall x, y \in \mathbb{F}_2^n,\ f(x \cdot y) = f(x) \cdot f(y)\}.$$

¿From standard linear algebra, it follows that $\mathcal{F}_\ell = \{L_\alpha\}_{\alpha \subseteq [n]}$, where the function L_α is defined as

$$L_\alpha(x) \stackrel{\text{def}}{=} \prod_{i \in \alpha} x_i,$$

and α is a subset of $[n] = \{1, \ldots, n\}$. It will also be useful for us to consider $\mathbb{F}_2$ as sitting inside the set of reals $\mathbb{R}$.

Consider the following inner product on the vector space of functions from $\mathbb{F}_2^n$ to $\mathbb{R}$: $\langle f, g \rangle = \frac{1}{2^n} \sum_x f(x) g(x)$. When the two functions take values in $\mathbb{F}_2$, we have

$$\begin{aligned} \langle f, g \rangle &= \mathbf{E}_x [f(x) g(x)] \\ &= \mathbf{P}_x [f(x) = g(x)] - \mathbf{P}_x [f(x) \neq g(x)] \\ &= 1 - 2\,\mathbf{P}_x [f(x) \neq g(x)] \end{aligned} \tag{6.1}$$

The set of functions $\mathcal{F}_\ell$ forms an orthonormal basis with respect to this inner product. For a function f, let $\hat{f}_\alpha$ denote its component along the basis vector L_α. In other words, $\hat{f}_\alpha = \langle f, L_\alpha \rangle$. By the Parseval relation, we have $\langle f, f \rangle = \sum_\alpha \hat{f}_\alpha^2$. Clearly, for any function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, we have $\langle f, f \rangle = 1$.

Going back to the quantities ϵ_f and δ_f we are interested in, observe that from equation (6.1) above,

$$\begin{aligned} \epsilon_f &= \min_\alpha \mathbf{P}_x [f(x) \neq L_\alpha(x)] \\ &= \frac{1}{2}(1 - \max_\alpha \hat{f}_\alpha). \end{aligned} \tag{6.2}$$

Now, $\delta_f = \mathbf{E}_{x,y} [I(x, y)]$, where $I(x, y)$ is the indicator function of the event $f(x \cdot y) \neq f(x) f(y)$. However, this event is the same as $f(x) f(y) f(x \cdot y) = -1$. Hence, we have $I(x, y) = \frac{1}{2}(1 - f(x) f(y) f(x \cdot y))$, and

$$\delta_f = \frac{1}{2}(1 - \mathbf{E}_{x,y} [f(x) f(y) f(x \cdot y)]). \tag{6.3}$$

We will show that

$$\mathbf{E}_{x,y}\left[f(x)f(y)f(x\cdot y)\right] \;\leq\; \max_{\alpha}\hat{f}_{\alpha},$$

which from equations (6.2) and (6.3) implies that $\epsilon_f \leq \delta_f$.

Expanding f in terms of the Fourier basis $\mathcal{F}_\ell$, and then using the linearity of the basis functions L_γ and the fact that $\mathbf{E}_x\left[g(x)h(x)\right] = \langle g, h\rangle$, we get

$$\begin{aligned}
\mathbf{E}_{x,y}\left[f(x)f(y)f(x\cdot y)\right] &= \mathbf{E}_{x,y}\left[\sum_{\alpha}\hat{f}_\alpha L_\alpha(x)\sum_{\beta}\hat{f}_\beta L_\beta(y)\sum_{\gamma}\hat{f}_\gamma L_\gamma(x\cdot y)\right] \\
&= \sum_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\mathbf{E}_{x,y}\left[L_\alpha(x)L_\beta(y)L_\gamma(x\cdot y)\right] \\
&= \sum_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\mathbf{E}_{x,y}\left[L_\alpha(x)L_\beta(y)L_\gamma(x)L_\gamma(y)\right] \\
&= \sum_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\mathbf{E}_{x}\left[L_\alpha(x)L_\gamma(x)\right]\,\mathbf{E}_{y}\left[L_\beta(y)L_\gamma(y)\right] \\
&= \sum_{\gamma}\hat{f}_\gamma^3.
\end{aligned}$$

We can now bound this quantity using the Parseval identity, and the fact that $\langle f, f\rangle = 1$:

$$\begin{aligned}
\sum_{\gamma}\hat{f}_\gamma^3 &\leq (\max_{\gamma}\hat{f}_\gamma)\sum_{\gamma}\hat{f}_\gamma^2 \\
&\leq \max_{\gamma}\hat{f}_\gamma.
\end{aligned}$$

As mentioned above, this implies that $\epsilon_f \leq \delta_f$.

Exercise 6.1 *Suppose we have a biased coin, with the probability of a '0' being* $\frac{1}{2}+\epsilon$ *and that of a '1' being* $\frac{1}{2}-\epsilon$. *Using the usual* $0, 1$ *notation and then the* $+1, -1$ *notation, prove that if we toss the coin* k *times, the probability of the parity of the results being* even *is* $\frac{1}{2}(1+(2\epsilon)^k)$.

What the exercise tells us is that no matter how badly biased a coin is, we can get a coin toss with arbitrarily small bias by taking the parity of a sequence of coin tosses of the original coin.

Exercise 6.2 *Let* $f, g, h : \mathbb{F}_2^n \to \mathbb{F}_2$. *Relate* $\mathbf{P}_{x,y}\left[f(x)g(y) = h(x\cdot y)\right]$ *to* $\max_\alpha\left|\hat{f}_\alpha\right|$, $\max_\beta\left|\hat{g}_\beta\right|$ *and* $\max_\gamma\left|\hat{h}_\gamma\right|$.

Note that when $f = g = h = L_\alpha$ or when $f = -g = -h = L_\alpha$, the probability is 1, and when $f = L_\alpha, g = L_\beta, h = L_\gamma$, where $\alpha \neq \beta$, the probability is $\frac{1}{2}$.

6.2 Probabilistically checkable proofs

The usual notion of a proof can be illustrated with the example of the *satisfiability problem.* To prove that a given boolean formula in 3CNF form is satisfiable, it suffices to give an assignment α that satisfies the formula as "proof." Given such a proof, it is easy to verify whether it satisfies the formula or not. Formally, the proof can be verified and accepted as a valid proof (or rejected) in polynomial time. On the other hand, a *probabilistically checkable proof* (PCP) consists of an *encoding* $A = E(\alpha)$ of the claimed satisfying assignment α. Such an encoding allows a verifier to examine the proof at only a few places and decide its validity. Formally, given a formula ϕ of size n and a probabilistically checkable "proof" A for its satisfiability, a polynomial time verifier can toss $r(n)$ coins, generate $q(n)$ queries $i_1, i_2, \ldots, i_{q(n)}$, and decide with high probability whether A encodes a satisfying assignment for ϕ or not on the basis of the bits $A(i_1), A(i_2), \ldots, A(i_{q(n)})$. Moreover, if the verifier accepts the proof with high probability, there is a decoding function D that can reconstruct a satisfying assignment α from A with positive probability. We now give some examples of such encodings.

6.2.1 The Hadamard code

The Hadamard code corresponding to an n-bit string α (interpreted now as the indicator of a subset of $[n]$) is the string $\{L_\alpha(x)\}_{x \in \mathbb{F}_2^n}$. The verifier for this code performs the linearity test described in the previous section: pick x, y uniformly at random from $\mathbb{F}_2^n$, and check if $A(x)A(y) = A(x \cdot y)$ (always interpreting the bits of A in the $+1, -1$ representation). If the verifier succeeds with probability $1 - \epsilon$, then from our analysis above, there is an α such that $\mathbf{P}_x\left[A(x) \neq L_\alpha(x)\right] \leq \epsilon$. Moreover, we have $\hat{A}_\alpha \geq 1 - 2\epsilon$, so we can use the following method to decode the string A: pick β with probability $\hat{A}_\beta^2$.

6.2.2 The Long Code

There is a dual view of the Hadamard code which arises from the observation that $L_\alpha(x) = L_x(\alpha)$. Here, we are identifying 0 with $+1$ and 1 with -1, as before. Thus, the Hadamard encoding of α is the sequence of values of all *linear* functions at the point α. We get the *long code* by extending this sequence to the sequence of values of *all* functions at the point α: $\text{LONG}_\alpha = \{f(\alpha)\}_{f : \mathbb{F}_2^n \to \mathbb{F}_2}$.

The long code may also be viewed as a restriction of the Hadamard code for 2^n-bit strings. Each function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ can be represented by a 2^n-bit string giving its values at all the points in the domain, and this string in turn specifies a subset of $[2^n]$. The point α then corresponds to a singleton subset of $[2^n]$, and its corresponding Hadamard code is clearly given by the αth bit of f as f ranges over all functions, as specified by the Long

Code. The notion of 'long code' then also applies to subsets $S = \{\alpha_1, \ldots, \alpha_k\}$ of $[2^n]$: $\textsc{Long}_S(f) = f(\alpha_1) \cdots f(\alpha_k)$.

Consider now a claimed encoding A of an n-bit string. In light of the second view of the long code presented above, it suffices to do the linearity test on A to verify its closeness to a valid codeword: pick $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ uniformly at random and check if $A(f \cdot g) = A(f)A(g)$. The following decoding algorithm is also then natural: pick a set $S \subseteq [2^n]$ with probability $\hat{A}_S^2$, and pick an element $\alpha \in S$ uniformly at random.

Now, when $A = \textsc{Long}_\alpha$, for some α, the verifier accepts with probability 1, and the decoder correctly outputs α. However, the verifier also accepts A with certainty when $A = \textsc{Long}_S$, for any set $S = \{\alpha_1, \ldots, \alpha_k\}$. In this case, the claimed proof A is not really committing to any particular string of the k, and ideally, we would like to reject such codewords which encode large sets with high probability. To accomplish this, we introduce a perturbation into the verification process: pick $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ uniformly at random, $\eta : \mathbb{F}_2^n \to \mathbb{F}_2$ such that for any β, $\eta(\beta) = 1$ with probability $1 - \epsilon$ and $\eta(\beta) = -1$ with probability ϵ, and then check if $A(f \cdot g \cdot \eta) = A(f)A(g)$.

With the above modification, if $A = \textsc{Long}_\alpha$ is an encoding of a singleton set, the verifier accepts with probability $1 - \epsilon$. Moreover, if $A = \textsc{Long}_S$ for $S = \{\alpha_1, \ldots, \alpha_k\}$, Exercise 6.1 tells us that we accept with probability $\frac{1}{2} + \frac{1}{2}(1 - 2\epsilon)^k$. What is the probability of acceptance for general A?

As in the case of linearity testing, we may use Fourier analysis to compute this probability. Let χ_S denote the codeword $\textsc{Long}_S$ corresponding to S. Now as before, we have

$$\mathbf{P}_{f,g,\eta}\left[A(f)A(g) = A(f \cdot g \cdot \eta)\right] = \frac{1}{2}(1 + \mathbf{E}_{f,g,\eta}\left[A(f)A(g)A(f \cdot g \cdot \eta)\right]).$$

The expected value above is then equal to

$$\sum_{S,T,U} \hat{A}_S \hat{A}_T \hat{A}_U \, \mathbf{E}_f\left[\chi_S(f)\chi_U(f)\right] \mathbf{E}_g\left[\chi_T(g)\chi_U(g)\right] \mathbf{E}_\eta\left[\chi_U(\eta)\right] = \sum_U \hat{A}_U^3 (1 - 2\epsilon)^{|U|}$$

Using the enhanced verification procedure, we have avoided the problem of accepting encodings of large sets. However, the codeword that most easily passes this test is $A \equiv 1$, which corresponds to committing to none of the strings α! To circumvent this problem, we allow only encodings that satisfy $A(-f) = -A(f)$ for all f. So that only such encodings pass the verification test, we partition the set of functions into two sets, one consisting of the negation of the functions in the other. Now, in doing our test, we only look up values of A at points in one of the sets of the partition. In other words, we consider only half the original long code as being available to us, the other half being implicitly given by the relation $A(-f) = -A(f)$. For such codewords, we clearly have $\hat{A}_\emptyset = 0$.

Exercise 6.3 *Show that for strings A such that $A(-f) = -A(f)$, the Fourier coefficients $\hat{A}_S$ corresponding to the* even *size subsets S are all* 0.