# THE FOURIER TRANSFORM ON FINITE GROUPS: THEORY AND COMPUTATION

ROHAN DANDAVATI

Abstract. This paper builds the theory of Fourier analysis on finite groups. First, Fourier analysis on the real domain is discussed. Then, Fourier analysis on finite abelian groups is developed. After this, a brief introduction to representation theory allows for the discussion of Fourier analysis on all finite groups. Algorithms are designed to perform these transforms, building to the efficient algorithm developed by Diaconis and Rockmore. Finally, an implementation of this algorithm for computing Fourier transforms on $S_n$ is demonstrated.

## Contents

## 1. Introduction

With far-reaching applications in numerous fields such as acoustics and medical imaging, Fourier analysis is an extremely powerful mathematical tool. Fourier analysis allows any "nice" function to be modeled as a sum of sine and cosine waves, by performing what is essentially a change of basis on the function. This moves the function into what is often called the *frequency domain*. One application of this transformation is in lossy compression for audio files. An audio signal, which gives intensity as a function of time, can be broken into its constituent frequencies using the Fourier transform. This allows for some of the less important frequencies—those at the higher range of the spectrum, or that have a small amplitude—to be discarded. This reduces the amount of data needed to encode the signal, while making sure that the information loss is difficult to perceive with the human ear.

*Date*: September 30, 2018.

The principles of Fourier analysis can be applied to functions mapping from arbitrary finite groups. In this case, the Fourier transform acts similarly, converting the function into its representation in a new basis. In some instances, this new representation can be far easier to work with. Researchers at the University of Chicago (led by Risi Kondor) have recently used Fourier analysis on the symmetric group in order to perform a similar type of "compression", allowing functions on the symmetric group to be stored and updated quickly and easily. This has applications for object-tracking in machine learning. Additional information on this application can be found in [9], while further discussion of applications of finite Fourier theory can be found in [13]. This paper will explore these extensions of Fourier theory. Following the development of Fourier transforms on finite groups, efficient algorithms for computing these transforms will be discussed, summarizing work done by Cooley, Tukey, Diaconis, and Rockmore [1] [5].

## 2. Preliminaries

More formality is needed for further discussion of Fourier analysis: what is a "nice" function? What properties should its domain and codomain have? How is a Fourier transform computed? To answer this questions, a few definitions are necessary.

**Definition 2.1.** The $L^2$-*norm* of a (measurable) function $f\colon \mathbb{X} \to \mathbb{C}$, where $\mathbb{X}$ is some measure space, is defined as $\|f\| := (\int_{\mathbb{X}} |f|^2)^{\frac{1}{2}}$.

*Remark* 2.2. In general, for any $p \in [1, \infty)$, the $L^p$ norm of a function is given as $\|f\|_p = (\int_{\mathbb{X}} |f|^p)^{\frac{1}{p}}$.

**Definition 2.3.** A function $f\colon \mathbb{X} \to \mathbb{C}$ is *Lebesgue square integrable* if $\|f\| < \infty$. The set of Lebesgue square integrable functions with domain $\mathbb{X}$ is denoted as $L^2_*(\mathbb{X})$.

**Definition 2.4.** A *Hilbert space* is an inner product space which is additionally a complete metric space under the induced metric (the induced metric defines the distance between two elements $x$ and $y$ as $\langle x-y, x-y\rangle^{\frac{1}{2}}$, where $\langle \cdot, \cdot \rangle$ denotes the inner product).

**Definition 2.5.** The *inner product* $\langle \cdot, \cdot \rangle$ on $L^2(\mathbb{X})$ is defined as $\langle f, g\rangle := \int_{\mathbb{X}} f\overline{g}$. This is a function mapping from $L^2_*(\mathbb{X}) \times L^2_*(\mathbb{X})$ to $\mathbb{C}$.

We want this inner product as defined above to truly be an inner product, so it is a good idea to check this. Linearity and conjugate symmetry are evident, so this leaves positive-definiteness. But here we run into a problem: let $A$ be any set of measure zero (for example, if $\mathbb{X} = \mathbb{R}$, we could let $A = \mathbb{Q}$), and let $f$ be the indicator function on $A$. Then $\int_{\mathbb{X}} f = \int_A 1 = 0$, by definition of $A$ being measure zero. Thus, $\langle \cdot, \cdot \rangle$ is not an inner product on $L^2_*(\mathbb{X})$! Luckily, it turns out that this is the only thing that can go wrong. Thus, as in usual mathematical style, we will ignore this issue by modifying the space we are working with.

**Definition 2.6.** Two measurable functions $f, g\colon \mathbb{X} \to \mathbb{Y}$ are said to be equal *almost everywhere* if $f = g$ except on a set of measure zero.

Now, almost everywhere equivalence defines an equivalence relation on $L^2_*(\mathbb{X})$. This allows us to improve upon our definition of $L^2$ space (and lets us get rid of that annoying asterisk).

**Definition 2.7.** $L^2(\mathbb{X}) = L^2_*(\mathbb{X})/\sim$, where $f \sim g$ if $f = g$ almost everywhere.

*Remark* 2.8. Note this small technicality of $L^2(\mathbb{X})$ formally being equivalence classes of functions equal almost everywhere is very frequently ignored; it only usually becomes relevant if you are actually concerned with the pointwise limit of functions. Usually, one can get by with thinking of an element of $L^2(\mathbb{X})$ as being an actual function, e.g., an element of $L^2_*(\mathbb{X})$. Thus, we will very frequently use this abuse of notation, referring to elements of $L^2(\mathbb{X})$ as functions.

To continue our above argument, on $L^2(\mathbb{X})$ it is true by construction that $\|f\| = 0$ if and only if $f = 0$, and thus we now have positive definiteness of the induced metric $\langle \cdot, \cdot \rangle$ on $L^2(\mathbb{X})$. This gives the first part of the following theorem

**Theorem 2.9.** *$L^2(\mathbb{X})$ is a Hilbert space, with the metric given by $\langle \cdot, \cdot \rangle$ as defined in Definition 2.5.*

*Remark* 2.10. The motivation for the weird definition of $L^2(\mathbb{X})$ is so that it becomes a Hilbert space, making $L^2(\mathbb{X})$ "nicer" than $L^2_*(\mathbb{X})$.

*Proof.* (of Theorem 2.9) From the previous discussion and construction of $L^2(\mathbb{X})$, we know it is an inner product space. Additionally, it is necessary to show that $L^2(\mathbb{X})$ is a complete metric space. To do so rigorously requires a good deal of additional theory. A complete discussion can be found in [12], p.67-68.

□

Now, let us shift our focus to the $L^2$ space of functions over a specific set. We will consider $L^2([0,1])$, which can also be interpreted as the set of functions $\mathbb{R} \to \mathbb{C}$ with period 1. Since $L^2([0,1])$ is a Hilbert space, we have a topological basis.

**Theorem 2.11.** *Define the function $e_k(x) = e^{2\pi i k x}$. Then the set $\{e_k \mid k \in \mathbb{Z}\}$ is an orthonormal topological basis of $L^2([0,1])$.*

*Proof.* First, to show orthonormality. Let $m, n \in \mathbb{Z}$, $m \neq n$. Then

$$\langle e_m, e_n \rangle = \int_{[0,1]} e_m \overline{e_n} = \int_0^1 e^{2\pi i m x} e^{-2\pi i n x} dx = \int_0^1 e^{2\pi (m-n) i x} dx.$$

Taking an antiderivative and evaluating at the endpoints, this gives

$$\int_0^1 e^{2\pi (m-n) i x} dx = \frac{1}{2\pi i (n-m)} (e^{2\pi i (n-m)} - e^0) = 0.$$

Finally,

$$\langle e_m, e_m \rangle = \int_0^1 e^{2\pi (m-m) i x} dx = \int_0^1 dx = 1,$$

meaning this is an orthonormal set.

Now we must prove that this set is a basis. We need to show that any element of $f \in L^2([0,1])$ can be written as a weighted sum of the $e_k$. But how would we figure out the weights? Assuming this set is a basis, it would make sense to weight each $e_k$ with the coefficient $\langle f, e_k \rangle$. After all, this is how we would represent a vector in $\mathbb{R}^n$ with the standard basis. But is this the best way to represent an element of $L^2([0,1])$? Suppose we have some set $A = \{a_k \mid k \in \mathbb{Z}\}$ where each $a_k \in \mathbb{C}$.

Let $S_N(f,A) := \left\| f - \sum_{k=-N}^{N} a_k e_k \right\|^2$. Now, perform the following manipulations (from [3]):

$$S_N(f,A) = \left\| f - \sum_{k=-N}^{N} a_k e_k \right\|^2 = \left\| f - \sum_{k=-N}^{N} \langle f, e_k\rangle e_k + \sum_{k=-N}^{N} (\langle f, e_k\rangle - a_k) e_k \right\|^2 .$$

Then, using the linearity and conjugate symmetry of the inner product, this can be manipulated to get that $S_N(f,A)$ is equal to

$$\left\| f - \sum_{k=-N}^{N} \langle f, e_k\rangle e_k \right\|^2 + \left\| \sum_{k=-N}^{N} (\langle f, e_k\rangle - a_k) e_k \right\|^2$$
$$+2Re(\langle f - \sum_{k=-N}^{N} \langle f, e_k\rangle, \sum_{k=-N}^{N} (\langle f, e_k\rangle - a_k) e_k\rangle).$$

As a result of the orthogonality of the $e_k$, the third term is zero. As the inner product is positive definite, it then must be that choosing $a_k = \langle f, e_k\rangle$ minimizes $S_N(f,A)$ and is therefore the best choice of coefficients for any finite representation of a function $f$ in this basis of $e_k$ functions. This shows the unicity of a function's representation in this basis. It is also necessary to show that a representation in this basis exists for each function in $L^2([0,1])$.

To show that the set of $e_k$ span $L^2([0,1])$, it is sufficient to show that $\lim_{N\to\infty} S_N(f,A) = 0$ (where $A$ is the set of all $e_k$). Without appealing to higher-level theory, this becomes a rather involved proof, so for the sake of brevity, we will refer to the proof given by Dym and McKean in [6], (p. 31 - 33). $\square$

Now that a topological basis has been established for $L^2([0,1])$, any $f \in L^2([0,1])$ can be written as $f = \sum_{k=-\infty}^{\infty} \langle f, e_k\rangle e_k$. This series representation of $f$ is known as the *Fourier series*. As this is really just a change of basis, we can define the Fourier transform $\hat{f}$ of a function $f$ as it's image in this new basis, a function that maps $\mathbb{Z} \to \mathbb{C}$:

$$\hat{f}(k) = \int_0^1 f(x) e^{-2\pi i k x} dx.$$

What about periodic functions with periods not equal to 1? By tweaking the frequency slightly, i.e. defining $e_k(x) = e^{\frac{2\pi i k x}{T}}$ for some period $T \in \mathbb{R}_+$ and normalizing the inner product by a factor of $\frac{1}{T}$, we can apply the above arguments to any periodic function in $L^2(\mathbb{R})$.

Now, we will turn our focus to expanding these tools to all functions in $L^2(\mathbb{R})$. To do so, we will take a limit of the period of an arbitrary function to infinity and see how this affects the function's Fourier transform (this method comes from discussion in [10]). Let's start with some function $f \in L^2(\mathbb{R})$, with period $T$. Adapting our Fourier transform as mentioned previously, we have

$$\hat{f}(k) = \frac{1}{T} \int_0^T f(x) e^{-\frac{2\pi i k x}{T}} dx$$

and

$$f(x) = \sum_{k=-\infty}^{\infty} \hat{f}(k) e^{\frac{2\pi i k x}{T}}.$$

Now, denote $\omega := \frac{2\pi}{T}$. Then, multiplying the right side of the equation above by $T(\frac{1}{T})$ gives

$$f(x) = \sum_{k=-\infty}^{\infty} T\hat{f}(k)e^{\omega ikx}\frac{1}{T} = \sum_{k=-\infty}^{\infty} T\hat{f}(k)e^{\omega ikx}\frac{\omega}{2\pi}$$
$$= \sum_{k=-\infty}^{\infty} \frac{\omega}{2\pi}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)e^{-\omega ikx}dx.$$

Taking $T \to \infty$, we get that $\omega \to 0$. Recognizing the above as a Riemann sum, and then defining

$$\hat{f}(y) := \int_{-\infty}^{\infty} f(x)e^{-2\pi ixy}dx$$

we get that

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(y)e^{2\pi ixy}dy.$$

Now, it can be seen that this change of basis into a representation as a sum of complex exponentials can be performed on all elements of $L^2(\mathbb{R})$. In the non-periodic case, we change from using discrete frequencies to continuous ones, and the sum turns into an integral.

*Remark* 2.12. One might note an apparent disparity in this transition from the periodic to the non-periodic case. In the case of a function with a finite period, we have $f\colon [0,T] \to \mathbb{C}$ and $\hat{f}\colon \mathbb{Z} \to \mathbb{C}$; in the limiting case, we instead get $f\colon \mathbb{R} \to \mathbb{C}$ and $\hat{f}\colon \mathbb{R} \to \mathbb{C}$. This is explained in the general theory of Fourier analysis on infinite (abelian) groups, but is beyond the scope of this paper.

## 3. Fourier Theory for All Finite Groups

### 3.1. **Abelian Groups.**

Though the previous section discussed elements of $L^2(\mathbb{R})$, nothing about the analysis depended heavily on the fact that the domain of these functions was $\mathbb{R}$. For this reason, Fourier theory can be extended to apply to any function $f\colon G \to \mathbb{C}$, where $G$ is some finite group. This finiteness assumption might seem to be an arbitrary distinction; after all, $\mathbb{R}$ is not finite, and yet we can still perform Fourier analysis on $L^2(\mathbb{R})$. This assumption is helpful for a few reasons. As $G$ is finite, there are no concerns of whether a function $G \to \mathbb{C}$ is "square integrable", as the function will always have a finite sum over the finite elements of the group. Additionally, finiteness means that we do not have to consider $L^2(G)$ as a set of equivalence classes. It is equal to the set of all maps from $G$ to $\mathbb{C}$—under the usual measure, modding out by almost everywhere equivalence has no effect on a finite set. Finally, the finiteness condition allows applications in computer science, as can be seen in section 4 (it is hard to run an algorithm on an input of infinite size!). Because of this, it will be assumed that any group referenced for the remainder of this paper will be finite.

As is often true with the study of groups, it is much easier to start off by considering abelian groups. A few definitions are necessary to do so.

**Definition 3.1.** A *(group) character* is a homomorphism from a group $G$ to the multiplicative group of $S^1$. Specifically, $\chi\colon G \to S^1$ is a group character if $\chi$ satisfies the following condition:

$$\chi(gh) = \chi(g)\chi(h)\ \forall g, h \in G.$$

**Lemma 3.2.** *If $\chi$ is a character of a (finite) group $G$, then $\chi(g)$ is a $|G|$-th root of unity for all group elements $g$ in $G$.*

*Proof.* Let $G$ be a group, $\chi$ a character of $G$. Let $g \in G$. Then $\chi(g^{|G|}) = \chi(id) = 1$, where *id* denotes the identity element of $G$. As $\chi$ is a homomorphism, $\chi(g^{|G|}) = \chi(g)^{|G|} = 1$, so $\chi(g)$ is a $|G|$-th root of unity. □

As before, the functions on a group $G$ can be made into an inner product space. Letting $f_a, f_b$ be functions from a group $G$ to the complex numbers, the inner product is defined as $\langle f_a, f_b\rangle = \sum_{g\in G} f_a(g)\overline{f_b(g)}$. As we are working over a finite group, this is identical to Definition 2.5.

**Theorem 3.3.** *The set of group characters of a (finite abelian) group $G$ is orthogonal in $L^2(G)$.*

*Proof.* Suppose $\chi_a \neq \chi_b$ are two characters of a (finite abelian) group $G$, meaning that there is some $g \in G$ with $\chi_a(g) \neq \chi_b(g)$. Then $\langle \chi_a, \chi_b\rangle = \sum_{g\in G} \chi_a(g)\overline{\chi_b(g)}$. Let $h \in G$. Then

$$\chi_a(h)\langle \chi_a, \chi_b\rangle = \sum_{g\in G} \chi_a(h)\chi_a(g)\overline{\chi_b(g)}.$$

Using the fact that $\chi_a$ is a group homomorphism and $G$ is abelian gives

$$\chi_a(h)\langle \chi_a, \chi_b\rangle = \sum_{g\in G} \chi_a(gh)\overline{\chi_b(g)}.$$

Using the bijection $g \mapsto gh$, the sum on the right side is then equivalent to

$$\sum_{g\in G} \chi_a(g)\overline{\chi_b(h^{-1}g)} = \sum_{g\in G} \chi_a(g)\chi_b(h)\overline{\chi_b(g)} = \chi_b(h)\langle \chi_a, \chi_b\rangle.$$

Therefore we get that

$$\chi_a(h)\langle \chi_a, \chi_b\rangle = \chi_b(h)\langle \chi_a, \chi_b\rangle.$$

As this is true for any $h$, and $\chi_a \neq \chi_b$, it must be that $\langle \chi_a, \chi_b\rangle = 0$. □

**Definition 3.4.** The *dual group* of an abelian group $G$, denoted as $\hat{G}$, is the set of characters of $G$.

**Lemma 3.5.** *Let $G$ be a finite abelian group. Then $|G| = |\hat{G}|$.*

*Proof.* First, suppose that $G$ is cyclic. Then $G = \mathbb{Z}/m\mathbb{Z}$ for some $m$ in $\mathbb{N}$. Let $k$ be an element of $\mathbb{Z}/m\mathbb{Z}$. Then $\chi_k(x) := e^{\frac{2\pi i k x}{m}}$ is a character, as $|\chi_k(x)| = 1$ for all $x$ in $G$, and $\chi_k(x + y) = \chi_k(x)\chi_k(y)$ for all $x, y$ in $G$. Such a character can be associated with each element of $G$, and each are distinct, as they all map the element 1 to a unique value. Therefore, it must be that $|\hat{G}| \geq |G|$. Additionally, as $\mathbb{Z}/m\mathbb{Z}$ is cyclic, generated by 1, any character is uniquely determined by the image of 1 under the character. By Lemma 3.2, the image must be an $m$-th root of unity, of which there are exactly $m$. Therefore we get that $|\hat{G}| \leq m = |G|$, which along with the earlier statement gives that $|G| = |\hat{G}|$ in the cyclic case.

Now, we must generalize to all finite abelian groups. Recall the structure theorem for finite abelian groups, which states that all finite abelian groups are the direct product of cyclic groups of prime power order. In other words, if $G$ is a finite abelian group, then $G = \mathbb{Z}/n_1\mathbb{Z} \times \mathbb{Z}/n_2\mathbb{Z} \times ... \times \mathbb{Z}/n_k\mathbb{Z}$. This means that any element $g$ can be represented as $(g_1, g_2, ..., g_k)$, where each $g_i$ is in the set $\{0, 1, ..., n_i - 1\}$.

Now define $\chi_g(x) := \chi_{g_1}(x_1)...\chi_{g_k}(x_k)$, where each $\chi_{g_i}$ is defined as in the cyclic case, and $x = (x_1, ..., x_k) \in G$. This is a map to the complex unit circle, as the product of complex numbers of norm 1 must also have norm 1. Additionally, it is a homomorphism, as each $\chi_{g_i}$ is a homomorphism in the cyclic case. Therefore a character can be identified with each $g \in G$. These characters are unique, as $G$ is generated by the elements $(1, 0, ..., 0), ..., (0, 0, ..., 0, 1)$, and any two characters will differ on where they map at least one of these basis elements. This gives that $|\hat{G}| \geq |G|$. Additionally, a basis element with a 1 in the $i$-th coordinate position must be an $n_i$-th root of unity. This implies that there are at most $n_1 n_2 ... n_k = |G|$ characters. □

**Theorem 3.6.** *Let $G$ be a finite abelian group. Then $G \cong \hat{G}$.*

*Proof.* Use the map $g \mapsto \chi_g$ as given in the previous proof. As shown previously, this maps generators of $G$ to generators of $\hat{G}$, and is a homomorphism, so it must be an isomorphism. □

*Remark* 3.7. The fact that $G$ is finite allows it to be self-dual. If $G$ were infinite, this would not necessarily be the case. One example of this is letting $G$ be the circle group, $\{z \in \mathbb{C} \mid |z| = 1\}$, under multiplication. It turns out that in this case, $\hat{G} = \mathbb{Z}$ (this was hinted at when we saw that $\hat{f} \colon \mathbb{Z} \to \mathbb{C}$ if $f \colon [0, 1] \to \mathbb{C}$). Here, not only is $G$ not isomorphic to $\hat{G}$, they do not even have the same cardinality!

**Theorem 3.8.** *The set of characters of a finite abelian group $G$ is a basis for $L^2(G)$.*

*Proof.* Using that $G$ is finite, we can appeal to a different, more obvious basis for $L^2(G)$, that of the "delta" functions. Define the following function:

$$\delta_g(x) = \begin{cases} 1 & x = g \\ 0 & \textit{otherwise.} \end{cases}$$

Then the set $\{\delta_g \mid g \in G\}$ spans $L^2(G)$, as any function $f$ can be written as

$$a_1 \delta_{g_1} + ... + a_n \delta_{g_n},$$

where each $a_i = f(g_i)$. It is also orthogonal, as the product $\delta_{g_i} \delta_{g_j} = 0$ for $g_i \neq g_j$. Therefore this set is a basis for $L^2(G)$, one with cardinality $|G|$. As proven in the previous two theorems, the set of characters of a group $G$ is an orthogonal set of order $|G|$, and therefore must be a basis for $L^2(G)$. □

By the previous theorems, any function on a finite abelian group $G$ can be written in terms of a Fourier series. This is done in the same way as the real case, using the equation

$$f = \sum_{g \in G} \langle f, \chi_g \rangle \chi_g.$$

The analog to the Fourier transform for an abelian group is then the function

$$\hat{f}(\chi) = \langle f, \chi \rangle = \sum_{g \in G} f(g) \overline{\chi(g)}.$$

Just like the Fourier transform on the real line, the equation given above transforms a function to basis components in a space where the bases are complex exponentials.

3.2. **Non-Abelian Groups.** Now that we have the tools of Fourier analysis on abelian groups, we will extend these practices to all finite groups. Unfortunately, there is no clear set of functions that can be used as a basis for functions on non-abelian groups (except for maybe the boring basis of just using delta functions). To find a new basis for functions on non-abelian groups, some basic representation theory is needed. Representation theory is a branch of math that allows the study of groups through the use of linear algebra. Some definitions are needed to formalize this.

**Definition 3.9.** A *representation* $(\rho, V)$ of a group $G$ is a group homomorphism $\rho\colon G \to \mathrm{GL}(V)$, where $V$ is a finite-dimensional vector space over $\mathbb{C}$. Here, $\mathrm{GL}(V)$ refers to the set of invertible linear transformations on $V$.

**Definition 3.10.** Let $(\rho, V)$ be a representation. Then $d_\rho := \dim V$ is used to denote the dimension of this representation.

**Example 3.11.** When looking for examples using non-abelian groups, it is usually helpful to look at the smallest non-abelian group, $S_3$, the group of permutations on 3 elements. $S_3$ can be represented with what's known as the *permuatation representation*, a map from $S_3$ to $\mathrm{GL}(\mathbb{C}^3)$, taking each element of $S_3$ to its corresponding permutation matrix. For example, the element (1 2) maps to the following matrix.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

It can be seen that both (1 2) and its image under this map represent the same permutation by performing matrix multiplication using the generic element in $\mathbb{C}^3$.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} b \\ a \\ c \end{bmatrix}$$

Therefore, the matrix representation of (1 2) actually transposes the first two coordinates of a vector in $\mathbb{C}^3$. Using this example, an alternative characterization of a representation can be seen—a representation can also be understood as a group action of $G$ on $V$. Making this example explicit using cycle notation, with $e$ representing the identity element, the map of the permutation representation of $S_3$ is as given below.

$$e \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (1\ 2) \mapsto \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (1\ 3) \mapsto \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$(2\ 3) \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad (1\ 2\ 3) \mapsto \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad (1\ 3\ 2) \mapsto \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

**Example 3.12.** Any group $G$ acts on itself by left multiplication, where $g \cdot h = gh$. Identifying the set of elements of $G$ with a basis set of some vector space $V$, define the map $\rho\colon G \to GL(V)$ by identifying each element with its action on these basis vectors given by left multiplication. Then $(\rho, V)$ is a representation of $G$, known as the *left regular representation.*

**Definition 3.13.** Let $(\rho, V)$ be a representation mapping some group $G$ to $GL(V)$. Then a *subrepresentation* $(\pi, W)$ is a representation of $G$ where $W \subseteq V$ such that $\rho(g)w \in W$ for all $w \in W$, and

$$\rho(g)|_W = \pi(g) \ \ \forall g \in G.$$

*Remark* 3.14. In other words, this means that $(\pi, W)$ is a representation where $W$ is a subspace of $V$ stable under the image of $G$, and $\pi$ is equal to the restriction of $\rho$ to $W$. Therefore, if we know $(\rho, V)$, $W$ gives the map $\pi$ by restriction. Because of this, it is common to refer to $W$ alone as a subrepresentation of $(\rho, V)$.

**Example 3.15.** Let $(\rho, V)$ be a representation. Denote $\pi_0 \colon G \to \{0\}$ as the map sending every element of $G$ to 0. Then $(\pi_0, \{0\})$ is a subrepresentation of $(\rho, V)$.

**Example 3.16.** Consider the permutation representation of $S_3$ given previously. Then the subspace of $\mathbb{R}^3$ spanned by the vector $(1, 1, 1)$ gives the vector space of a subrepresentation of $\mathbb{R}^3$. This is because any permutation of the entries of this vector acts as the identity.

**Definition 3.17.** A representation $(\rho, V)$ (where $V \neq \{0\}$) is called *irreducible* if its only subrepresentations are $(\rho, V)$ and $(\pi_0, \{0\})$.

**Example 3.18.** The permutation representation of $S_3$ given previously is reducible, as it has a subrepresentation that is neither 0 nor $\mathbb{R}^3$. $S_3$ has another representation which is irreducible. Recall that $S_3$ is isomorphic to $D_6$, the group of symmetries of the equilateral triangle—this can be seen by labeling the vertices of the triangle and considering the symmetries as permutations of these vertices. Also, recall that $D_6$ is generated by the elements $r$, the counterclockwise rotation by $\frac{2\pi}{3}$, and $s$, a fixed reflection. These satisfy the relations $r^3 = s^2 = e$ and $rs = sr^{-1}$. Embedding the equilateral triangle in $\mathbb{R}^2$ and using the standard basis, the matrices representing these operations are:

$$r \mapsto \begin{bmatrix} \cos(\frac{2\pi}{3}) & -\sin(\frac{2\pi}{3}) \\ \sin(\frac{2\pi}{3}) & \cos(\frac{2\pi}{3}) \end{bmatrix}, \quad s \mapsto \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

It can be seen that $r$ and $s$ as given above satisfy the necessary constraints, and therefore define a representation of $D_6$, and equivalently one of $S_3$. This representation has no subrepresentations other than $(\pi_0, \{0\})$ — if it did, this would mean that it had a one-dimensional subrepresentation. However, any line through the origin could not be fixed by $r$, so this representation could not have a one-dimensional subrepresentation. Therefore this representation is irreducible.

**Definition 3.19.** Let $(\rho, V)$ and $(\pi, W)$ be representations of a group $G$. Then the *direct sum* of these representations is a map $\phi \colon G \to \mathrm{GL}(V \oplus W)$, satisfying that $\phi(g)(v, w) = (\rho(g)(v), \pi(g)(w))$ for $(v, w)$ in $V \oplus W$.

**Theorem 3.20.** *Let $G$ be a finite group. Then if $(p, V)$ is a finite dimensional representation of $G$ (meaning $V$ is finite dimensional), it can be written as the direct sum of a finite number of irreducible representations. More concretely, there are irreducible representations $(\pi_1, V_1), \ldots, (\pi_k, V_k)$ with $V = V_1 \oplus, \ldots, \oplus V_k$, and in* *an appropriate basis, $\rho(g)$ can be written in block diagonal form, as below*

$$\rho(g) = \begin{bmatrix} \pi_1(g) & & & \\ & \pi_2(g) & & \\ & & \ddots & \\ & & & \pi_k(g) \end{bmatrix}.$$

*Proof.* Let $G$ be a finite group, and let $(\rho, V)$ a representation. We will prove this by induction on dimension. In the 1-dimensional case, $(\rho, V)$ must be irreducible, as $V$ could have no non-zero subspace. Now, suppose that $d_\rho > 1$ and that the theorem has been proven true for all dimensions less than $d_\rho$. If $(\rho, V)$ is irreducible, then we are done. Suppose $(\rho, V)$ is reducible. Then there must be some non-trivial subrepresentation $(\pi, W)$ of $(\rho, V)$, with $W \subsetneq V$ and $W \neq \{0\}$. By the induction hypothesis, it must be that $(\pi, W)$ decomposes into a direct sum of irreducible representations. Now consider the space $V/W$. Let an element $g \in G$ operate on a coset of $V/W$, $v + W$, by $g \cdot (v + W) = \rho(g)(v) + W$. This is an action on $V/W$, as $\rho(g)$ preserves $W$, since $(\pi, W)$ is a subrepresentation of $(\rho, V)$. Therefore, $\rho$ defines a representation $(\rho', V/W)$. Again by the induction hypothesis, it must be that $(\rho', V/W)$ also decomposes as a direct sum of irreducible representations. This then gives $V = W \oplus V/W$, and

$$\rho(g) = \begin{bmatrix} \pi(g) & \\ & \rho'(g) \end{bmatrix}$$

for all $g \in G$. As both $(\pi, W)$ and $(\rho', V/W)$ can be decomposed into a direct sum of irreducible representations, it must be that the same can be done with $(\rho, V)$. $\square$

**Example 3.21.** To see this theorem in use, consider the example of the permutation representation of $S_3$. Refer to this representation as $(\rho, \mathbb{C}^3)$. It was stated previously that the subspace of $\mathbb{C}^3$ generated by the vector $(1, 1, 1)$ is preserved under multiplication by $\rho(g)$ for all $g \in S_3$. Now, consider $\mathbb{C}^3/\langle(1, 1, 1)\rangle$. This is spanned by the vectors $(1, -1, 0), (0, 1, -1)$. Observing the action of the elements of $S_3$ on these basis vectors gives what is known as the "standard representation" of $S_3$ (in general, the standard representation of $S_n$ is determined through this procedure). Combining these two representations, one can rewrite the permutation representation of $S_3$ in block diagonal form, with the upper left $1 \times 1$ block giving the trivial representation (mapping each element to 1), and the bottom right $2 \times 2$ block giving the standard representation, shown below.

$$e \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (1\,2) \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \qquad (1\,3) \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$(2\,3) \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \qquad (1\,2\,3) \to \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & -1 \end{bmatrix} \qquad (1\,3\,2) \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

What is the purpose of working with these representations? We will soon see that representations of a finite group $G$ are a way of realizing an alternate basis for functions $G \to \mathbb{C}$, just as characters were used in the abelian case. Recognizing that a group character is just a one-dimensional representation makes these tools consistent between abelian and non-abelian cases. A little more theory is needed before we will be able to prove the full utility of representations.

**Definition 3.22.** Let $(\rho, V)$ and $(\pi, W)$ be representations, and $L\colon V \to W$ be a linear map. Then $L$ *intertwines* $\rho$ *and* $\pi$ if

$$L\rho(g) = \pi(g)L \quad \forall g \in G.$$

**Definition 3.23.** Let $(\rho, V)$ and $(\pi, W)$ be representations. Then they are said to be *equivalent* if there exists some vector space isomorphism $L$ that intertwines $\rho$ and $\pi$.

**Example 3.24.** The standard representation of $S_3$ is equivalent to the representation of $S_3$ given in Example 3.18.

**Proposition 3.25.** *Let $L\colon V \to W$ intertwine $(\rho, V)$ and $(\pi, W)$. The restriction of $\rho$ to the kernel of $L$ gives a subrepresentation of $\rho$. The restriction of $\pi$ to the image of $L$ gives a subrepresentation of $\pi$.*

*Proof.* Let $x \in \ker L$. Then $L(\rho(g)(x)) = \pi(g)(L(x))$ as $L$ intertwines $\rho$ and $\pi$. Then, as $x \in \ker L$, this implies that $L(\rho(g)(x)) = 0$ for all $g$ in $G$. So $\rho(g) \ker L \subseteq \ker L$, and therefore the restriction to $\ker L$ induces a subrepresentation of $\rho$.

A similar procedure can be followed to prove the second part of the statement. Let $y \in \text{Im } L$. This means that $y = Lx$ for some $x \in V$. Then $\pi(g)(y) = \pi(g)(L(x)) = L(\rho(g)(x))$ for all $g$ in $G$, as $L$ intertwines the two operators. Therefore, $\pi(g)(y) \in \text{Im } L$ for all $y \in \text{Im } L$, meaning the image of $L$ induces a subrepresentation of $\pi$. □

**Lemma 3.26.** *(Schur's Orthogonality Lemma) Let $(\rho, V)$, $(\pi, W)$ be two irreducible representations. Let $L\colon V \to W$ intertwine $\rho$ and $\pi$. Then $L$ is either the 0 map or an isomorphism.*

*Proof.* (from [13], Chapter 15, Lemma 1) Suppose $L$ is not the 0 map. Then $\ker L$ is a subrepresentation of $\rho$ by the previous proposition, and $\ker L \neq V$, so it must be by irreducibility of $\rho$ that $\ker L = \{0\}$. Therefore $L$ is injective. Furthermore, as the image of $L$ is not equal to $\{0\}$, and the image must give a subrepresentation of $\pi$, it must be that $\text{Im } L = W$. Therefore, $L$ is an isomorphism. □

**Theorem 3.27.** *Let $(\rho, \mathbb{C}^a)$ and $(\pi, \mathbb{C}^b)$ be inequivalent irreducible representations of a finite group $G$. Let $\rho_{n,m}\colon G \to \mathbb{C}$ be the function mapping each element $g \in G$ to the element in the $n$-th row and $m$-th column of $\rho(g)$. Define $\pi_{i,j}$ similarly. Then $\rho_{n,m}$ and $\pi_{i,j}$ are orthogonal. More formally,*

$$\langle \rho_{n,m}, \pi_{i,j} \rangle = \sum_{g \in G} \rho_{n,m}(g)\overline{\pi_{i,j}(g)} = 0.$$

*Proof.* (from [13], Chapter 15, Theorem 1) Let $M$ be some linear mapping $\mathbb{C}^b \to \mathbb{C}^a$. Then define

$$N := \sum_{g \in G} \rho(g) M \pi(g^{-1}).$$

Now, let $y \in G$. Then $\rho(y)N = \rho(y) \sum_{g \in G} \rho(g) M \pi(g^{-1}) = \sum_{g \in G} \rho(yg) M \pi(g^{-1})$, as $\rho$ is a homomorphism. The next manipulation is similar to one performed in the proof of orthogonality of group characters—we recognize that as we iterate through elements $g \in G$, $gy$ also works as an index of $G$. Then, we get

$$\rho(y)N = \sum_{gy \in G} \rho(gy)M\pi((yg)^{-1}y) = \left( \sum_{gy \in G} \rho(gy)M\pi((yg)^{-1}) \right) \pi(y) = N\pi(y).$$

As this holds for any $y \in G$, $N$ is an intertwining operator between $(\rho, V)$ and $(\pi, W)$. Therefore, by Schur's orthogonality lemma, $N$ must be either the 0 map or an isomorphism. It must not be an isomorphism, as it was assumed that $(\rho, V)$ and $(\pi, W)$ were inequivalent, so $N$ must therefore be the 0 map. As we stated that $M$ could be any map from $W \to V$, let $M$ have a 1 in the $m$-th row and the $j$-th column, and be otherwise filled with zeros. Then the entry of the $n$-th row and the $i$-th column of $N$ must be 0. As this entry is equal to $\sum_{g \in G} \rho_{n,m}(g)\overline{\pi_{i,j}(g)} = \langle \rho_{n,m}, \pi_{i,j} \rangle$, it must be that

$$\langle \rho_{n,m}, \pi_{i,j} \rangle = 0.$$

□

**Lemma 3.28.** *Let $(\rho, V)$ be an irreducible representation, and let $L\colon V \to V$ intertwine $\rho$ with itself. Then $L = xI$, where $x$ is a scalar, and $I$ is the identity linear transformation.*

*Proof.* Let $(\rho, V)$, $L$ as stated. Let $x$ be an eigenvalue of $L$ and $W$ the corresponding eigenspace. Let $w \in W$. Then $L(\rho(g)(w)) = \rho(g)(L(w))$, as $L$ intertwines $\rho$ with itself. Since $w \in W$, $L(w) = xw$. Therefore $\rho(g)(L(w)) = \rho(g)(xw) = x\rho(g)(w)$, meaning that $L(\rho(g)(w)) = x\rho(g)(w)$, so $\rho(g)(w) \in W$ for all $g$ in $G$. As $W$ is stable under $\rho$, it must induce a subrepresentation of $\rho$. The irreducibility of $\rho$ then implies that $W$ must either be $\{0\}$ or $V$, meaning that $L$ is equal to $xI$. □

**Theorem 3.29.** *Let $(\rho, \mathbb{C}^a)$ be an irreducible representation of $G$. Then $\langle \rho_{n,m}, \rho_{i,j} \rangle \neq 0$ if and only if $n = i$ and $m = j$.*

*Proof.* Similar to before, define

$$N := \sum_{g \in G} \rho(g)M\rho(g^{-1}),$$

where $M$ is any linear mapping $\mathbb{C}^a \to \mathbb{C}^a$. Then, $N$ intertwines $\rho$ with itself, as this is just a special case of part of the proof of Theorem 3.27. By the above lemma, this means that $N = xI$ for some $x \in \mathbb{C}$. Taking the trace of $N$, we get

$$|G| \operatorname{tr}(M) = \operatorname{tr}(xI),$$

where $I$ is the identity matrix, as the trace of a matrix is preserved under conjugation. The identity above implies that $|G| \operatorname{tr}(M) = xa$, where $a$ is the dimension of $\rho$. As before, let $M$ have a 1 in the $m$-th row and the $j$-th column, and otherwise be filled with zeros. This gives that the $n$-th row and the $i$-th column of $N$ are equal to $\sum_{g \in G} \rho_{n,m}(g)\overline{\rho_{i,j}(g)} = \langle \rho_{n,m}, \rho_{i,j} \rangle$.

We have that $x \neq 0$ if and only if $\operatorname{tr}(M) \neq 0$, which occurs if and only if $m = j$. Additionally, in the case that $x \neq 0$, the value in the $n$-th row and the $i$-th column of $N$ is non-zero if and only if $n = i$. So $\langle \rho_{n,m}, \rho_{i,j} \rangle \neq 0$ if and only if $n = i$ and $m = j$ □

Now that, from the two theorems above, we have that matrix entries are orthogonal, we are almost ready to define the general Fourier transform on finite groups. Only the following definitions and proposition are needed.

**Definition 3.30.** The *dual* of $G$, denoted as $\hat{G}$, is defined as the set of all irreducible representations of $G$.

**Proposition 3.31.** *Let $G$ be a finite group. Then*

$$\sum_{\rho \in \hat{G}} d_\rho^2 = |G|.$$

The proof of this proposition requires a good deal more theory than will be needed as we continue. For a full proof, please see [13], p.256. The previous theorems along with this proposition demonstrate that the matrix entries of the irreducible representations are an orthogonal set of size $|G|$. Recall that, just like in the abelian case, the set of delta functions form a orthogonal basis of $L^2(G)$, with this basis being of size $|G|$. Therefore, as the set of matrix entries are an orthogonal set of size $|G|$, they must also form a basis. Using this fact, we define the Fourier transform for all finite groups as the following:

$$\hat{f}(\rho) = \sum_{g \in G} f(g)\rho(g).$$

Unlike our previous iterations of the Fourier transform, this is a function on representations that returns linear maps. By noting that abelian group characters are one-dimensional representations, and therefore so are their conjugates, we see that this definition is consistent with the finite abelian case.

## 4. Computing Fourier Transforms

In the previous sections we have managed to develop the tools of Fourier analysis on $L^2(\mathbb{X})$, where $\mathbb{X}$ can be $\mathbb{R}$ (or any closed subinterval of $\mathbb{R}$), or any finite group. These tools have many potential applications. However, Fourier analysis is only useful if a Fourier transform can actually be computed. In many cases the integral

$$\int_{-\infty}^{\infty} f(x)e^{-2\pi ixy}dx = \hat{f}(y)$$

is difficult to compute. This is especially true in the case that $f$ cannot be expressed in a closed form, or has no formula at all, and is attained through observation. In order to approximate the Fourier transform in this case, a sample of the function can be taken. Suppose the function $f$ is sampled at $n$ points, $x_0, \ldots, x_{n-1}$. Then the Fourier transform of $f$ can be approximated at $n$ "frequency components", using

$$\hat{f}(y_j) = \sum_{i=0}^{n-1} f(x_i)e^{\frac{-2\pi i x_i y_j}{n}}. \tag{4.1}$$

As can be seen by referring back to the section 3.1, this is just the formula for the Fourier transform on the cyclic group of order $n$. Therefore, Fourier transforms on finite groups can be useful in computing real Fourier transforms. Unfortunately, computing these Fourier transforms directly is slow. Since computing each value of the Fourier transform involves $O(n)$ operations, entirely determining the Fourier transform of a function on a cyclic group using the naïve algorithm requires $O(n^2)$

operations. For many practical applications, this is prohibitively inefficient. One way to cut down this run-time is to use inherent symmetries of both the cyclic group and the Fourier transform itself.

## 4.1. **The Cooley-Tukey Fast Fourier Transform (FFT).**

We will start with what is known as the "Cooley-Tukey" algorithm, which computes the Fourier transform on a finite cyclic group of order $n$, where $n$ is a power of 2. This algorithm was first published in 1965, but was actually known to Gauss in 1805, though it was only published posthumously [7]. It is included under the umbrella of a larger group of similar efficient algorithms for computing the Fourier transform on finite cyclic groups, that are commonly known as the Fast Fourier Transform (FFT).

To begin, return to (4.1). This sum can be split into two separate sums, one over even $i$ values, and one over odd values, as shown below:

$$\hat{f}(y_j) = \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i}) e^{\frac{-2\pi(2x_i)y_j}{n}} + \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i+1}) e^{\frac{-2\pi(2x_i+1)y_j}{n}}.$$

Factoring out and rearranging these sums gives a much more suggestive form:

$$\hat{f}(y_j) = \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i}) e^{\frac{-2\pi(x_i)y_j}{\frac{n}{2}}} + e^{\frac{-2\pi ij}{n}} \sum_{i=0}^{\frac{n}{2}-1} f(x_{2i+1}) e^{\frac{-2\pi(x_i)y_j}{\frac{n}{2}}}.$$

The first summation is a Fourier transform on a cyclic group of size $\frac{n}{2}$. The second term is also a Fourier transform, but with a root of unity tacked on the front. More specifically, the first sum is the Fourier transform of the function on the even-indexed elements of the group, while the second is a scaling of the transform of the function on the odd-indexed elements of the group. Denoting these functions as $f_e$ and $f_o$, respectively, this gives the following equation:

$$\hat{f}(y_j) = \hat{f}_e(y_j) + e^{\frac{-2\pi ij}{n}} \hat{f}_o(y_j).$$

This means that if the Fourier transforms of $f_e$ and $f_o$ are calculated previously, $\hat{f}$ can be recovered with just one multiplication and addition per group element, for a total of $O(n)$ operations. If $T(n)$ denotes the run-time of computing the Fourier transform on $n$ elements, this gives the following recurrence relation:

$$T(n) \leq 2T\left(\frac{n}{2}\right) + O(n).$$

To further speed up the calculations, $\hat{f}_e$ and $\hat{f}_o$ can themselves be calculated recursively in this same manner. This recursive process terminates when $n = 1$, as in this case, the Fourier transform of a function on one element is just the function itself. This type of algorithm is known as a "divide and conquer" algorithm, as it divides one large problem into smaller subproblems, solves the subproblems individually and uses these solutions to construct a solution to the large problem. This procedure gives an algorithm that performs $\log_2(n)$ "layers" of recursion, each with a run-time of $O(n)$, meaning that the Cooley-Tukey algorithm has a run-time of $O(n \log_2(n))$. This is a massive improvement from the naïve $O(n^2)$ algorithm. So even though this algorithm comes from just rearranging the Fourier transform, it provides a considerable speedup. For sufficiently large functions, this can be the difference between minutes and days.

For a more explicit description of this algorithm, refer to the pseudocode given below, for a function called $fft()$. In this code, the functions *splitEvens* and *splitOdds* are helper functions to extract the values of $f$ on even and odd valued elements. As array storage is language-dependent, implementations of these helper functions would be language-specific.

```
**Input**: f an array storing the values of the function $f$; n, the order of $G$
**Output**: $\hat{f}$, an array containing the values of the Fourier transform of $f$
**if** $n = 1$ **then**
    return f // base case, where $\hat{f} = f$
**else**
    $evens \leftarrow splitEvens(f)$ // utility functions to pull all even/odd
    $odds \leftarrow splitOdds(f)$   // elements from the array
    $t_{evens} \leftarrow fft(evens, \frac{n}{2})$ // computing F.T. recursively
    $t_{odds} \leftarrow fft(odds, \frac{n}{2})$
$\hat{f} \leftarrow \emptyset$
**for** $j \in \{1, 2, \ldots, \frac{n}{2} - 1\}$ **do**
    // using recurrence relation to build $\hat{f}$
    $\hat{f}[j] \leftarrow t_{evens}[j] + e^{\frac{-2\pi ij}{n}} t_{odds}[j]$
    $\hat{f}[j + \frac{n}{2}] \leftarrow t_{evens}[j] - e^{\frac{-2\pi ij}{n}} t_{odds}[j]$
*return* $\hat{f}$
```

4.2. **An Algorithm for All Finite Groups.** The Cooley-Tukey algorithm is quick, but is limited in scope: it only works for groups that are cyclic and of a certain order. However, the idea behind the Cooley-Tukey algorithm is powerful—that the symmetries of a group can be exploited to save on computational effort. This idea was used by Diaconis and Rockmore in 1990 to develop an efficient algorithm that works on all finite groups [5]. The way this algorithm works is to form a recurrence relation, as was done in the Cooley-Tukey algorithm. One natural way to form a recurrence relation for an algorithm on a group is to use a descending chain of subgroups. More formally, a chain is some series of subgroups $H_1, \ldots, H_n$ of a group $G$ with $G \supset H_1 \supset H_2 \supset \cdots \supset H_n = \{id\}$. Then, the Fourier transform can be calculated recursively, crawling down the chain to build the Fourier transform on $G$. This is what was done in the Cooley-Tukey algorithm, using the chain $\mathbb{Z}/2^n\mathbb{Z} \supset \mathbb{Z}/2^{n-1}\mathbb{Z} \supset \cdots \supset \mathbb{Z}/2\mathbb{Z} \supset \{id\}$.

To give a more complete formulation of Diaconis and Rockmore's algorithm, we will return to the definition of the Fourier transform on a finite group:

$$\hat{f}(\rho) = \sum_{g \in G} f(g)\rho(g).$$

Let $H$ be a subgroup of $G$, with the index of $[G : H] = n$.

*Remark* 4.2. There exist a set of elements $\{g_1, \ldots, g_n\}$ such that $\sqcup_{i=1}^{n} g_i H = G$. We will call this a set of *coset representatives* of $H$ in $G$. This is also known as a *left transveral* of $H$ in $G$.

Choosing such a $g_1, \dots, g_n$, the sum above can be decomposed into the following equation:

$$\hat{f}(\rho) = \sum_{i=1}^{n} \sum_{h \in H} f(g_i h)\rho(g_i h).$$

Defining the function $f_i(g) := f(g_i g)$ and using the fact that $\rho$ is a homomorphism gives the rearrangement

$$\hat{f}(\rho) = \sum_{i=1}^{n} \rho(g_i) \sum_{h \in H} f_i(h)\rho(h).$$

So far, this is just a rewriting of the definition. The breakthrough comes from the fact that the inner sum is actually a Fourier transform! More specifically, it is the Fourier transform of the function $f_i$ on the representation $\rho$ restricted to the subgroup $H$. Denoting the restriction of $\rho$ to $H$ as $\rho|_H$, we can now write the Fourier transform of $\rho$ as below:

$$\hat{f}(\rho) = \sum_{i=1}^{n} \rho(g_i)\hat{f}_i(\rho|_H).$$

One issue appears here: $\rho|_H$ is not necessarily irreducible, as can be seen in the following example.

**Example 4.3.** Let $G = S_3$ and let $H$ be the natural embedding of $S_2$ into $S_3$ ($H = \{e, (12)\}$). Then, it can be seen that $\{e, (123), (132)\}$ are coset representatives of $H$, as $eH \cup (123)H \cup (132)H = \{e, (12)\} \cup \{(123), (13)\} \cup \{(132), (23)\} = S_3$. Consider $\rho_{std}$, the standard representation on $S_3$. Then the restriction $\rho_{std}|_H$ is the following map:

$$e \mapsto \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (12) \mapsto \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}.$$

This is a representation, but not an irreducible one—it must be reducible, as the only irreducible representations on $S_2$ are one-dimensional, since it is abelian.

The above example illustrates that $\rho|_H$ is not always irreducible when considered as a representation of $H$. However, when $\rho$ is finite dimensional, by Theorem 3.20 it can be written as a direct sum of irreducible representations. In the example, $\rho_{std}|_{S_2}$ maps $S_2$ to $\mathbb{C}^2$, which itself is equal to $\mathbb{C} \times \mathbb{C}$. Therefore, it can be seen that $\rho_{std}|_{S_2}$ is equal to a direct sum of two irreducible representations on $\mathbb{C}$. These happen to be the trivial and sign representations. Therefore, in the appropriate basis, the map $\rho_{std}|_{S_2}$ can be written as:

$$e \to \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad (12) \to \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

While $\rho|_H$ may not be irreducible, it can be constructed in block-diagonal form using other irreducible representations. This means that the Fourier transform on these irreducible representations can be calculated and used to construct $\hat{f}_i(\rho|_H)$. Therefore, we need to determine a set of irreducible representations of $H$ such that every representation (of $H$) arising from restriction of an irreducible representation of $G$ can be written as a direct sum of elements of this set. Then, it is sufficient to take the Fourier transform of $\hat{f}_i$ on this set of representations, and use this to build $\hat{f}_i(\rho|_H)$ through direct summation.

As summarized by Diaconis [4], this algorithm reduces to a few basic steps.

- First, choose a subgroup $H$ of $G$. Determine some left transversal $\{g_1, \dots, g_n\}$ of $H$ in $G$.
- For each of the $g_i$, take the Fourier transform of $f_i$ on some set of irreducible representations of $H$.
- For each $\rho \in \hat{G}$, use the results from the previous step to construct $\hat{f}_i(\rho|_H)$. Given an appropriate choice of a basis and a set of irreducible representations of $H$ from the previous step, these matrices can be constructed in block-diagonal form, allowing many of these blocks to be used multiple times in this step—this allows for computational savings!
- Finally, sum over the $\hat{f}_i(\rho|_H)$, multiplying them by $\rho(g_i)$ to construct $\hat{f}(\rho)$ for each $\rho$ in $\hat{G}$.

Let's try a simple example.

**Example 4.4.** Let $G = \mathbb{Z}/2^n\mathbb{Z}$, for some $n \in \mathbb{N}$. Let the subgroup $H$ be $\mathbb{Z}/2^{n-1}\mathbb{Z}$, with the natural embedding being $H = \{0, 2, 4, \dots, (2^{n-1} - 1)2\}$. Here, $g_1 = 0$, $g_2 = 1$ are a set of coset representatives, as $(0 + H) \cup (1 + H) = G$. Using these choices, we will see how the general algorithm allows us to rearrange this transform. We will start with the original definition of the Fourier transform on G:

$$\hat{f}(j) = \sum_{m=0}^{2^n-1} f(m) e^{\frac{-2\pi i m j}{2^n}}.$$

Now, rearranging the sum using the subgroups and the coset representatives:

$$\hat{f}(j) = \sum_{k=0}^{1} e^{\frac{-2\pi i k j}{2^n}} \sum_{m=0}^{2^{n-1}-1} f_k(2m) e^{\frac{-2\pi i (2m) j}{2^n}}.$$

Using the fact that $f_k(x) = f(k + x)$, as $\mathbb{Z}/2^n\mathbb{Z}$ uses additive notation, we get the following equation, including a few further rearrangements.

$$\hat{f}(j) = \sum_{k=0}^{1} e^{\frac{-2\pi i k j}{2^n}} \sum_{m=0}^{2^{n-1}-1} f(k + 2m) e^{\frac{-2\pi i m j}{2^{n-1}}}$$

When $k = 0$, this sums over the even valued elements of the group. When $k = 1$, this sums over the odd elements of the group, multiplying the sum by a factor of $e^{\frac{-2\pi i j}{2^n}}$. This reduces to the familiar equation

$$\hat{f}(j) = \hat{f}_{evens}(j) + e^{\frac{-2\pi i k j}{2^n}} \hat{f}_{odds}(j).$$

So the Cooley-Tukey algorithm is really just a specific case of Diaconis and Rockmore's generalized algorithm! The Cooley-Tukey algorithm's simplicity and ease of implementation is due to the fact that $\mathbb{Z}/2^n\mathbb{Z} \supset \mathbb{Z}/2^{n-1}\mathbb{Z} \supset \cdots \supset \{id\}$ gives an easy chain of subgroups to work with, and that $\mathbb{Z}/2^n\mathbb{Z}$ is abelian.

4.3. **Implementation and Run-Time Analysis.** Let's consider an implementation of this algorithm for the purpose of computing Fourier transforms on $S_n$. This is a useful group to consider, as every finite group of order $n$ is isomorphic to a subgroup to $S_n$ by Cayley's theorem. Additionally, the embedding $S_n \supset S_{n-1} \supset \cdots \supset S_1 = \{id\}$ gives a natural tower of subgroups to use for our algorithm.

Before we delve further into this algorithm, we need a few preliminary facts from the representation theory of $S_n$. Given that the focus of this section is the implementation of this algorithm, these statements will be presented without proof. Those interested in proofs of these statements can refer to work by Diaconis [4, 5].

**Definition 4.5.** An *integer partition* of a natural number $n$ is a set $\{\lambda_1, \dots, \lambda_k\}$ where each $\lambda_i \in \mathbb{N}$, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ and $\lambda_1 + \cdots + \lambda_k = n$.

**Example 4.6.** The number 4 has the following integer partitions: $\{4\}, \{3, 1\}, \{2, 2\}$, $\{2, 1, 1\}$, and $\{1, 1, 1, 1\}$.

**Theorem 4.7.** *The set of integer partitions of $n$ is in bijection with $\hat{S}_n$, the set of irreducible representations of $S_n$ (from* [4]*, Chapter 7, Theorem 1) .*

Through what is known as the Young Orthogonal Representation (YOR), any partition of $n$ can be mapped to an irreducible representation of $S_n$. The YOR is especially useful when considering what happens when a representation is restricted to $S_{n-1}$.

**Theorem 4.8.** *(The Branching Theorem* [4]*) Let $\rho$ be an irreducible representation on $S_n$, represented by the partition $\{\lambda_1, \dots, \lambda_k\}$ of $n$. Let $\rho|_{S_{n-1}}$ be the restriction of $\rho$ to the subgroup $S_{n-1} \subset S_n$, the set of elements stabilizing $n$. We know that $\rho|_{S_{n-1}}$ splits into a direct sum over a collection of irreducible representations. This collection of irreducible representations are the ones represented by partitions that can be created by subtracting 1 from any of the $\lambda_i$ without violating their decreasing property.*

**Example 4.9.** Consider the partition $\{2, 2, 1\}$ of 5, which corresponds to an irreducible representation $\rho$ of $S_5$. Then $\rho|_{S_4}$ is a direct sum of the representations of $S_4$ given by the partitions $\{2, 1, 1\}$ and $\{2, 2\}$. Note that $\{1, 2, 1\}$ would not be a subrepresentation of $\rho_{S_4}$, as this is not a valid partition.

By appealing to these theorems, we now have an easy way to index the irreducible representations of $S_n$, by the partitions of $n$. We can also see how these irreducible representations decompose when restricted to $S_{n-1}$. This gives the basis for an implementation of Diaconis and Rockmore's algorithm for Fourier transforms of functions on $S_n$. Before we fully develop this algorithm, let us consider an implementation of the basic algorithm for computing a Fourier transform on a function on $S_n$. This algorithm has been implemented in Sage, a computer algebra system built on top of Python. It is essentially Python with several useful packages and modules added on.

```
# naive fourier transform on S_n
def slowFT(func, n):
    #constructing S_n
    G = SymmetricGroup(n)
    groupSize = len(G)
    # determining all partitions
    parts = Partitions(n).list()
    numParts = len(parts)
    # array to store result of the F.T
    ftResult = {}
    for i in range(numParts): # iterating through irreducible reps
```

```
        # constructing new list entry to hold running sum
        rep = SymmetricGroupRepresentation(parts[i], "orthogonal")
        # representation of identity element
        idRep = rep(G[0])
        ftResult[str(parts[i])] = idRep
        for j in range(1,groupSize):
            # adding to running sum created above
            ftResult[str(parts[i])] += (func(G[j]))*rep(G[j])
    return ftResult
```

In the code above, the functions *SymmetricGroup*(), *Partitions*(), and *SymmetricGroupRepresentation*() are utilities provided by SageMath. The function iterates through all possible partitions of $n$, creates the corresponding irreducible representation $\rho$ for each one, and then calculates $\hat{f}(\rho)$ by directly computing the sum $\sum_{g\in S_n} f(g)\rho(g)$. The resultant Fourier transform $\hat{f}$, is stored in the dictionary *ftResult*, a set of key-value pairs where each key is a partition, referring to the value $\hat{f}(\rho)$, where $\rho$ is the representation given by the partition. This code is somewhat readable, but is also incredibly slow. The implementation of the fast algorithm outlined by Diaconis and Rockmore is more difficult, but results in impressive speed-up. To implement this recursive algorithm, a few helper functions are necessary.

```
# function to find coset representatives
def getCosetReps(G, n):
    # generate a subgroup using (1,2,...,n)
    H = G.subgroup([nCycle(n)])
    # return its list of elements
    return H.list()
```

The function *getCosetReps*() returns a set of representatives for the cosets of $S_n/S_{n-1}$. To do so, it takes advantage of the following proposition.

**Proposition 4.10.** *Let* $\tau = (1, 2, \ldots, n) \in S_n$. *Then* $\{\tau, \tau^2, \ldots, \tau^n = e\}$ *is a set of coset representatives of* $S_n/S_{n-1}$.

*Proof.* Suppose $\tau^a$ and $\tau^b$ for $1 \leq a, b \leq n$ are such that $\tau^a S_{n-1} = \tau^b S_{n-1}$. This would then imply that $\tau^a\tau^{-b} \in S_{n-1}$. As $\tau$ does not fix the element $n$, it must then be that $a = b$. Therefore the powers of $\tau$ form a set of coset representatives of $S_n/S_{n-1}$. □

The function *getCosetReps*() simply uses the function *nCycle*() to create the element $(1, 2, \ldots, n)$ and makes a list of all powers of this element, which happens to be the subgroup generated by this element. The helper functions given below are also needed.

```
# creates a new function by composing with multiplication
def buildFunc(func, g):
    def newFunc(h):
        return func(g*h)
    return newFunc

# returns partitions indicating subreps of a rep given by a partition
# cannot accept the partition [1]
```

```
def getSubReps(partition):
    newPartList = []
    k = len(partition)
    for i in range(k-1):
        # check if valid subrepresentation
        if partition[i] > partition[i+1]:
            # if so, copy, modify, and append to list
            newPart = partition[:]
            newPart[i] -= 1
            newPartList.append(newPart)
    # the last subrep
    lastPart = partition[:]
    if lastPart[k-1] > 1:
        lastPart[k-1] -= 1
    else:
        # removing last element of partition if it's a 1
        del lastPart[-1]
    newPartList.append(lastPart)
    return newPartList

# function that builds block diagonal restricted transforms
# given a partition and an FT
def makeBlockFT(ft, part):
   # print(ft)
  #  print("calling makeBlockFT, partition = " + str(part))
    # breaking the partition into subreps
    subReps = getSubReps(part)
    # making a list of the matrices of the FT for every given subrep
    subRepList = [ft[str(i)] for i in subReps]

    bdMatrix = block_diagonal_matrix(subRepList)
    return bdMatrix
```

The first of the helper functions above, *buildFunc*(), modifies a given function $f$, returning the function $f_i\colon h \mapsto f(g_i h)$. As discussed previously, the Fourier transforms of these modified functions will be used to construct our final result. The next function, *getSubReps*(), takes a partition giving an irreducible representation, and returns a list of all partitions representing subrepresentations of the restriction of the original representation to $S_{n-1}$. Finally, the function *makeBlockFT*() takes in the Fourier transform of a function on $S_{n-1}$, and a partition giving an irreducible representation of $S_n$. It uses this to create a block diagonal matrix giving the value of the Fourier transform of this representation restricted to $S_{n-1}$. In the words of our initial description of the algorithm, *makeBlockFT*() uses $\hat{f}_i$ and a representation $\rho$ to produce $\hat{f}_i(\rho|_{S_{n-1}})$. Now that we have a sufficient set of helper functions defined, we are ready to implement the algorithm.

```
# actual computation of FT using Diaconis & Rockmore's algorithm
def fastFT(func, n):
   # print("calling computeFT, n = " +  str(n))
    G = SymmetricGroup(n)
```

```
if n == 1:
    # base case, return dictionary just containing function value
    # mapping from trivial rep
    return {"[1]" : matrix([func(G[0])])}
else:
    cosetReps = getCosetReps(G,n)
    # to store result of F.T. on S_n-1
    recursed  = [None]*n
    for i in range(n):
        funcG = buildFunc(func, cosetReps[i])
        # recursive step
        recursed[i] = fastFT(funcG, n-1)
    # make irreps
    ftRestricted = [None]*n
    parts = Partitions(n).list()
    numParts = len(parts)
    for i in range(n):
        # building restricted transforms
        ftRestricted[i] = {}
        for j in range(numParts):
            blockMat = makeBlockFT(recursed[i], parts[j])
            ftRestricted[i][str(parts[j])] = blockMat
    # computing final FT results
    ftResult = {}
    for j in range(numParts):
        # making the rep
        rep = SymmetricGroupRepresentation(parts[j],
                "orthogonal")
        for i in range(n):
            if i==0:
                #initializing the matrix
                start = rep(cosetReps[i])
                start *= ftRestricted[i][str(parts[j])]
                ftResult[str(parts[j])] = start
            else:
                # adding to the running sum
               addOn = rep(cosetReps[i])
               addOn *= ftRestricted[i][str(parts[j])]
               ftResult[str(parts[j])] += addOn
    return ftResult
```

Now that this algorithm has been implemented, how does its run-time compare with the naïve algorithm? Let's begin with determining the run-time of the naïve algorithm on a finite group $G$. For any given irreducible representation $(\rho, V)$, calculating $\hat{f}(\rho)$ in this way requires $O(d_\rho^2|G|)$ operations, where $d_\rho$ denotes the dimension of V. Using Proposition 3.31, calculating the Fourier transform over all irreducible representations requres $O(|G|^2)$ operations for any finite group.

TABLE 1. Comparison of Naïve and Fast Fourier Transform on $S_n$ for Small $n$.

| $n$ | Run-time (in seconds) for $S_n$ SlowFT() | FastFT() |
|---|---|---|
| 1 | 0.0004 | 0.0004 |
| 2 | 0.0020 | 0.0018 |
| 3 | 0.0111 | 0.0110 |
| 4 | 0.1050 | 0.1053 |
| 5 | 4.2183 | 4.1372 |
| 6 | 389.92 | 69.22 |
| 7 | 55102.37 | 1684.99 |
| 8 | — | 5804.63 |

Now to determine the improvement given by our algorithm. The Cooley-Tukey algorithm satisfied the neat recurrence relation $T(n) \leq 2T(\frac{n}{2}) + O(n)$. Unfortunately, since Diaconis and Rockmore's algorithm depends on the choice of subgroup, no such recurrence relation exists in the general case. This specific implementation of the algorithm on $S_n$ satisfies $T(n!) \leq nT((n-1)!) + x$, where $x$ denotes the run-time of reconstructing the restricted transforms on $S_n$ from the transforms on $S_{n-1}$ and summing over the product of these restricted transforms and the $\rho(g_i)$. This depends on efficiency of the matrix multiplication. The naïve matrix multiplication algorithm multiplies two $n \times n$ matrices in $O(n^3)$ time. Recent advances have resulted in algorithms that run in $O(n^{2.38})$ time [2]. The theoretical lower bound of run-time for matrix multiplication is $O(n^2)$, as any matrix multiplication algorithm would at least need to read over all entries in both matrices, of which there are $2n^2$ in total. By calculating the run-time of the finite group algorithm on $S_n$ assuming $O(n^2)$ matrix multiplication, Diaconis and Rockmore found that their algorithm executes on $S_n$ in $O((n!)n^2)$ time. Noting that $\log_2(n!)$ is roughly equal to $n^2$, this means that the algorithm has a run-time of approximately $O(|G|\log_2(|G|))$, like the Cooley-Tukey algorithm.

For experimental data comparing the run-time of the above implementations of the naïve algorithm and the Diaconis and Rockmore algorithm, see the table above. Note that there is no data for $SlowFT()$ execution on $S_8$ as there was simply not enough time for the algorithm to execute before this paper was submitted. The algorithm ran for over 3 days without terminating.

4.4. **Opportunities for Further Improvement and Investigation.** While the implementation of the fast algorithm above provides an impressive speed-up when compared to the basic algorithm, there is still a lot of room for improvement. The algorithm was implemented using Sage, which was chosen for ease of use. Unfortunately, this ease of use comes at the price of efficiency, as such a program would most likely execute more quickly in a language such as C. As far as the author is aware, such an algorithm has never been previously implemented in Sage, though implementations for other algorithms for transforms on the symmetric group have been written in C++ [8] and Julia [11]. Additionally, this implementation is not very memory efficient, as it stores a massive amount of data while performing the computation. This data is in the form of several dictionaries that are needed to store results from the recursive calculations of the Fourier transform.

Future implementations could aim to combat these issues of inefficiencies of time and memory. Finally, as this algorithm is specifically for use on $S_n$ future analysis of other groups could prove very interesting. Additionally, the above algorithm always uses one specific tower of subgroups of $S_n$. Diaconis and Rockmore showed in their original paper that to minimize the number of operations needed to execute the algorithm it is necessary to design the chain of groups $G = G_1 \supset G_2 \supset \cdots \supset G_k = \{id\}$ in such a way that minimizes $\sum_{i=1}^{k-1}[G_i : G_{i+1}]$ ([5], Proposition 2). As a result, it could be productive to explore alternative subgroup chains of $S_n$, to see which chain(s) minimize this sum (the chain used in the implementation above has a sum of $n + (n-1) + \cdots + 2$). Doing so could lead to further improvements in run-time.

## Acknowledgments

I would like to first of all thank my mentor Michael Neaton. Michael provided invaluable support and guidance throughout the research and writing process. Without him, this paper would not have been possible. I would also like to extend my gratitude to Peter May for allowing me to participate in this program, as it has been a wonderful learning experience.

## References

[1] Cooley, J. W., and Tukey, J. W. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation 19*, 90 (1965), 297–301.

[2] Coppersmith, D., and Winograd, S. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation 9*, 3 (1990), 251–280.

[3] DeTurck, D. Bases for $L^2(S^1)$ and Fourier series. `https://www.math.upenn.edu/~deturck/m426/notes03.pdf`, February 2005. Class notes for Math 426 (Partial Differential Equations) at the University of Pennsylvania.

[4] Diaconis, P. *Group Representations in Probability and Statistics.* Institute of Mathematical Statistics, 1988, ch. Representation Theory of the Symmetric Group, pp. 131–140.

[5] Diaconis, P., and Rockmore, D. Efficient computation of the Fourier transform on finite groups. *Journal of the American Mathematical Society 3*, 2 (1990), 297–332.

[6] Dym, H., and McKean, H. *Fourier Series and Integrals.* Academic Press, 1972.

[7] Heideman, M. T., Johnson, D. H., and Burrus, S. Gauss and the history of the fast Fourier transform. *IEEE ASSP Magazine 1*, 4 (1984), 14–21.

[8] Kondor, R. $\mathbb{S}_n$ob: a C++ library for fast Fourier transforms on the symmetric group, 2006. Available at `http://www.cs.columbia.edu/~risi/Snob/`.

[9] Kondor, R., Howard, A., and Jebara, T. Multi-object tracking with representations of the symmetric group. In *Artificial Intelligence and Statistics* (2007), pp. 211–218.

[10] Kun, J. The Fourier series — a primer. `https://jeremykun.com/2012/04/25/the-fourier-series/`, April 2012. Blog post on Math∩Programming.

[11] Plumb, G., Pachauri, D., Kondor, R., and Singh, V. $\mathbb{S}_n$FFT: A Julia toolkit for Fourier analysis on functions over permutations. *Journal of Machine Learning Research 16* (2015), 3469–3473.

[12] Rudin, W. *Real and Complex Analysis*, 3 ed. McGraw-Hill, 1987.

[13] Terras, A. *Fourier Analysis on Finite Groups and Applications.* Cambridge University Press, 1999.