

Vanier College

The Sound of Music:  
Audio Data Analysis Using Fourier Transform

Written by:

Jordan Hoppenheim (1955566)

Tiffany Miller (2065452)

Presented to: Christian Stahn

Linear Algebra II

201-HTK-05

May 20th, 2022

# Table of Contents

<b>1 Introduction and Overview</b>	<b>3</b>
1.1 Complex Waves .....	3
1.2 Fourier Transform .....	4
1.3 Complex Numbers .....	5
1.3.1 Roots of Unity .....	6
1.4 Change of Basis .....	7
1.4.1 Understanding the DFT change of Basis .....	7
<b>2 Equations and Proofs</b>	<b>9</b>
2.1 Interesting Properties of the Transformation Matrix .....	11
<b>3 Complex Sinusoids</b>	<b>15</b>
3.1 Circular Motion of Complex Sinusoids .....	15
3.2 Justifying the Sign of Omega .....	16
3.3 Demonstration of the New Basis .....	19
<b>4 Sample Calculations</b>	<b>23</b>
<b>5 Application to Music</b>	<b>26</b>
5.1 Musical Frequencies .....	25
5.2 The Experiment .....	26
5.2.1 Results and Analysis .....	26
<b>6 Conclusion</b>	<b>33</b>
<b>Appendix A</b>	<b>37</b>
<b>Appendix B</b>	<b>40</b>

## **Abstract**

The purpose of this paper is to exhibit the applicability of the discrete Fourier transform, and the analysis of signals in terms of sinusoidal waveforms. The discrete Fourier transformation is a method of audio data analysis that brings a signal from the time domain to the frequency domain. This paper first guides the reader through an overview of the discrete Fourier transform, providing a brief explanation of correlating topics as well as demonstrating the basics on how to compute this transform using sample hand calculations. Later, the application of the DFT is seen through analyzing an A4 piano note sound file using Python coding language. Python efficiently transforms the piano note from the standard time domain into the frequency domain using the FFT algorithm. In the frequency domain, the Fourier coefficients are found from the peaks in the frequency plot. Using these coefficients, the component harmonic frequencies were individually plotted in the time domain and then combined in order to reconstruct the original sound wave to sound the same as the original A4 piano note. It is concluded that the FFT is a sophisticated and efficient way of analyzing audio data.

# 1 Introduction and Overview

To attain a comprehensive understanding of the Fourier transform audio data analysis, it is essential to have a strong understanding of the discrete Fourier transform and its related topics.

## 1.1 Complex Waves

Complex waves are a combination of multiple simple sine waves. In the case of a note being produced by an instrument, a complex wave is the combination of component harmonic frequencies (all integer multiples of the fundamental frequency) and the amplitude of each component wave. The complex wave created by sound is known as a signal. This can be visually represented with the discrete Fourier transform.

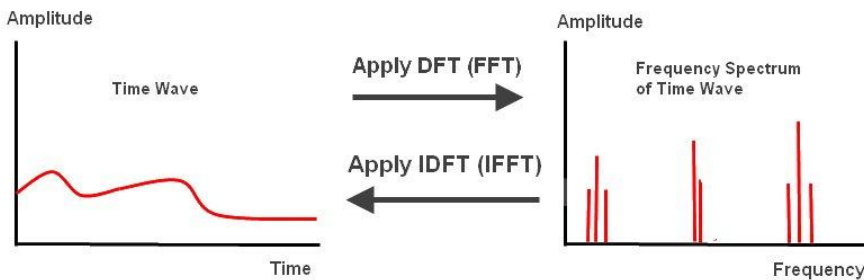
All waves have a frequency, phase and an amplitude. A frequency is the number of times a wave can complete a cycle in a given time (how fast the cycle of a wave is), phase is the wave's position relative to time, and amplitude is the height of the wave's peak from equilibrium.

## 1.2 Fourier Transform

Joseph Fourier, a French mathematician, derived a method of analysis in the early 1800s, allowing a complex wave to be broken up into a series of simple sinusoidal waves (Thummalapalli 2010). This method of analysis can be applied to vectors through the Discrete Fourier Transform (DFT), using discrete time steps. As opposed to continuous time, a discrete-time signal is represented and defined at certain instants of time in its sequence (discrete time). The input to the transform is called a **signal**, existing in the time domain, and the output is called a **spectrum**, existing in the frequency domain. Any sampled analog signal at discrete time points rises to a vector of length  $N$ . This transform breaks a continuous time-based signal into its

constituent component frequencies (Ragel 2011). The multiplicative factor for each component is called a “Fourier coefficient”. This coefficient can be real or complex.

To perform calculations more efficiently, most computer programs use a more sophisticated and fast version of the DFT, called the fast Fourier transform (FFT). Therefore, the fast Fourier transform algorithm can be applied to any sound file and can be deconstructed into its component sinusoidal waves. This can be applied using the Python computer programming language. Through Python, the sample audio file can be read, and the data can be fed into the FFT algorithm to produce a frequency plot. This graph plots each frequency present in the original sound wave relative to how loud it is (amplitude) in an amplitude (in meters) vs. frequency (in Hz) graph.



*Figure 2: Representation of time domain to frequency domain*

The peaks in the frequency plot represent a Fourier coefficient: a component in the Fourier Basis. With that, the amplitude and frequencies of the peaks are then extracted and used to plot the component sine waves. A sinusoid is a function of time having the following form:

$$y = a * \sin(2\pi fn + \phi),$$

where:

a is the amplitude (in meters)

$f$  is the frequency (in Hz or cycles per second)

$n$  is time (in seconds)

$\phi$  is the phase (in radians)

### 1.3 Complex Numbers

Complex numbers are numbers that consist of two parts — a real number and an imaginary number. An imaginary number is a real number multiplied by an imaginary unit  $i$ , in which  $i$  is defined by its property  $i^2 = -1$ . A complex number is described as  $z = a + bi$ , by distance  $r$  from the origin and angle  $\phi$  relative to the positive real axis.  $a$  is the real part and  $b$  is the imaginary part.

The modulus  $r$  is calculated by  $r = \sqrt{a^2 + b^2}$ , and the argument  $\phi$  is found by  $\phi = \tan^{-1}\left(\frac{b}{a}\right)$

Every complex number corresponds to a point on the complex plane. The horizontal axis  $(a,0)$  is the real axis with real numbers, while the vertical axis  $(0,b)$  is the imaginary axis with imaginary numbers.

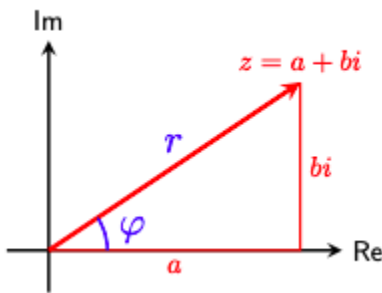


Figure 3: Graphical Representation of a Complex Number

Euler's formula relates the complex exponential to the cosine and sine functions.

$$z = a + bi = r (\cos \varphi + i \sin \varphi) = re^{i\varphi}$$

### 1.3.1 Roots of Unity

Complex roots are represented as a complex number  $z=a+bi$ . The square root of any negative number  $c < 0$  can be expressed as  $\sqrt{|c|} * \sqrt{-1}$ , where  $\sqrt{-1} = i$ .

Every non zero number  $w = re^{i\varphi}$ , has  $n$  distinct  $n$ th roots, that is, solutions to the equation  $z^n = w$ . By letting,  $r=1$  and  $\varphi=0$  in Euler's formula, we get complex numbers, called the  $N$ th roots of unity, the solutions to  $z^n = 1$

$$z_h = (\cos(\frac{2h\pi}{N}) + i \sin(\frac{2h\pi}{N})), \quad h=0, 1, \dots, N-1$$

$N$ th roots of unity are said to be primitive when powers generate all roots of unity. The mapping with the DFT matrix requires  $N$  to have a primitive root.

We define the primitive root of unity with  $h=1$ :

$$e^{2\pi i/N} = \cos(\frac{2\pi}{N}) + i \sin(\frac{2\pi}{N})$$

Later we will see that the  $N$ th roots of unity are used to generate all the sinusoids used by the DFT and its inverse.

## 1.4 Change of Basis

Another way of viewing the discrete Fourier transform is that the DFT is a tool for translating between bases and coordinates. This translation changes the domain (x-axis) of a signal from time to frequency.

A basis is a set of linearly independent vectors that span a vector space. Changing a basis involves taking a vector that is expressed in terms of one basis' coordinates, and converting it into the coordinates of another basis. Since a basis spans the same set of vectors, the input vector's length and direction remains the same, but the coordinates that represent the vector will change. Here, the vector space is  $\mathbb{C}^N$ . There are different bases for  $\mathbb{C}^N$  including the standard basis (time domain) and the frequency basis (frequency domain).

The forward Fourier transform takes a signal vector that is expressed in the standard time basis to the frequency basis. The backwards transform does the reverse and turns a function of frequency into a function of time.

### 1.4.1 Understanding the DFT change of Basis

We will be looking at two bases of  $\mathbb{C}^N$ ; which is the vector space of all signals of length N. First, there is the standard basis. In mathematics, the standard basis of a coordinate vector space is the set of vectors whose components are all equal to zero, except one with a value of 1. The standard basis (time domain), is simply:

$$\vec{e}_0 = [1, 0, 0, \dots, 0]$$

$$\vec{e}_1 = [0, 1, 0, \dots, 0]$$



⋮  
⋮

$$\vec{e}_{N-1} = [0, 0, 0, \dots, 1]$$

With the time domain, a signal  $\{c_0, c_1, \dots, c_{N-1}\}$  can be expressed as a linear combination of these basis vectors,  $c_0 \vec{e}_0 + c_1 \vec{e}_1, \dots, c_{N-1} \vec{e}_{N-1}$ . The coefficients  $c_i$  are the coordinates of the signal in the standard basis.

Next, there is the frequency basis that consists of the frequency signals. The frequency basis is an orthogonal basis, consisting of the columns of  $\mathbf{F}^{-1}$ .

$$\vec{f}_0 = [1, 1, 1, \dots]$$

$$\vec{f}_1 = [1, \bar{\omega}, \bar{\omega}^2, \bar{\omega}^3, \dots, \bar{\omega}^{N-1}]$$

$$\vec{f}_2 = [1, \bar{\omega}^2, \bar{\omega}^4, \bar{\omega}^6, \dots, \bar{\omega}^{N-2}]$$

⋮  
⋮

$$\vec{f}_{N-1} = [1, \bar{\omega}^{N-1}, \bar{\omega}^{N-2}, \bar{\omega}^{N-3}, \dots, \bar{\omega}] \quad \text{where } \bar{\omega} = e^{2\pi i/N}$$

The basis vectors include the root of unity as that is the reduced form for those vectors.

To write any change of basis matrix, the basis vectors are used as columns. That is why we can say the columns of matrix  $\mathbf{F}^{-1}$  form the frequency basis.

Given two bases, the standard basis and the frequency basis, of vector space  $\mathbb{C}^N$ , there are two change-of-basis matrices.  $\mathbf{F}$  is the transformation matrix that converts from the standard

basis to the frequency basis. And  $F^{-1}$  is the transformation matrix that converts a signal from the frequency basis to the standard basis. This will be proven later in the paper.

This works because, to find the change of base, you take the inner product (or dot product) of the vector you want to change with each linearly independent basis vector. Each inner product solution will correspond to the coordinates of your original vector in that new basis, thus delivering your original vector in a new basis.

As we move further in the paper, we will see that the definition of the forward DFT looks like a dot product, seeing that it is the sum of products of the input signal  $f$  and the complex exponential, thus changing the basis of  $f$ .

## 2 Equations and Proofs

The discrete Fourier transform is a matrix transform that takes a vector of data, sampled at discrete time steps, to a vector of Fourier coefficients for that data. Sin and cosines of discrete frequencies that are increasing from 0 to N-1 are used.

The forward DFT is defined as:

$$F_k = \sum_{n=0}^{N-1} f_n e^{-i(2\pi/N)nk} \text{ for any finite signal series } f = [f_0, f_1, \dots, f_{N-1}].$$

Where:

$F_k$  = Signal in the frequency domain

$f_n$  = Signal amplitude at time n (could be real or complex)

n= nth discretized time (0, 1, 2,..., N-1) in seconds (column number of DFT matrix)

k= frequency index (0, 1, 2,..., N-1) (row number of DFT matrix)

N= number of samples (integer)

Since the DFT is a linear transformation, the forward transformation can be expressed using matrix multiplication:  $F_k = F f_n$

$$\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_{N-1} \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \dots & \omega^{N-1} \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{N-2} \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{N-1} & \omega^{N-2} & \omega^{N-3} & \dots & \omega^1 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

Where the DFT NxN matrix:

$$F = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 & \dots & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 & \dots & \omega^{N-1} \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 & \dots & \omega^{N-2} \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 & \dots & \omega^{N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^0 & \omega^{N-1} & \omega^{N-2} & \omega^{N-3} & \dots & \omega^1 \end{bmatrix}$$

The rows and columns of this matrix are labeled from 0 to N-1.

$\omega$  is defined as  $e^{-i(2\pi/N)}$ .

The DFT matrix  $F$  is an example of a Vandermonde matrix, where the exponents of  $\omega$  is the row number multiplied by the column number.  $F$  is a square matrix with orthogonal columns. This matrix is a transformation matrix. So, each entry is  $\omega^{k*n}$ ,  $k$  being the row number and  $n$  being the column number:  $\omega^{k*n} = e^{-2\pi n k i / N} = \cos\left(\frac{-2\pi n k}{N}\right) + i \sin\left(\frac{-2\pi n k}{N}\right)$

It is important to note that:

The entries of the matrix that have  $\omega = e^{-i(2\pi/N)}$  is for the forward discrete Fourier transform, going from  $f_n$  to  $F_k$ .

The entries of the matrix that have  $\overline{\omega} = e^{i(2\pi/N)}$  is for the reverse discrete Fourier transform going from  $F_k$  to  $f_n$ . This will be seen later in the paper.

## 2.1 Some Interesting Properties of the Transformation Matrix $F$

1. The matrix  $F$  is symmetric, therefore  $F^T = F$

Proof

$$(F)_{k*n} = \omega^{k*n} = \omega^{n*k}$$

Since  $k$  is symmetric, we observe that the  $k$ th column of  $F$  is also the  $k$ th DFT sinusoid.

$$2. \overline{F^T} F = N * I_N$$

Proof:

**Part 1:** Represent  $\overline{F^T}$  as  $F^\dagger$

$\text{Det}(F) \neq 0$  since each entry will always be an exponent of  $\overline{\omega}$

From property 1, we know that  $F^T = F$ . So  $\overline{F^T} = \overline{F}$ . Let  $\overline{F^T} = F^\dagger$ . Now we just need to conjugate the matrix, we do this by conjugating each entry. Therefore,

$$F^\dagger = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \overline{\omega} & \overline{\omega}^2 & \dots & \overline{\omega}^{(N-1)} \\ \dots & \overline{\omega}^2 & \overline{\omega}^4 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \overline{\omega}^{(N-1)} & \dots & \overline{\omega}^{(N-1)^2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & e^{i2\pi/N} & e^{i4\pi/N} & \dots & e^{i2\pi(N-1)/N} \\ \dots & e^{i4\pi/N} & e^{i8\pi/N} & \dots & e^{i2\pi(N-1)/N} \\ \dots & \vdots & \vdots & \vdots & \vdots \\ 1 & e^{i2\pi(N-1)/N} & e^{i2\pi(N-1)^2/N} & \dots & e^{i2\pi(N-1)(N-1)/N} \end{bmatrix}$$

Where  $\bar{\omega} = e^{2\pi i/N}$

**Part 2:** Express  $\mathbf{F}^\dagger \mathbf{F}$  as an equation

Each entry of the matrix  $\mathbf{F}^\dagger \mathbf{F}$  is found by the **inner product** of the (k') th row  $\mathbf{F}^\dagger$  with kth column of  $\mathbf{F}$ . The inner vector product is a mathematical operation that takes 2 vectors of equal length and computes a scalar solution. Since matrix  $\mathbf{F}$  has complex entries, the procedure of the inner product for real numbers will not work since  $(i)^2 = -1$ . Therefore, complex inner product on  $\mathbb{C}^N$  is defined by the standard Hermitian inner product:

$$\langle \vec{x}, \vec{y} \rangle = \bar{x}_1 y_1 + \bar{x}_2 y_2 + \dots + \bar{x}_n y_n$$

$$= (\bar{x}_1, \dots, \bar{x}_n) \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

$$= \bar{x}^T y$$

Where vectors  $x$  and  $y \in \mathbb{C}$

*Aside: Complex conjugation*

$z \in \mathbb{C}$ ,  $z = a + bi$  where the conjugate  $\bar{z} = a - bi$

$$\overline{z + w} = \bar{z} + \bar{w}, \text{ where } w \in \mathbb{C}$$

$$\overline{zw} = \bar{z} \bar{w}$$

$$\bar{\bar{a}} = a, \text{ where } a \in \mathbb{R}$$

Thus, the proof can be written as:

$$\vec{u}_k^T * \vec{u}_{k'} = \sum_{n=0}^{N-1} (e^{2\pi i/N})^{kn} (e^{-2\pi i/N})^{k'n} = \sum_{n=0}^{N-1} (\overline{\omega})^{(k-k')n}$$

### Part 3: Proof of Inner Product

Since the columns of  $F$  are orthogonal, the inner product between any column with another column in that matrix is equal to zero. Thus, when  $k' \neq k$ ,  $\langle k', k \rangle = 0$

Let  $B$  represent the sum:

$$B = 1 + x + x^2 + x^3 + \dots + x^{N-1}$$

$$xB = x + x^2 + x^3 + \dots + x^{N-1} + x^N$$

$$B - Bx = (1 + x + x^2 + x^3 + \dots + x^{N-1}) - (x + x^2 + x^3 + x^4 + \dots + x^{N-1} + x^N) = (1 - x^N)$$

$$(1 - x)B = (1 - x^N) \rightarrow B = (1 - x^N)(1 - x)^{-1}$$

$$\sum_{n=0}^{N-1} (\omega)^{(k-k')n} = \frac{(1 - (\omega)^{(k-k')N})}{(1 - (\omega)^{(k-k')})} = \frac{(1 - (e^{i2\pi/N})^{(k-k')N})}{(1 - (e^{i2\pi/N})^{(k-k')})} = \frac{(1 - e^{i2\pi \cdot (k-k')})}{(1 - (e^{i2\pi/N})^{(k-k')})}$$

$$e^{i2\pi \cdot (k-k')} = \cos(2\pi \cdot (k-k')) + i \sin(2\pi \cdot (k-k')) = 1$$

$\rightarrow$  for all  $(k-k') \neq 0$

$$\text{This proves: } \sum_{n=0}^{N-1} (\omega)^{(k-k')n} = \frac{(1 - e^{i2\pi \cdot (k-k')})}{(1 - (e^{i2\pi/N})^{(k-k')})} = 0 \text{ for all } (k-k') \neq 0$$

Furthermore, taking the inner product of a column vector of  $F$  with itself is equal to  $N$ . When

$k' = k$ ,  $\langle k', k \rangle = N$ .  $k' = k$ :  $\overline{(\omega)}^{(k-k')n}$  is always equal to 1

$$\sum_{n=0}^{N-1} (\omega)^{(k-k')n} = 1 + \omega^{(k-k')} + \omega^{(k-k')^2} + \omega^{(k-k')^3} + \dots + \omega^{(k-k')^{N-1}}$$

$k'=k$ :  $\omega^{(k-k')n}$  will always = 1, and this concludes  $\sum_{n=0}^{N-1} \omega^{(k-k')n} = N \cdot 1 = N$

As a product of this proof, the matrix product of

$$F^{\dagger} F = \begin{bmatrix} N & 0 & 0 & \cdots & 0 \\ 0 & N & 0 & \cdots & 0 \\ 0 & 0 & N & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & N \end{bmatrix} = N \cdot I$$

The equation  $F^{\dagger} F = N \cdot I_N$  implies that the columns of  $F$  are orthogonal, which we already knew.

One of the axioms for complex inner products is positive definiteness. For all  $v, \in \mathbb{C}^N$

$\langle v, v \rangle = \|v\|^2 \geq 0$ , and  $\langle v, v \rangle = 0$  if and only if  $v = 0$ .

$\langle v, v \rangle$  gives a positive real number

The positive definite property is intended to capture the idea of length. The length of vector  $v$  is the inner product with itself, and length is nonnegative. The only vector that should have length of zero is the zero vector.

Therefore, now that we know how to compute complex inner product, we can confirm that

$$F^{\dagger} F = N \cdot I_N$$

$$\text{So, } F^{-1} = \frac{1}{N} F^{\dagger}$$

3. We can write the inverse of the forward DFT as  $f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{i(2\pi/N)nk}$

Proof:

$$\mathbf{F}^\dagger \mathbf{F} = N * \mathbf{I}_N$$

$$\frac{1}{N} \mathbf{F}^\dagger \mathbf{F} * f_n = \mathbf{I}_N * f_n ; \text{ for } n \text{ is from } 0 \text{ to } N$$

$$\frac{1}{N} \mathbf{F}^\dagger * F_k = f_n ; \text{ for } n \text{ and } k \text{ are from } 0 \text{ to } N$$

$$f(x_n) = \frac{1}{N} \sum_{k=0}^{N-1} F(x_k) e^{i(2\pi/N)nk}$$

4.  $\mathbf{F}$  is a Unitary matrix

Normalizing  $\mathbf{F}$  makes the transform unitary. The normalized DFT matrix is given by

$$\hat{\mathbf{F}} = \frac{1}{\sqrt{N}} \mathbf{F} \text{ by definition. The corresponding inverse normalized DFT is simply } \overline{\hat{\mathbf{F}}} . \text{ So, } \hat{\mathbf{F}} \overline{\hat{\mathbf{F}}} = \mathbf{I}$$

. This implies that the columns of  $\mathbf{F}$  are orthonormal. We can now call  $\mathbf{F}$  a unitary matrix. A unitary matrix is a complex matrix with orthonormal columns, and whose inverse equals its conjugate transpose. Since  $\mathbf{F}$  is unitary, this means that the columns of that matrix form an orthonormal basis with respect to the inner product determined by  $\mathbf{F}$ . An orthonormal basis of a vector space is a basis whose vectors are mutually orthogonal and normalized.

### 3 Complex Sinusoids

A complex sinusoid is defined in terms of amplitude, frequency and phase. It consists of a cosine component that represents the real part of the complex sinusoid, and a sine component that represents the imaginary part. A complex sinusoid can be visualized as a 3 dimensional helix in space. The 3 dimensions are the real axis, imaginary axis, and time.

#### 3.1 Circular Motion of Complex Sinusoids

Recall Euler's identity  $e^{i\theta} = \cos(\theta) + i\sin(\theta)$ . Multiplying this equation by  $A \geq 0$  and setting  $\theta$  as  $2\pi f * n$  (where  $n$  is time in seconds), we get the definition of the complex sinusoid:



$$y = Ae^{i2\pi fn} = A\cos(2\pi fn) + iA\sin(2\pi fn),$$

A is the modulus of the complex coefficient for all n:  $|A| = \sqrt{a^2 + b^2}$ . Since the modulus of the complex sinusoid is constant, it must lie on a circle in the complex plane. For example,  $\omega = e^{2\pi i/N}$  traces out counterclockwise circular motion along the unit circle in the complex plane (Smith 2002).

Given that the real part of  $e^{i2\pi fn}$  is  $\cos(2\pi fn)$  and the imaginary part is  $\sin(2\pi fn)$ , it can be interpreted that sinusoidal motion is the projection of circular motion into any straight line. Therefore, in the complex plane, the  $\cos(2\pi fn)$  motion is the projection of  $e^{i2\pi fn}$  onto the real axis, and the  $\sin(2\pi fn)$  motion is the projection of  $e^{i2\pi fn}$  onto the imaginary axis. The following figure shows the projections of a complex sinusoid.

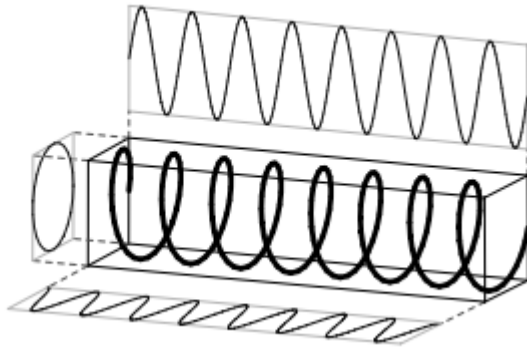


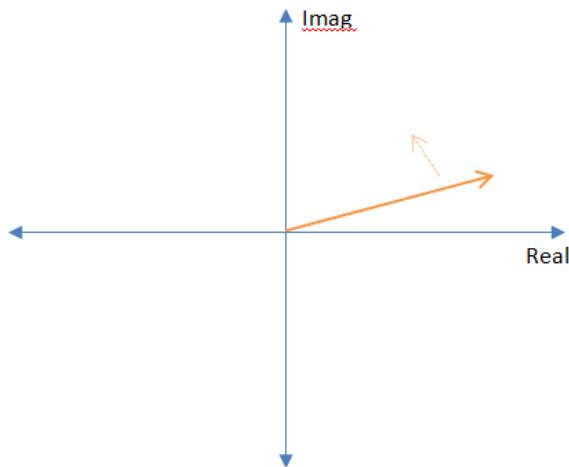
Figure 4: Projections of a complex sinusoid

This figure has 3 dimensions: real axis, imaginary axis, and time. Thus, this 3D plot can be read with two 2D graphs.

### 3.2 Justifying the Sign of Omega

Why does the forward Fourier transform use  $\omega = e^{-2\pi i/N}$ , whereas the reverse Fourier transform uses  $\overline{\omega} = e^{2\pi i/N}$ ?

If we look at complex frequencies in the complex plane, they look like constant vectors rotating. Positive frequencies rotate counter-clockwise, negative frequencies rotate clockwise, and frequencies at 0Hz do not rotate at all.



*Figure 5: Representation of Rotating Frequencies*

The forward Fourier transform complex exponential has a negative exponent to intentionally rotate in the opposite direction as the frequencies that they are ‘looking’ for.

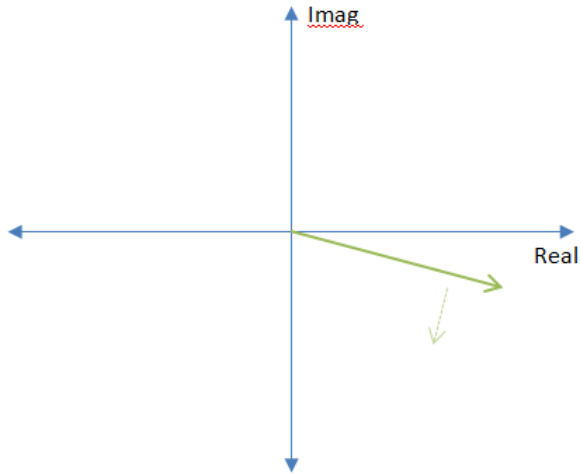


Figure 6: *Intentional Opposing Rotation Done by the Forward DFT*

The reason for the opposite rotation is that when two frequency vectors are multiplied, their phases will repeatedly cancel out, so when results are summed together, there will be a massive

vector due to the individual vectors lining up:  $F_k = \sum_{n=0}^{N-1} f_n e^{-i(2\pi/N)nk}$  (Clay 2014). This is

essentially how the Fourier transform ‘looks’ for frequencies.

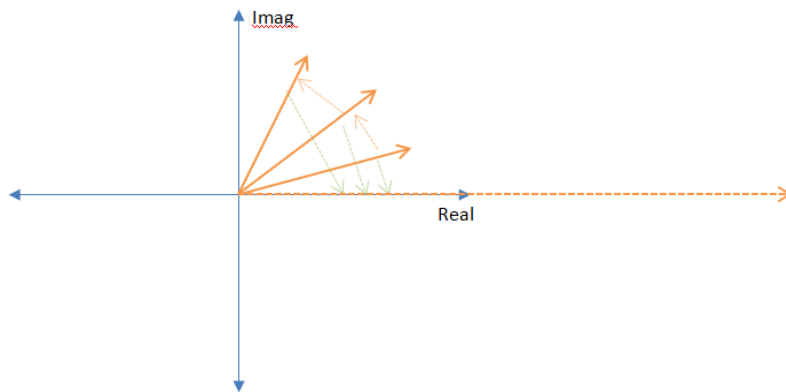


Figure 7: *Demonstration of How Phases Cancel Out*

Therefore, to plot the basis vectors of the frequency domain, we use  $\bar{\omega} = e^{2\pi i/N}$ . With N samples, the basis vectors run from  $\vec{f}_0$  to  $\vec{f}_{N-1}$ .

$\bar{\omega}$  to any power  $k*n$  gives cosines and sines

$$\bar{\omega}^{k*n} = e^{2\pi i k n / N} = \cos\left(\frac{2\pi k n}{N}\right) + i \sin\left(\frac{2\pi k n}{N}\right),$$

Given each entry is  $(F)_{kn} = \bar{\omega}^{k*n}$ , row number k times column number n, the column vector is a vector as a function of row k. Thus, each vector in the frequency basis is a function of k, a complex sinusoid with frequency k. It is also important to note that the column number n determines the number of oscillations.

### 3.3 Demonstration of the New Basis

#### Example:

The following example shows the new basis when N=8. When N=8, the basis vectors include  $\{\vec{f}_0, \vec{f}_1, \vec{f}_2, \vec{f}_3, \vec{f}_4, \vec{f}_5, \vec{f}_6, \vec{f}_7\}$ . Where each vector  $f$  in the basis are column vectors of  $F^{-1}$ . Also, since N=8, the length of each vector is 8, and there will be 8 basis vectors (first 8 columns for  $F^{-1}$ ); a basis of  $\mathbb{C}^8$ .

Basis vector  $\vec{f}_0$  has n=0, therefore we expect 0 oscillations.

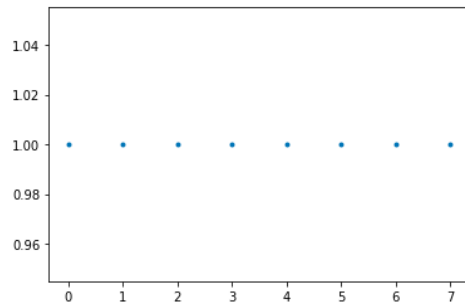
Each entry will be  $\bar{\omega}^{-0*k}$ , where k is the row number (k=0 to N-1 so in this case k=0 to 7)

$$\vec{f}_0 = [\bar{\omega}^{-0*0}, \bar{\omega}^{-0*1}, \bar{\omega}^{-0*2}, \bar{\omega}^{-0*3}, \bar{\omega}^{-0*4}, \bar{\omega}^{-0*5}, \bar{\omega}^{-0*6}, \bar{\omega}^{-0*7}] = [1, 1, 1, 1, 1, 1, 1, 1]$$

$$\bar{\omega}^{-0*k} = e^{2\pi i 0 k / N} = \cos\left(\frac{2\pi 0 k}{N}\right) + i \sin\left(\frac{2\pi 0 k}{N}\right) = 1$$

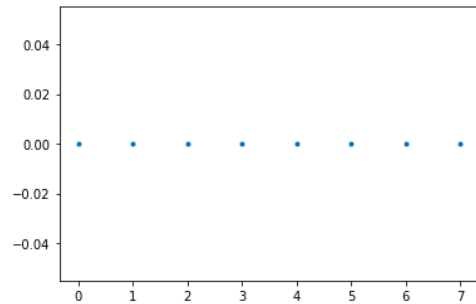
✓ [10] `plt.plot(np.real(col), '.')`

[<matplotlib.lines.Line2D at 0x7f2a89058850>]



✓ `plt.plot(np.imag(col), '.')`

[<matplotlib.lines.Line2D at 0x7f2a88fcc590>]



✓ 0s complete

$\vec{f}_1$  has  $n=1$ , therefore we expect 1 oscillation.

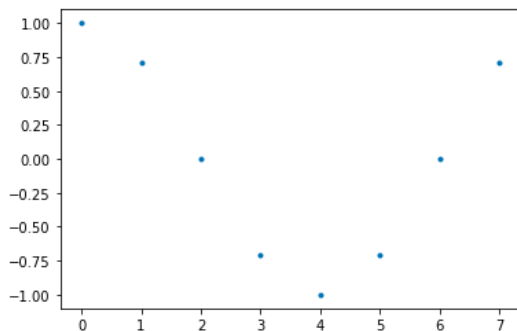
Each entry will be  $\omega^{-1*k}$

$$\vec{f}_1 = [\omega^{-0*1}, \omega^{-1*1}, \omega^{-2*1}, \omega^{-3*1}, \omega^{-4*1}, \omega^{-5*1}, \omega^{-6*1}, \omega^{-7*1}] = [\omega^0, \omega^{-1}, \omega^{-2}, \omega^{-3}, \omega^{-4}, \omega^{-5}, \omega^{-6}, \omega^{-7}]$$

$$\omega^{-1*k} = e^{2\pi k i / N} = \cos\left(\frac{2\pi k}{N}\right) + i \sin\left(\frac{2\pi k}{N}\right)$$

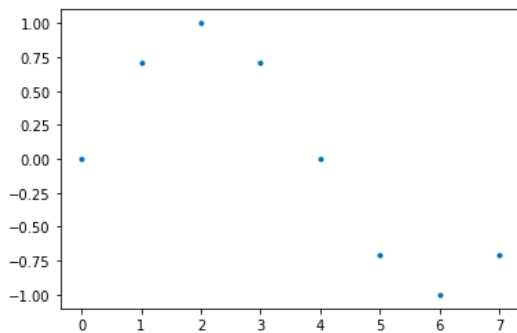
```
✓ [7] plt.plot(np.real(col), '.')
```

[<matplotlib.lines.Line2D at 0x7f2a895d1e50>]



```
✓ ▶ plt.plot(np.imag(col), '.')
```

[<matplotlib.lines.Line2D at 0x7f2a890ca290>]

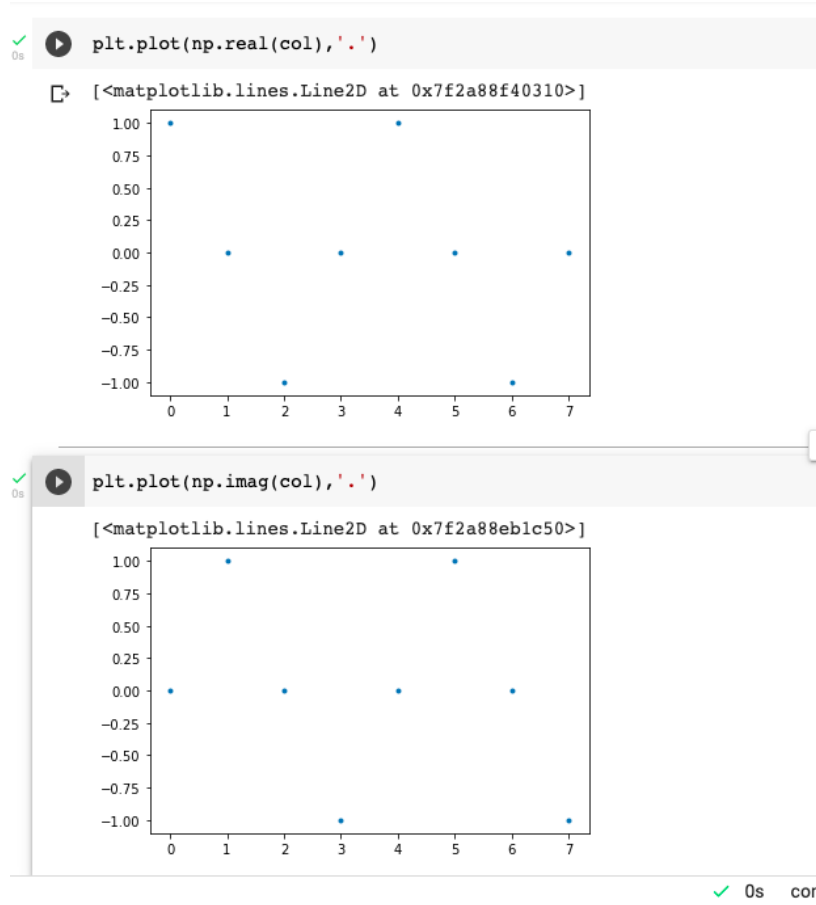


✓ 0s completed at 12:33 AM

$\vec{f}_2$  has  $n=2$ , therefore we expect 2 oscillations.

Each entry in the column vector will be  $\omega^{-2*k} = e^{2*2\pi k i/N} = \cos(\frac{4\pi k}{N}) + i\sin(\frac{4\pi k}{N})$

Column 2 =  $[\omega^{-2*0}, \omega^{-2*1}, \omega^{-2*2}, \omega^{-2*3}, \omega^{-2*4}, \omega^{-2*5}, \omega^{-2*6}, \omega^{-2*7}] = [\omega^0, \omega^{-2}, \omega^{-4}, \omega^{-6}, \omega^{-8}, \omega^{-10}, \omega^{-12}, \omega^{-14}]$



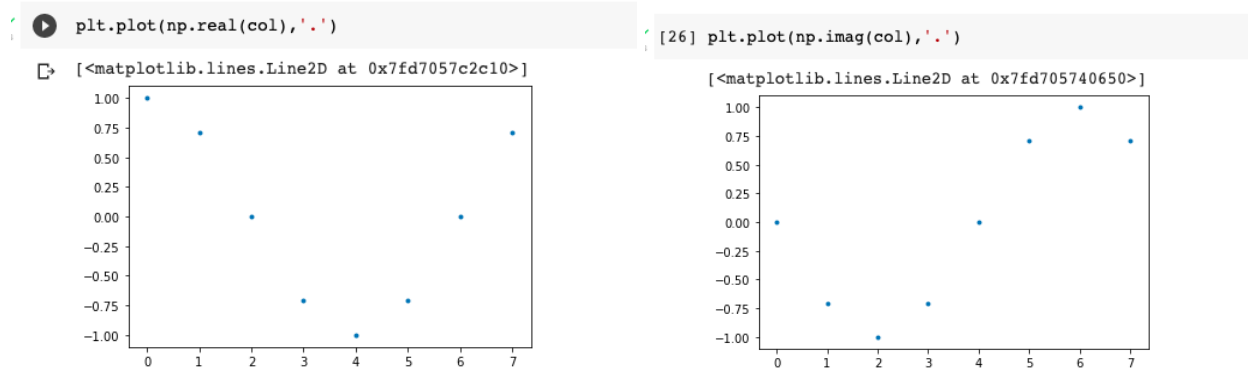
The plot of  $\vec{f}_3$  to  $\vec{f}_7$  can be seen in the link:

<https://colab.research.google.com/drive/1IgfICtc2oMpY2nvn-b6FAnfrXm9HaYVP?usp=sharing>

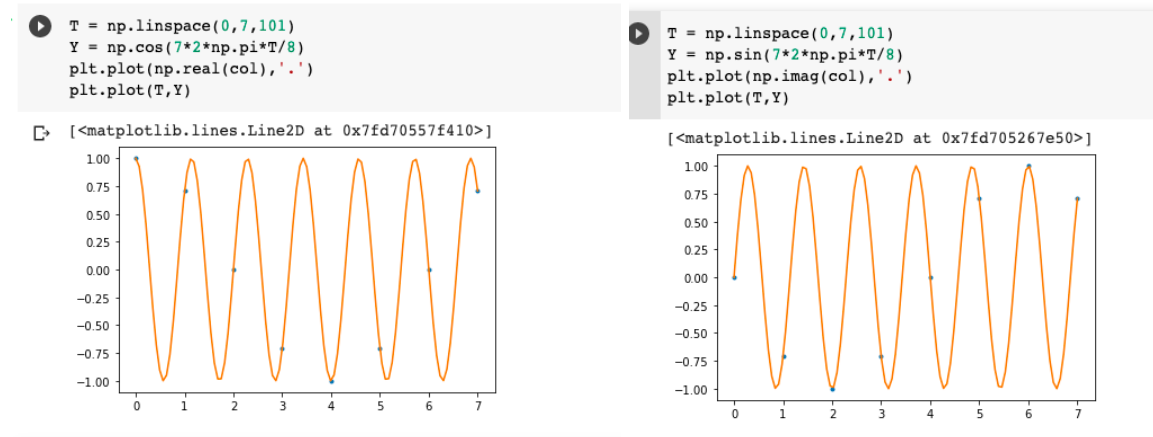
These graphs are to understand the meaning of these basis vectors. They represent harmonic oscillations with frequency  $k$ . Any signal can be expressed in this basis.

*Aside:*

An important note with the  $N=8$  example is that we can observe the phenomenon that if you sample a signal at discrete time intervals, you cannot ‘see’ frequencies beyond a certain cut-off. For example, in the Python basis plots, frequency 6 oscillation looks like 2 oscillations and frequency 7 oscillation looks like 1 oscillation. Looking at the  $\vec{f}_7$  plot, at first glance, we notice only one oscillation:



But, when applying the cosine and sine waves respectively to the real and imaginary part, we see that frequency 7 does in fact have 7 oscillations.



Therefore, if you are sampling at discrete time intervals, the highest frequency you can distinguish is not  $N$ , but  $N/2$ . This also explains to some extent why one only looks at the first  $N/2$  Fourier coefficients.

## 4 Sample Calculations

The following is an example of the discrete fourier transform, how to convert a given time signal to frequency. This can be shown with a 4x4 case.



ex:  $N=4 \Rightarrow 4$  pt DFT

$$F_k = \sum_{n=0}^{N-1} f_n e^{-i(2\pi nk/N)}$$

$$= \sum_{n=0}^3 f_n e^{-i(2\pi nk/4)}$$

$$= \sum_{n=0}^3 f_n e^{-i\pi nk/2}$$

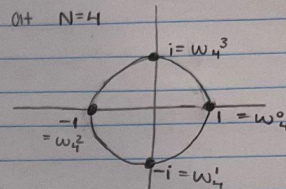
4x4 matrix:

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 \\ \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}$$

where each entry is  $\omega^{k \cdot n}$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

\*we know the entries from the unitary circle



as an example, let  $f_n = \{1, 2, 3, 4\}$

$$f_k = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1+2+3+4 \\ 1-2i-3+4i \\ 1-2+3-4 \\ 1+2i-3-4i \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ -2+2i \\ -2 \\ -2-2i \end{bmatrix}$$

Hilroy

```
[ ] trans = np.fft.fft(np.array([1,2,3,4]))
trans

array([10.+0.j, -2.+2.j, -2.+0.j, -2.-2.j])
```

This hand calculation matches the python file, confirming that fft is  $(F^* \text{ signal})$

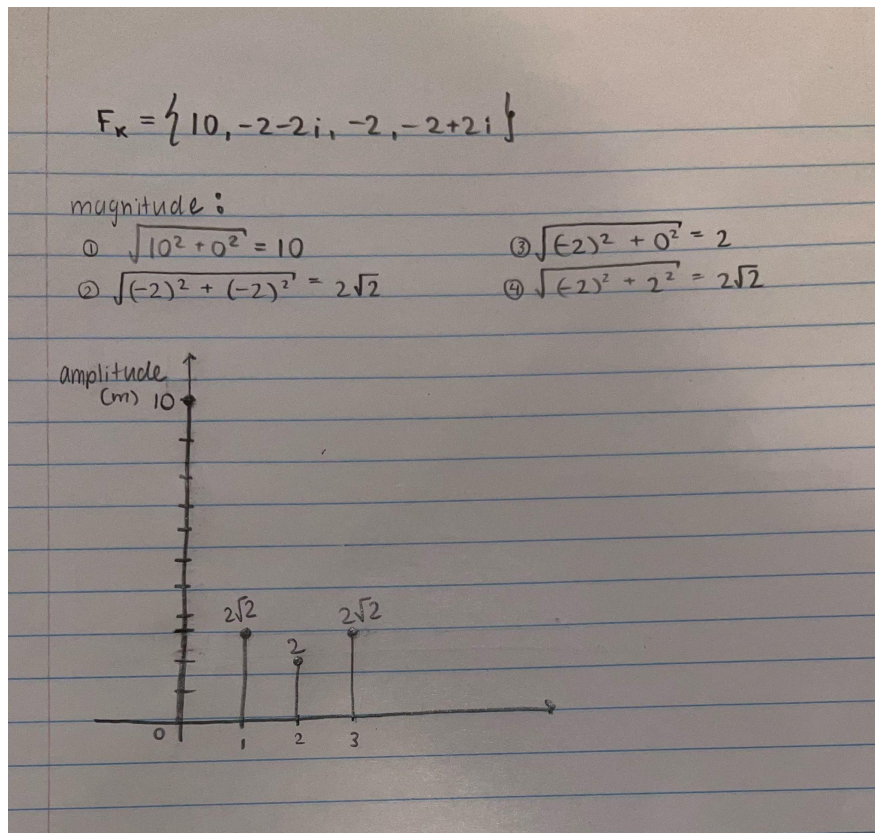
Aside:

Applying ifft in python, the original signal is return, thus implying that in python, ifft is ( $F^{-1}$  \*signal)

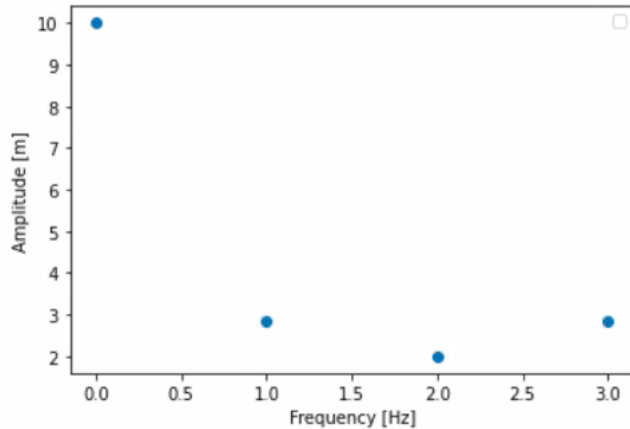
```
[ ] np.fft.ifft(trans)

array([1.+0.j, 2.+0.j, 3.+0.j, 4.+0.j])
```

Back to the example, now that we have the complex Fourier coefficients of signal {1, 2, 3, 4}, we can plot the coefficient on a frequency vs. amplitude graph. This is done by taking the modulus of each coefficient. Taking the modulus as it shows how strong it is at each frequency.



Here the peaks that we see correspond to the frequencies contained in the original signal. Each peak represents a Fourier coefficient: a component in the Fourier Basis. It is verified with python:



## 5 Application to Music

### 5.1 Musical Frequencies

Frequency is the number of times that a wave can complete one full cycle in a time span of a second (Crowell, 15). Objects and mechanical systems vibrate at specific frequencies called resonant frequencies. A resonant frequency is the oscillation of a system at its natural or unforced resonance. Resonance is when one object vibrating at the same natural frequency of the second object forces that second object into vibrational motion (the Physics Classroom). The matching vibrations of the other object increases the amplitude of the object's oscillations. Most vibrating objects have more than one resonant frequency and those used in musical instruments typically vibrate at harmonics (integer multiples) of the fundamental (lowest resonant frequency of the vibrating object). Resonant frequencies are also musical pitches (how high and low sounds are). The higher the frequency, the higher the pitch; and the lower the frequency, the

lower the pitch. Pitch is how the human ear hears and understands frequency. So, a note sounds ‘higher’ or ‘lower’ according to its frequency (Chase 2022).

Pitch frequencies in equal temperament are based on A4=440Hz to the nearest Hertz (middle C=C4). Equal temperament is where an octave is divided into 12 evenly spaced notes.

Each octave refers to the interval between one frequency and its double or its half. For example, if octave 2 has frequency X, octave 3 will have frequency 2X, and octave 1 will have frequency  $\frac{1}{2}X$ . Thus, from any frequency basis, the frequency of a note can be determined by:  $f_n = f_0 * 2^{(n/12)}$  where n is the number of half-steps between  $f_0$  and  $f_n$ .

Sound is created when an object vibrates. Sound waves are transferred from the vibrating object to the eardrum to hear the sound. Knowing that A4=440 Hz, with A4 on a piano having the same frequency as A4 on the guitar, what is the reason for these instruments sounding different? Overtone and harmonics are the answer to the question. When a frequency is played on an instrument, other frequencies, called harmonics, are created. Each instrument has a unique harmonic character. Although all the instruments are playing the same note, it is their component frequencies that give it their unique sound. The fundamental frequency of a note has component resonant frequencies, where the resonant frequencies depend on the shape of the object.

## 5.2 The Experiment

For this experiment, a piano playing the note A4 was recorded. A mono sound file was used. The most common sample rate to record a sound is 44100 samples per second, so in python, that is what was used. Using N= 44100 samples/sec python creates a NxN, 44100x44100, matrix. Using the FFT command in python, the sound file was converted into the

frequency domain. Next, the 5 highest peaks were plotted in a time domain graph (amplitude vs. time graph). Then, we combine all the component waves to reconstruct the original piano sound.

*Aside:*

It is important to note that in the context of the project, with  $y = a * \sin(2\pi fn + \phi)$ , the phase ( $\phi$ ) is in fact changing in the frequency domain. Though, we plot the absolute value of the Fourier coefficients thus killing the phase.

### 5.2.1 Results and Analysis

Python link:

<https://colab.research.google.com/drive/1aXHxvy00I3OzWUz7qYytXJ3E9JM7FgY1?usp=sharing>

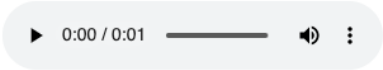
1. First, a sound file “piano.wav” was imported into python. This is a 1 second sound of the A4 piano note at 440Hz.

The following assumes that a sound file "piano.wav" is saved on the Google Drive, in the "Colab Notebooks" directory.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

Mounted at /content/drive

[ ] wav_piano = "/content/drive/My Drive/piano.wav"
    Audio(wav_piano)
```



2. The next code verifies that there is a 4410 sample rate, and that the sound file is in fact 1 second long.

```

] samplerate, data = wavfile.read(wav_piano)
print(f"sample rate = {samplerate}")
print(f"number of channels = 1")
n_datapoints = data.shape[0]
print(f"{n_datapoints} data points")
length_sec = n_datapoints / samplerate
print(f"length = {length_sec:.2f} seconds".format(length_sec))

sample rate = 44100
number of channels = 1
46981 data points
length = 1.065 seconds
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: WavFileWarning: Chunk (non-data) not understood, skipping it.
"""Entry point for launching an IPython kernel.

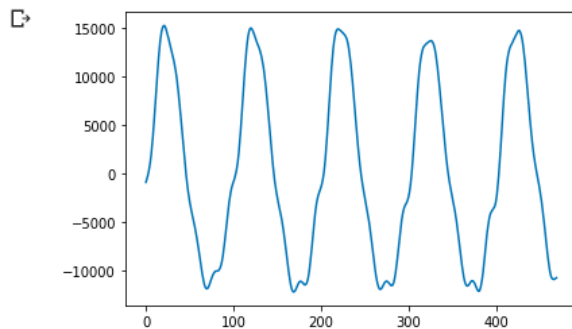
```

- Next we plotted an amplitude (y-axis) vs. time (x-axis) graph. This is the signal in the time domain.

```

plt.plot(data[int(0.2*n_datapoints):int(0.21*n_datapoints)], label="Left channel")
plt.show()

```



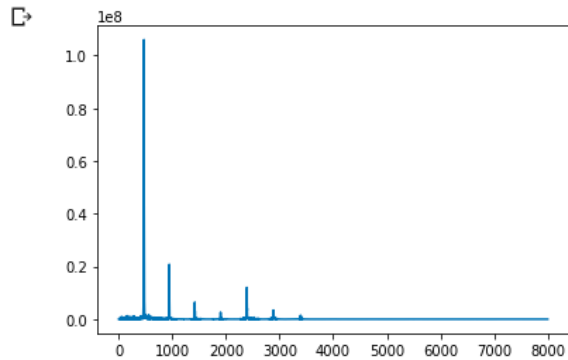
- The FFT command in python, efficiently converts this time based signal into the frequency domain. Here, the graph is a frequency plot with amplitude on the y-axis and frequency on the x-axis.

```

y = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
ax.plot(np.abs(y[:N//2]))
plt.show()

```



5. The highest 5 peaks were plotted in their respective, separate graph. Each graph is magnified (zoomed in) to accurately see the frequency peak with its corresponding amplitude.

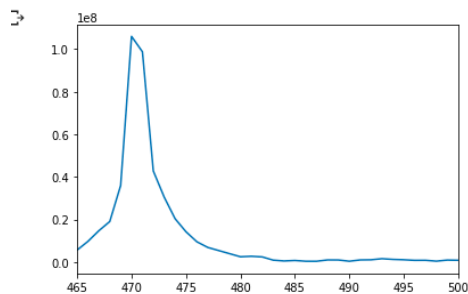
$f_1$ :

```

y = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
plt.xlim([465, 500])
ax.plot(np.abs(y[:N//2]))
plt.show()

```

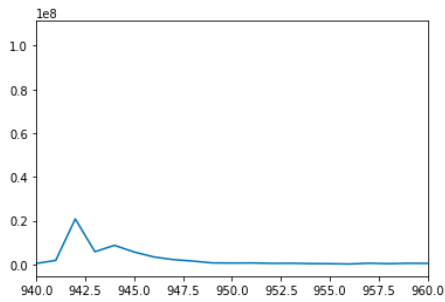


In  $f_1$ , the peak corresponds to 470 Hz at 1 m.

$f_2$ :

```
] yf = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
plt.xlim([940, 960])
ax.plot(np.abs(yf[:N//2]))
plt.show()
```

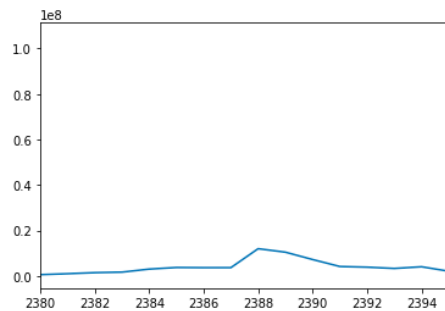


In  $f_2$ , the peak corresponds to 942 Hz at 0.2 m.

$f_3$ :

```
] yf = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
plt.xlim([2380, 2395])
ax.plot(np.abs(yf[:N//2]))
plt.show()
```



In  $f_3$ , the peak corresponds to 2388 Hz at 0.17 m.



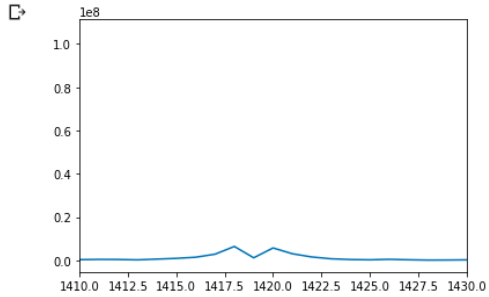
$f_4$ :

```

y = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
plt.xlim([1410, 1430])
ax.plot(np.abs(y[:N//2]))
plt.show()

```



In  $f_4$ , the peak corresponds to 1417.8 Hz at 0.10 m.

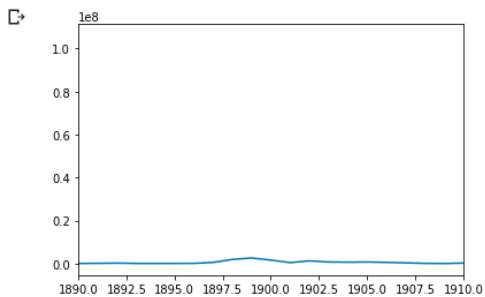
$f_5$ :

```

y = np.fft.fft(data[:])
N = signal.shape[0]

fig, ax = plt.subplots()
plt.xlim([1890, 1910])
ax.plot(np.abs(y[:N//2]))
plt.show()

```



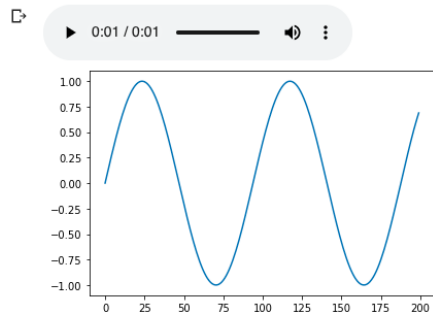
In  $f_5$ , the peak corresponds to 1898 Hz at 0.05 m.

4. Next, each component frequency was plotted separately in the time domain. The amplitude corresponding to the frequency peak is the **Fourier Coefficient** of each component. For

example, at the frequency peak of signal 3, its amplitude was 0.17m. Therefore, signal 3=  
 $0.17 * \sin(2 * \pi * f3 * \text{times})$ ”.

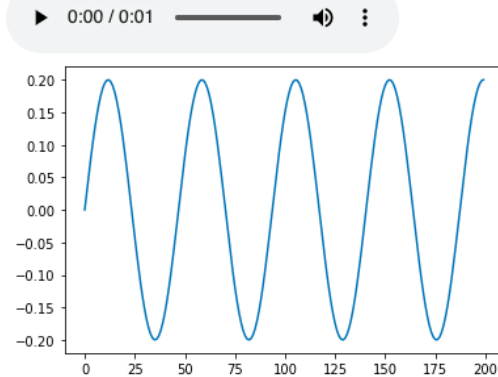
```
import numpy as np
f1 = 470.0          # in Hertz
f2 = 942.0
f3 = 2388.0
f4 = 1417.8
f5 = 1898.0        # in Hertz
sample_rate = 44100 # in Hertz
duration = 1        # in seconds
times = np.linspace(0, duration, duration*sample_rate)
signal1 = np.sin(2*np.pi*f1*times)

fig, ax = plt.subplots()
ax.plot(signal1[:200])
Audio(data=signal1, rate=sample_rate)
```

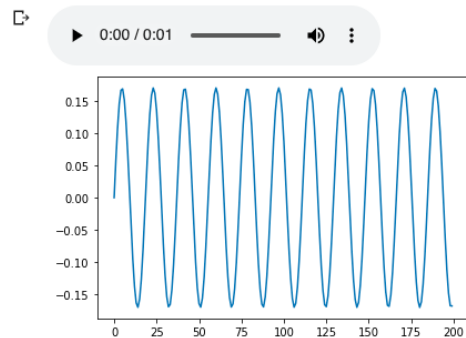


```
signal2 = 0.2*np.sin(2*np.pi*f2*times)
fig, ax = plt.subplots()
ax.plot(signal2[:200])
Audio(data=signal2, rate=sample_rate)
```

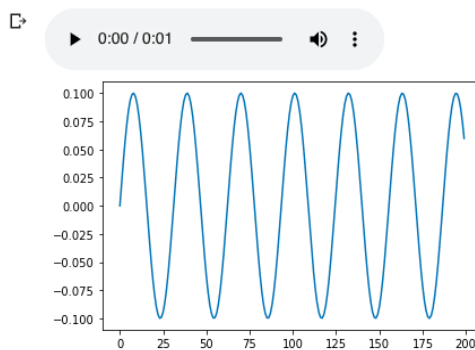
3

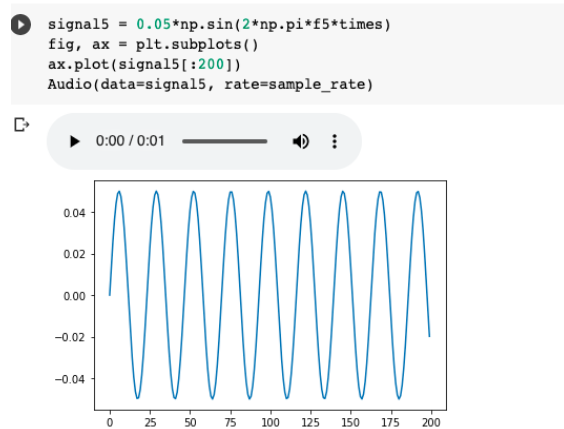


```
signal3 = 0.17*np.sin(2*np.pi*f3*times)
fig, ax = plt.subplots()
ax.plot(signal3[:200])
Audio(data=signal3, rate=sample_rate)
```



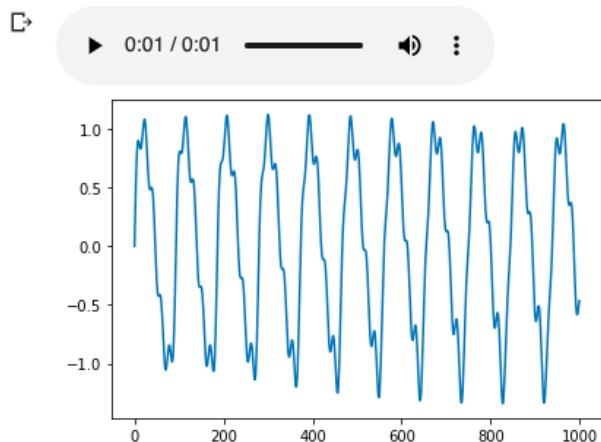
```
signal4 = 0.1*np.sin(2*np.pi*f4*times)
fig, ax = plt.subplots()
ax.plot(signal4[:200])
Audio(data=signal2, rate=sample_rate)
```





5. Finally, these component frequencies in the time domain were combined to make a **complex wave**, returning the original sound of the A4 piano note.

```
signal = signal1 + signal2 + signal3 + signal4 + signal5
fig, ax = plt.subplots()
ax.plot(signal[:1000])
Audio(data=signal, rate=sample_rate)
```



\*If you play this reconstructed note that was made in python, it sounds very similar to the original piano A4 note.

#### *Aside: Conversion*

In the time domain, the x-axis is time and the y-axis is amplitude. In the frequency domain, the x-axis is frequency (in Hz), running from 0 to N-1, and the y-axis is amplitude. The

units of frequency are in oscillations per second.  $\frac{1 \text{ oscillation}}{1 \text{ second}} = 1\text{Hz}$ . This conversion factor is crucial to convert our frequency peaks to Hertz. For example, if we have 20 oscillations in the graph, but the sample was taken over 5 seconds, the following conversion must be computed to convert to Hz:

$$\frac{20 \text{ oscillations}}{5 \text{ seconds}} = \frac{x \text{ oscillations}}{1 \text{ second}} \rightarrow x = 4 \text{ oscillations}$$

Now we can write our frequency in Hz : Frequency =4Hz.

In this case, since our signal was recorded at 1 second, a conversion is not needed.

## 6 Conclusion

The discrete Fourier transformation is a method of audio data analysis that brings a signal from the time domain to the frequency domain. A signal is inputted in the standard basis and is returned in the frequency basis, where each vector in the frequency basis is a complex sinusoid with frequency  $k$  and  $n$  number of oscillations. The DFT matrix  $F$  is a change-of-basis matrix that converts from the standard basis to the frequency basis, and  $F^{-1}$  is the change-of-basis matrix that converts a signal from the frequency basis to the standard basis. Applying the discrete Fourier transform to instruments, knowing that every instrument sounds different due to their component harmonic frequencies, a piano note at A4 was analyzed. After importing the piano sound file into Python, an amplitude vs. time graph was plotted to demonstrate the signal in the time domain. Then, using Python's FFT algorithm, a frequency plot, amplitude vs. frequency graph, was plotted. Since the peaks of the graph demonstrate the Fourier coefficients, component harmonic frequencies were individually plotted in the time domain and then combined in order to reconstruct the original sound wave to sound the same as the original A4 piano note. Hence, the fast Fourier transform is an extraordinary and efficient way of analyzing audio data.

## References

1. Acoustic phonetics: Combining waves. (n.d.). Retrieved April 9, 2022, from <https://home.cc.umanitoba.ca/~krussll/phonetics/acoustic/combining-waves.html>
2. Chaudhary, Kartik. "Understanding Audio Data, Fourier Transform, FFT, Spectrogram and Speech Recognition." *Medium*, Towards Data Science, 4 June 2021, <https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520>.
3. *CIS391*, <https://www.seas.upenn.edu/~cis391/>.
4. "Complex Roots - Definition, Formula, Application, Examples." *Cuemath*, <https://www.cuemath.com/numbers/complex-roots/>.
5. "DFT Using Matrix Method (Problems) - Discrete Fourier Transform - Discrete Time Signal Processing." *YouTube*, YouTube, 4 Nov. 2016, <https://www.youtube.com/watch?v=PLTpmCmBd60>.
6. "Digital Audio Basics: Audio Sample Rate and Bit Depth." *IZotope*, IZotope, Inc., 5 May 2020, <https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html>.
7. ECS, Inc. International. "What Is Resonant Frequency?" *ECS Inc. International*, <https://ecsxal.com/news-resources/video-learning/126-everything-you-need-to-know-about-quartz-crystal-resonators/129-understanding-resonant-frequency#:~:text=Resonant%20frequency%20is%20the%20oscillation,find%20with%20a%20simple%20pendulum>.
8. "Fourier Transform and Frequency Domain HTTP Graphics CMU." *SlideToDoc.com*, <https://slidetodoc.com/fourier-transform-and-frequency-domain-http-graphics-cmu/>.
9. "Frequency, Cycle, Wavelength, Amplitude and Phase: Aruba Blogs." *Blogs.arubanetworks.com*, 9 Feb. 2021, <https://blogs.arubanetworks.com/industries/frequency-cycle-wavelength-amplitude-and-phase/#:~:text=The%20phase%20involves%20the%20relationship,said%20to%20be%20in%20phase>.
10. justinjustin 23922 silver badges88 bronze badges, et al. "Why Is a Negative Exponent Present in Fourier and Laplace Transform?" *Signal Processing Stack Exchange*, 1 July 1962, <https://dsp.stackexchange.com/questions/19004/why-is-a-negative-exponent-present-in-fourier-and-laplace-transform>.
11. Khillar, Sagar. "Difference between FFT and DFT." *Difference Between Similar Terms and Objects*, 29 Sept. 2021, <http://www.differencebetween.net/technology/difference-between-fft-and-dft/>.
12. Lay, David C. *Linear algebra and its applications* / David C. Lay, University of Maryland, College Park, Steven R. Lay, Lee University, Judi J. McDonald, Washington State University. – Fifth edition.

13. “Matrix Formulation of the DFT: Mathematics of the DFT.” *DSP*,  
[https://www.dsprelated.com/freebooks/mdft/Matrix\\_Formulation\\_DFT.html](https://www.dsprelated.com/freebooks/mdft/Matrix_Formulation_DFT.html).
14. “Physics Tutorial: Resonance.” *The Physics Classroom*,  
<https://www.physicsclassroom.com/class/sound/Lesson-5/Resonance>.
15. *Real and Complex Inner Products - Math.columbia.edu*.  
<https://www.math.columbia.edu/~rf/innerprods.pdf>.
16. Rongeegee, et al. “What Exactly Is a Basis in Linear Algebra?” *Mathematics Stack Exchange*, 1 Dec. 1964,  
<https://math.stackexchange.com/questions/2195513/what-exactly-is-a-basis-in-linear-algebra>.
17. Samuel Chase has been playing music since he was 5 years old. “What Is Pitch in Music?: Hellomusictheory.” *Hello Music Theory: Learn Music Theory Online*, 11 May 2022,  
<https://hellomusictheory.com/learn/pitch/>.
18. *Spring 2014 Braithwaite Inner Products - Ups*.  
<http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-braithwaite-inner-products.pdf>.
19. Stahn, Christian. “Linear Algebra 2”, Winter 2022, Vanier college, Montreal.
20. “Text Technologies for Data Science Course Homepage.” *Course Homepage*,
21. Sundararajan, D. *The Discrete Fourier Transform: Theory, Algorithms and Applications*. World Scientific, 2001.
22. <https://www.inf.ed.ac.uk/teaching/courses/tts/>.
23. “Unitary Matrix.” *Unitary Matrix - an Overview | ScienceDirect Topics*,  
<https://www.sciencedirect.com/topics/mathematics/unitary-matrix>.
24. *University of Florida*. <https://ufdcimages.uflib.ufl.edu/AA/00/01/17/30/00001/Vibrations.pdf>.

## Appendix A

### Preliminary Example: Python Code

From Ipython.display import Audio

%matplotlib inline

Import numpy as np

Import matplotlib.pyplot as plt

### **Frequency Basis Vectors**

N=8

omega= np.exp (2\*np.pi\*1j/N)

col= [omega\*\*0\*k) for k in range (N)]

Col

plt.plot(np.real(col), ‘.’)

plt.plot(np.imag(col), ‘.’)

T=np.linspace (0,7,101)

Y= np.cos (2\*np.pi\*T/8)

plt.plot (np.real (col), ‘.’)

plt.plot(T,Y)

### **FFT Experiment**

f1= 470hz

f2= 942hz

.

.

sample\_rate= 44100

duration=1

time= np.linspace (0,duration,duration\*sample\_rate)

```
signal1= np.sin (2*np. pi*f1*times)
```

```
fig,ax=plt.subplots ()
```

```
Ax.plot (signal1[:200])
```

```
Audio (data=signal1, rate= sample_rate)
```

### **Importing sound files**

```
From scipy.io import wavfile
```

```
From google.colab import drive
```

```
Drive.mount ('/content/drive')
```

```
wav_piano= "/content/drive/My drive/piano.wav"
```

```
Audio (wav_piano)
```

```
sample,data=wavfile.read(wav_filename)
```

```
Print (f"number of channels= {data.shape [1]}")
```

```
n_datapoints= data.shape [0]
```

```
Print (f"{n_datapoints} data points")
```

```
length_sec= n_datapoints/ samplerate
```

```
Print ("length= {:.3f}. Format (length_sec))
```

```
time= np.linspace (0.,length_sec, data.shape [0])
```

```
Plt.plot (time, data[:,0], label= "Left channel")
```

```
Plt.plot (time, data[:,1], label= "right channel")
```



```
Plt.legend ()  
Plt.xlabel ("Time [s]")  
Plt.ylabel ("Amplitude [m]")  
plt.show
```

### **Frequency Plot**

```
yf= np.fft.fft(data[:])  
  
N=signal.shape [0]  
  
fig, ax=plt.subplots ()  
  
Ax.plot (np.abs (yf[:N//2]))  
  
plt.show()
```

```
yf= np.fft.fft(data[:])  
  
N=signal.shape [0]  
  
fig, ax=plt.subplots ()  
  
Plt.xlim ([465,500])  
  
Ax.plot (np.abs (yf[:N//2]))  
  
plt.show()
```

## Appendix B

[Frequency basis vector Python link](#)

[FFT experiment Python link](#)