Fourier Analysis:

A Mathematical Investigation on Musical Pitch

Candidate Number: hlm215

Table of Contents:

Introduction

The digitisation of signals from the real world into bits which can be interpreted by computers, has increasing prevalence as new technology develops (Royakkers et al., 2018). Digitisation has a crucial application in many areas, such as data processing in engineering, applied mathematics, physics and chemistry (University of California Berkeley Department of Astronomy, n.d.). Fourier Analysis is a tool used to process these sinusoidal time-domain signals in the frequency domain, which presents useful information about the frequency components of the signal.

My internal assessment explores this notion with a musical application, aiming to determine the pitch of recorded piano notes through analysing the signal using Fourier Analysis. Three recordings were created: the first one of one note, the second containing two notes played simultaneously, and the third with three notes played together. The investigation stems from my rare auditory ability of having absolute pitch, where I am able to determine and recreate the pitch of any note without any reference pitch, as well as my musical interests as a piano player. Electronic tuners are usually used to identify the pitch of a note, indicating how far away the pitch of the note is from the nearest set pitch. As a flute player, I discovered a major problem with these tuners, in which other sounds interfere with the tuning process, especially in an orchestra setting where the flute is generally outclassed in volume by other instruments playing different notes simultaneously. As a result, rapid fluctuations in the pitch are picked up and displayed by the tuner due to the multitude of sounds playing at the same time.

Fundamentally, sound can be expressed as a sinusoidal wave in the time-domain (*Figure 1*). The principle of superposition of waves states that the resultant wave when two waves meet is the sum of the individual waves at every point (see *Figure 2*) (Boston University Department of Physics, n.d.). Reversing this logic, any wave, such as a sound wave, can also be expressed as a combination of smaller constituent sinusoidal waves. Similar to how my tuner picked up sounds from multiple sources as a singular wave, a piano note is comprised of several smaller constituent waves as well as the fundamental frequency which defines and distinguishes different pitches from each other. Different instruments playing the same pitch or note produce different tone colours or sounds due to differences in these smaller waves, known as the instrument's harmonics (Teach Me Audio, 2019, para. 9).

$$\textbf{Amplitude} = \mathbf{f(t)} = \mathbf{sin(t)},$$

where t = time, t: [0,4π] and f(t): [-1,1]

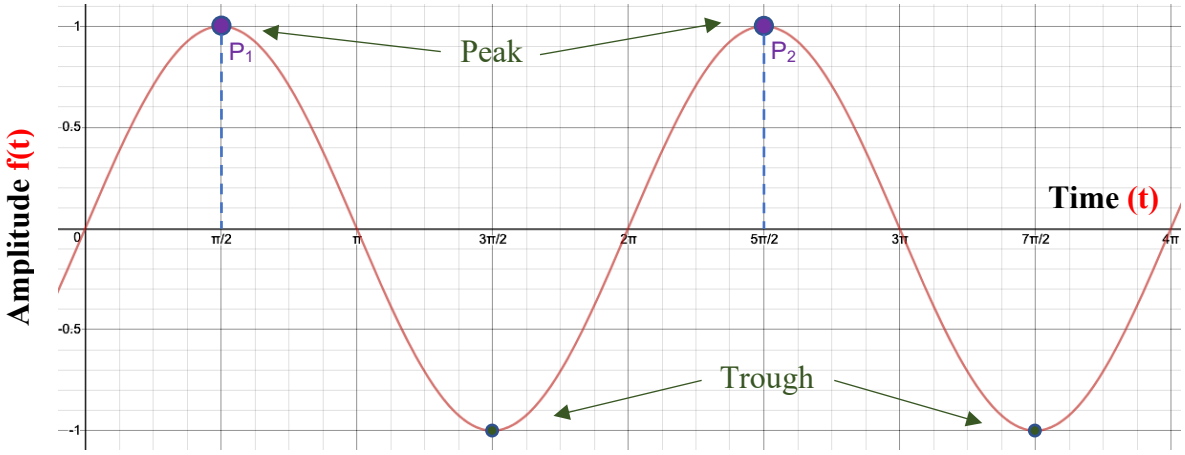


Figure 1: Graph of Amplitude = sin (t)

A sine graph has the equation:

$$y = A\sin(B(x - D)) + C \qquad \text{(Eq. 1)}$$

The following information can be determined from *Equation 1*:

- **Amplitude** (A): maximum displacement from an equilibrium position
  *This is determined by the **A** value, which is 1 in Figure 1.*
- **Time period** (T): time required for one complete cycle; time difference between two successive peaks, or two successive troughs
  This is determined by the expression $T = \frac{2\pi}{B}$. *In this example,* $T = 2\pi$, *as determined by the differences in time between peaks* P_1 *and* P_2 *on the graph where* $T = \frac{5\pi}{2} - \frac{\pi}{2} = \frac{4\pi}{2} = 2\pi$.
- **Frequency** (f): number of wave cycles that pass a fixed point in a given amount of time (usually 1 second). By definition, $f = \frac{1}{T}$, which is $= \frac{B}{2\pi}$.
  In this case, $f = \frac{2\pi}{2\pi} = 1\ Hz$.
- **D** shifts the wave horizontally, indicating a change in phase of the wave.
  *Negative values of **D** represent a translation to the right, while positive values of **D** represent a translation to the left.*
- **C** shifts the wave vertically.

Determining the pitch of a single note

Identification of Clip 1 from the time-domain

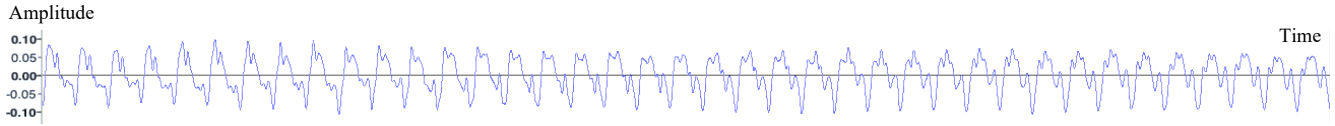


Figure 2: Clip 1 around 0.249000s to 0.337440s (time-domain)

As seen above, the frequency of a wave can be determined from a time-domain graph using its inverse relationship with time period. With my sample recorded using the program Audacity, I calculated the time period of the piano note A4 by taking an average over 39 cycles. Using a large number of cycles improves the reliability of calculating a single small time period, especially given the inadequacies with the software in displaying exact values for time, which is given to six significant figures. Copying and pasting the clip (*Figure 2*) into another Audacity file to more accurately gauge a reading for the time for 39 periods, I found this was 0.088413s. This means that one period is 0.00227s, and subsequently, the frequency is 441 Hz.

$$T = \frac{0.088413}{39} = 0.002267 = 0.00227s \text{ (3 s.f.)}$$

$$f = \frac{1}{T} = \frac{39}{0.088413} = 441.1116 = 441\ Hz \text{ (3 s.f.)}$$

The expected frequency was, however, 440 Hz, which defines the pitch A4. At first, I thought this error was due to my piano which had not been tuned for a few months. However, Audacity's spectral analysis (which uses the Fourier Transform) prompted the frequency to be 440 Hz (*Appendix 1*). I thus concluded that this error was possibly due to the inaccurate measurement for time due to the limit of reading, where the troughs could have not been precisely aligned. Another method has to be used in order to find the frequency, especially for multiple notes played together as this method only allowed for the frequency of one note to be calculated, as there is only one time period recurring.

Expressing a sound wave as a combination of constituent waves allows us to find the frequency components of the original sound wave as each of the constituent waves contain information about the frequency, how much of the frequency is present (amplitude) and its phase shift. *Figure 3* shows how adding two waves together changes the shape of the red resultant wave, and how this is represented in both the time and frequency domains.

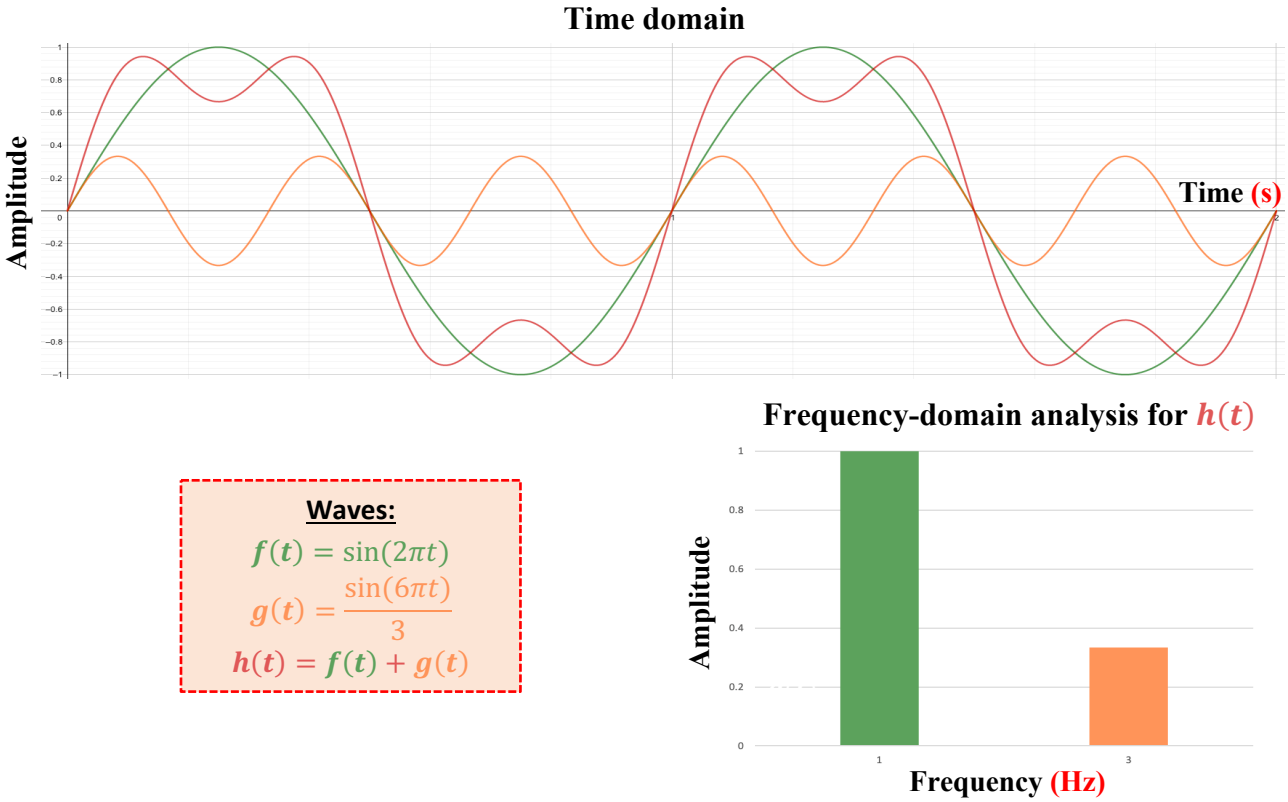


Figure 3: Representation of wave superposition in both time and frequency domains

The Fourier Transform converts or transforms a time-domain function into a frequency-domain graph, essentially representing the same signal from a different perspective, allowing us to find out about its frequency components.

Identification of Clip 1 from the frequency-domain

Audacity's spectral analysis feature allows us to obtain graphs such as *Appendix 1*, displaying the magnitude of frequency components, where the number of frequency components displayed is dependent on the number of samples used. In this case, 2048 samples were used. Lower numbers of samples used did not correspond to a peak at the expected frequency of 440 Hz as the input data was not sampled enough to be able to approximate the original time-domain signal, resulting in inaccurate results. The fundamental frequency being measured should be the most prominent sound (with the highest magnitude), defining the main pitch heard. However, the number of samples used was limited by the small amount of time (0.088413s) the clip was, although I could have extended the clip with more time-domain samples or by repeating the existing clip. For my investigation, I used the majority of the entire recording for my own calculations, which returned very good expected results. Also, as can be seen in *Appendix 1*, there are multiple smaller peaks after the one expected at 440 Hz. This is due to the **harmonic series**, a musical phenomenon where a fundamental frequency is joined by successive frequencies that are integer multiples of that fundamental, which is the lowest one of these collective frequencies (Nave, 2017, para. 1). This is why the peak at 440 Hz is joined by smaller peaks at 880 Hz, 1320 Hz, 1760 Hz etc.

Fourier Transforms and the DFT

Fundamentally, a transform extends the function of a function, converting a set of data into another set of data in the same number of samples, rather than just converting one or more inputs to one output as a regular function would. Jean-Baptiste Joseph Fourier, a French mathematician and physicist, demonstrated that any periodic signal could be expressed as a sum of sine waves with various amplitudes, frequencies and periods, devising the Fourier Transform to convert a set of signals between the time-domain and frequency-domain. This section investigates Discrete Fourier Transforms (DFT), one of the four types of Fourier Transforms that concerns discrete and periodic signals (Smith, 1997, p. 144). DFTs require evenly-spaced samples of time (displayed on the horizontal-axis of a time-domain graph) that are converted into frequency bins (which are also on the horizontal-axis on a frequency-domain graph). These frequency bins indicate frequency components in the sample where amplitude, displayed on the vertical-axis, depicts the magnitude or strength of these frequencies.

The DFT equation

$$F(k)=\sum_{n=0}^{N-1} f(n)\times e^{-i\frac{2\pi kn}{N}} \qquad \text{(Eq. 2)}$$

F(k): Magnitude of corresponding frequency bin *k* (on output frequency-domain function)
Values: [0,1,2,3…(N-1)]
f(n): Amplitude of corresponding sample *n* (from input time-domain function)
k: Discrete output frequency bin number
Values: [0,1,2,3…(N-1)]
n: Discrete input sample number
Values: [0,1,2,3…(N-1)]
N: Total number of samples
i : Imaginary number $\sqrt{-1}$
e and **π** are both constants

Note: the DFT notation may be confusing for some, so make sure you remember the difference between **Capital F**(k) and **lower-case f**(n)!

The analysing function $e^{-i\frac{2\pi kn}{N}}$, which is $(e^{\frac{-i2\pi}{N}})^{kn}$, can be expressed as W_N^{kn} where W_N, the primitive Nth root of unity, is $e^{-i\frac{2\pi}{N}}$. This analysing function can be expressed as a point moving clockwise (due to the negative in the formula) on a complex unit circle with circumference 2π. The $^{kn}/_{N}$ portion indicates the proportion of the circumference the point is at.

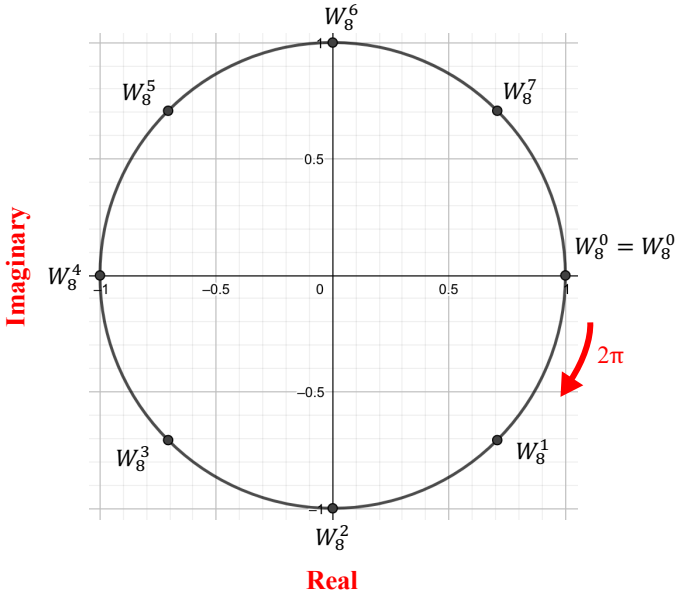


Figure 4: N^{th} roots of unity, with N=8 as an example

The Nth roots of unity have two properties:

1. **Periodic Property**

   As can be demonstrated in *Figure 4*, the roots of unity are periodic in that as you move along the unit circle and the powers of W_N incrementally increase at each point, if the power is greater than a multiple of N, it is equivalent to the root of unity without the integer multiple of N.

   $$W_N^{kn+mN} = W_N^{kn} \text{ where } m \in \mathbb{Z}$$

   e.g.

   $$W_8^8 = W_8^0$$
   $$W_8^{12} = W_8^4$$
   $$W_8^{24} = W_8^0$$
   $$W_8^{31} = W_8^7$$

2. Symmetry Property

A complex point can be expressed as the negative of its diagonally-opposite or symmetrical root of unity, where the signs of both imaginary and real values are flipped.

In *Figure 5*, Z_1 (+a+ib) and Z_3 (-a-ib) are symmetrical, where Z_3 (-a-ib) can be expressed as -(a+ib), or $-Z_1$.

As can be seen in *Figure 4*:

$$W_8^4 = -W_8^0$$
$$W_8^5 = -W_8^1$$
$$W_8^6 = -W_8^2$$
$$W_8^7 = -W_8^3$$
$$W_8^7 = -W_8^3$$

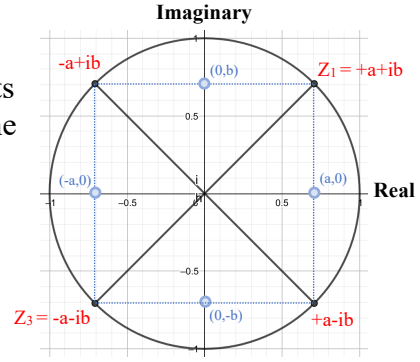


Figure 5: Symmetry Property

Using Euler's Identity to expand the analysing function $e^{-i\frac{2\pi kn}{N}}$:

Since $e^{i\theta} = \cos(\theta) + i\sin(\theta)$

$$e^{i\frac{-2\pi kn}{N}} = \cos\left(\frac{-2\pi kn}{N}\right) + i\sin\left(\frac{-2\pi kn}{N}\right)$$

$\therefore$ Equation 2 can be rewritten as: $F(k) = \sum_{n=0}^{N-1} f(n) \times \left(\cos(\frac{-2\pi kn}{N}) + i\sin(\frac{-2\pi kn}{N})\right)$

By expanding the analysing function, it can be seen that in the complex notation of the DFT equation, cosine and sine waves are added together, representing the superposition theory from before. This means we can find values for the analysing function by finding values for both cosine and sine components, substituting in corresponding k and n values. The cosine and sine values form a complex vector:

$$\cos(\frac{-2\pi kn}{N}) + i\sin(\frac{-2\pi kn}{N})$$
$$a + ib$$

By using an Argand diagram (*Figure 6*), information about magnitude and phase can be extracted from the complex vector ($a + ib$) which results from summing up all the cosine and sine components for each k[th] frequency bin.

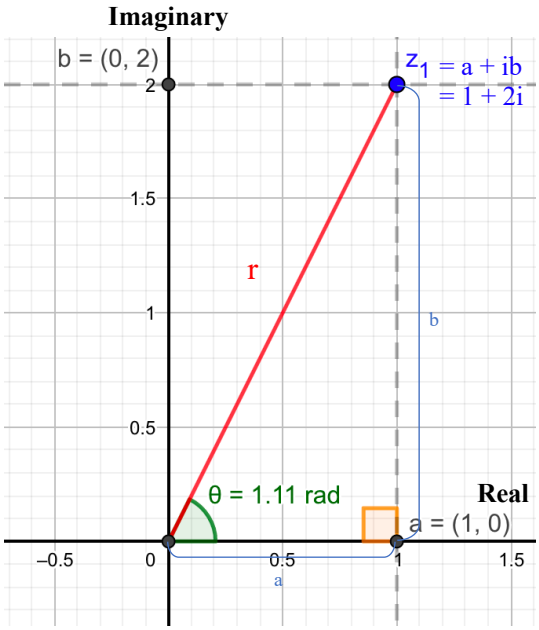


Figure 6: Argand diagram for 1+2i

r is the magnitude of the complex number Z_1, which can be calculated using Pythagorean's Theorem:

$$r^2 = a^2 + b^2$$
$$\therefore r = \sqrt{a^2 + b^2}$$
$$= \sqrt{1^2 + 2^2} = \sqrt{5}$$

θ indicates the phase shift, which can be calculated using trigonometry:

$$\tan\theta = \frac{b}{a}$$
$$\therefore \theta = \arctan\left(\frac{b}{a}\right)$$
$$\theta = \arctan\left(\frac{2}{1}\right) = \arctan 2 = 1.11 \text{ rad (3 s.f.)}$$

We can then plot frequency bin *k* (horizontal axis) and its corresponding magnitude *F(k)* (vertical axis) on a frequency-domain graph, such as on *Pages 9-10*. Furthermore, phase information can be combined with its corresponding frequency information to prove that individual sine curves can be added together to reconstruct the original time-domain signal.

Identification of the Pure Tone A4 using the DFT (simplified example)

For the purposes of simplicity, I decided to use the DFT equation to analyse the frequency spectrum of the pure tone of A4. This is an electronically-generated tone (using Audacity) that produces a pure sine wave (as can be seen in *Figure 7*), without any instrumental harmonics.

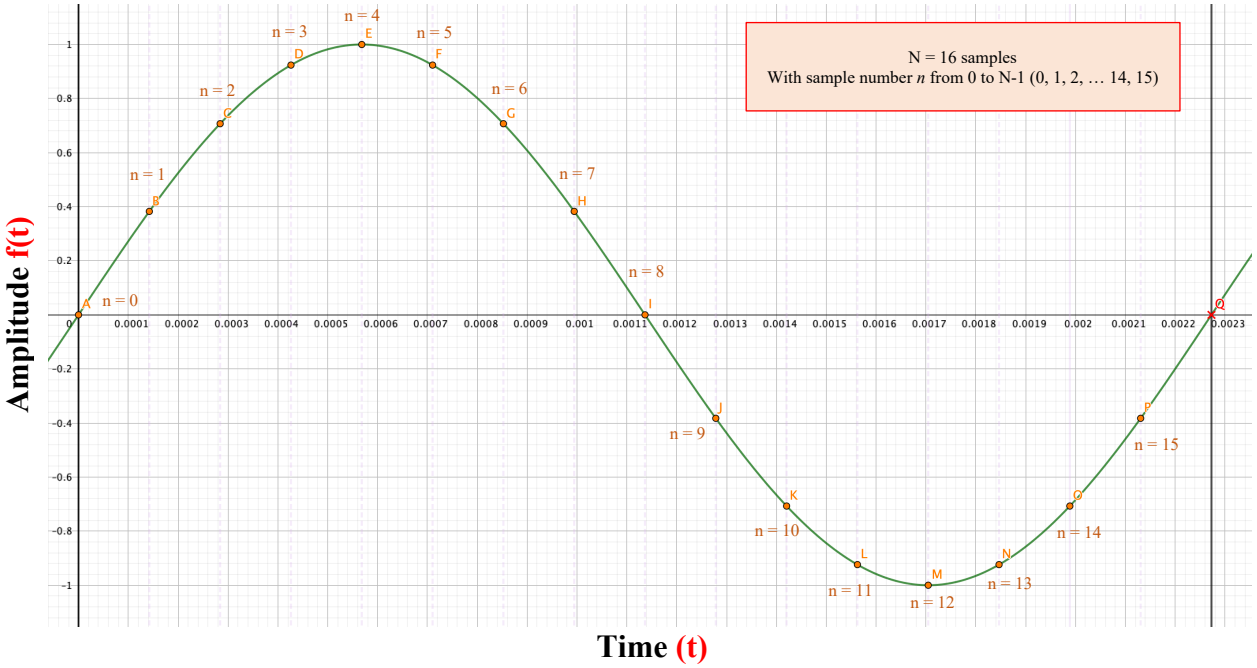


Figure 7: A4 Pure tone time-domain graph

Sampling frequency f_s is the number of samples in a second. In this example, I have:

16 samples in 0.00227s

$$\therefore f_s = N \times \frac{1}{T} = 16 \times \frac{1}{0.00227} = 7048.458 Hz \text{ (3 d.p.)}$$

My obtained input time-domain data is expressed in the first two columns of *Table 1*.

$$F(k) = \sum_{n=0}^{N-1} f(n) \times e^{-i\frac{2\pi kn}{N}}$$

Examining the DFT equation, we will need N number of frequency bins (output), since k = 0,1,2,3…(N-1). For each frequency bin, k, there are also N number of n samples considered, where n = 0,1,2,3…(N-1). Thus, the order of computational complexity is $O(N^2)$.

Performing the calculations:

For Frequency Bin k = 0

$$F(0) = \sum_{n=0}^{N-1} f(n) \times e^{-i\frac{2\pi 0n}{16}} = \sum_{n=0}^{N-1} f(n) \times e^{0} = \sum_{n=0}^{N-1} f(n) = f(0) + f(1) + f(2) + f(3) + \ldots + f(14) + f(15) = 0$$

This was the expected value since f(n) is periodic and symmetric about the time-axis.

For Frequency Bin k = 1

$$F(1)=\sum_{n=0}^{N-1} f(n)\times e^{-i\frac{2\pi n}{16}} = \sum_{n=0}^{N-1} f(n)\times e^{-i\frac{\pi n}{8}}$$

The analysing function can be expanded using Euler's Identity

$$=\sum_{n=0}^{N-1} f(n)\times\left(\cos(\frac{-\pi n}{8})+i\sin(\frac{-\pi n}{8})\right)$$

$$=\sum_{n=0}^{N-1} f(n)\times\cos(\frac{-\pi n}{8})+\sum_{n=0}^{N-1} f(n)\times i\sin(\frac{-\pi n}{8})$$

This can now be calculated as a complex sum

$$= f(0)\times\left(\cos(0)+i\sin(0)\right)+ f(1)\times\left(\cos(\frac{-\pi\times1}{8})+i\sin(\frac{-\pi\times1}{8})\right)+ f(2)\times\left(\cos(\frac{-\pi\times2}{8})+i\sin(\frac{-\pi\times2}{8})\right)$$

Substituting n = 0 ; n = 1 ; n = 2

$$+ f(3)\times\left(\cos(\frac{-\pi\times3}{8})+i\sin(\frac{-\pi\times3}{8})\right)+\ldots+ f(14)\times\left(\cos(\frac{-\pi\times14}{8})+i\sin(\frac{-\pi\times14}{8})\right)+ f(15)\times\left(\cos(\frac{-\pi\times15}{8})+i\sin(\frac{-\pi\times15}{8})\right)$$

n = 3 ; … ; n = 14 ; n = 15

I calculated the analysing function separately in terms of its cosine and complex sine components before adding them together, which is expressed on the table below:

n	f(n)	$f(n)\times\cos(\frac{-\pi n}{8})$	$f(n)\times i\sin(\frac{-\pi n}{8})$
0	0.00000	0	0
1	0.38268	0.35355	-0.14645i
2	0.70711	0.50000	-0.50000i
3	0.92388	0.35355	-0.85355i
4	1.00000	0	-i
5	0.92388	-0.35355	-0.85355i
6	0.70711	-0.50000	-0.50000i
7	0.38268	-0.35355	-0.14645i
8	0.00000	0	0
9	-0.38268	0.35355	-0.14645i
10	-0.70711	0.50000	-0.50000i
11	-0.92388	0.35355	-0.85355i
12	-1.00000	0	-i
13	-0.92388	-0.35355	-0.85355i
14	-0.70711	-0.50000	-0.50000i
15	-0.38268	-0.35355	-0.14645i
Total		0	$4\times(-0.14645-0.50000-0.85355)i-2i=-8i$

Table 1: DFT calculation for the first frequency bin k =1

$\therefore F(1)=0-8i$

Magnitude = $|F(1)|=\sqrt{0^2+(-8)^2}=\sqrt{64}=8$

Phase shift = $\arctan(-8/0)$ = *undefined* as dividing by 0 ∴ No phase shift

Completing a similar computation for the other frequency bins, I obtained the following data:

k	F(k)	Magnitude
0	$0+0i$	0
1	$0-8i$	8
2	$0+0i$	0
3	$0+0i$	0
4	$0+0i$	0
5	$0+0i$	0
6	$0+0i$	0
7	$0+0i$	0
8	$0+0i$	0
9	$0+0i$	0
10	$0+0i$	0
11	$0+0i$	0
12	$0+0i$	0
13	$0+0i$	0
14	$0+0i$	0
15	$0+8i$	8

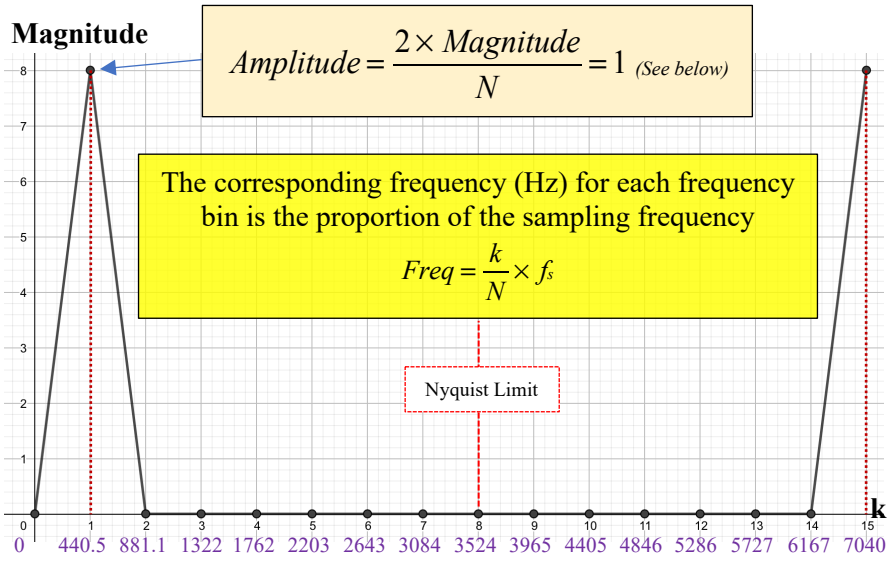


Figure 8: Frequency spectrum for pure tone A4

As can be seen on *Figure 8*, there is a peak at 440.529 Hz, indicating that it is the most dominant frequency, or only dominant frequency in this example. The second peak at 7040 Hz can be ignored, as all values above the Nyquist Limit of N/2 (3524 Hz) can be ignored as they represent negative frequencies. Since the input function is periodic, the output frequency function is also periodic, so these values would be present to the left of the vertical axis as negative frequencies. We can disregard these redundant values and just plot up to the Nyquist Limit and double the remaining magnitudes. The magnitude obtained is calculated using N number of n samples, which has to be averaged out by dividing by N in order to find amplitude.

Since DFTs like these take up a very long time to compute by hand, I decided to write a script in MATLAB (*see below*) to aid the calculation process, which allowed me to compute the large number of 65536 samples in each of my recorded data sets. With my minimal coding experience, I had to learn how to use MATLAB and program in its syntax, which was unexpectedly time-consuming. In the end, I was able to develop a general DFT program that works for all values of N. Although I expected the computational time to take a while, I did not expect that it would take as long as it did, as will be seen later in the investigation.

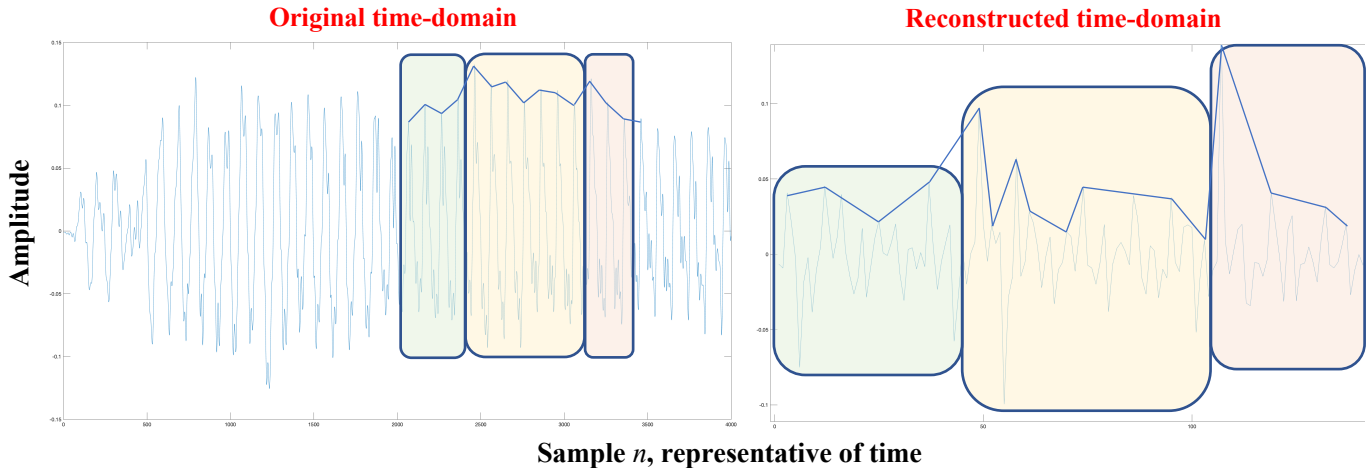
```
% Importing the data
[Y,X] = audioread('SingleNoteClip.wav'); % Processes the audio file
f = Y(1:65536,1); % Selects and creates an array for the amplitudes
N = length(f); % N is the total number of samples

% Displays time-domain graph (input)
figure;
plot(f)
title('Input signal');
xlabel('Time (s)');
ylabel('Amplitude');

% Setting up arrays
r = zeros(N,1); % Magnitude
p = zeros(N,1); % Phase
```

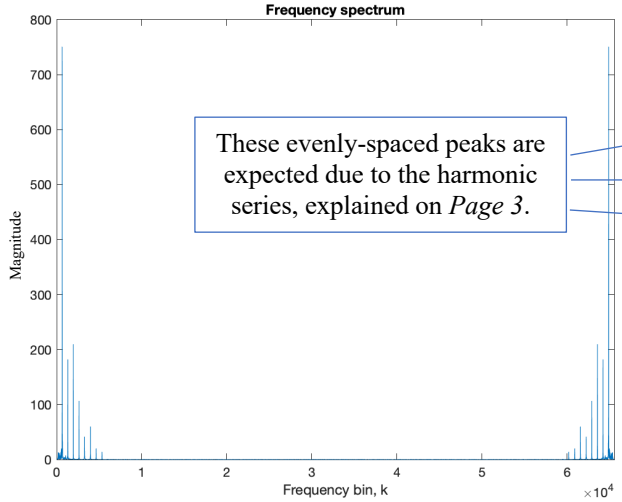
```
% Recursive calculation
for k = 0:(N-1)
    for n = 0:(N-1)
        b(n+1) = f(n+1)*(sin((-2*n*k*pi)/N));
        a(n+1) = f(n+1)*(cos((-2*n*k*pi)/N));
    end

% Calculating magnitude using arrays a and b
r(k+1) = sqrt(sum(a)^2 + sum(b)^2);
% Calculating phase using arrays a and b
p(k+1) = atan(sum(b)/sum(a));

end

% Displays frequency-domain graph (output)
figure;
plot(r)
title('Frequency spectrum');
xlabel('Frequency bin, k');
ylabel('Amplitude');
xlim([0 N]);
```

I had to compensate by moving the arrays by one sample to the right since MATLAB cannot deal with array numbers of zero. I will also have to compensate for this when interpreting the graph, which does not begin from k=0, instead shifted one point to the right.

I also programmed another script (*Appendix 2*) which attempts to reconstruct the original time-domain graph by adding together many sine waves with the frequency values obtained, and their corresponding amplitude and phase information, using the basic definition of a sine wave (Equation 1). This process is important in compressing file size, typically converting to the frequency domain to remove certain unperceivable frequencies before converting back to the time-domain, without the frequencies. However, the Inverse DFT is commonly used for this purpose. As can be seen below, my method provides a very close approximation to the original sampled data, which proves the superposition property of sinusoidal waves.

Single Note: A4 (440 Hz)





These evenly-spaced peaks are expected due to the harmonic series, explained on *Page 3*.

Peaks				
k	**Frequency** (Hz)	**Pitch**	**Magnitude**	**Amplitude**
654	440.085	**A4**	750.7143	0.0229
1308	880.170	A5	182.0089	0.0056
1966	1322.946	E6	209.6353	0.0064
2627	1767.741	A6	106.5609	0.0033

Note: the actual frequency bin value is one less than the one plotted, as explained above.

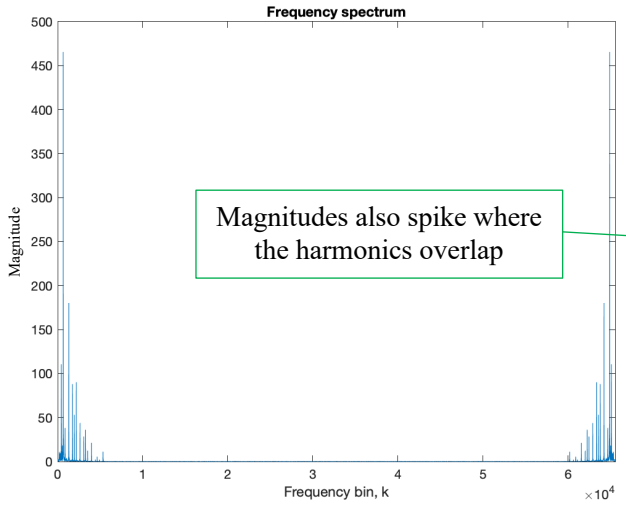
Calculation time: 715.092s = 11 min 55 sec

The pitches corresponding to the obtained frequencies can be found by comparing them against Michigan Technological University's Pitch-Frequency table (Suits, 2019).

The fundamental frequencies identified (corresponding to peaks) indicate the piano notes recorded in the sample. These frequencies are followed by its overtone harmonics.

Two Simultaneous Notes: D4 (293.66 Hz) and A4 (440 Hz)

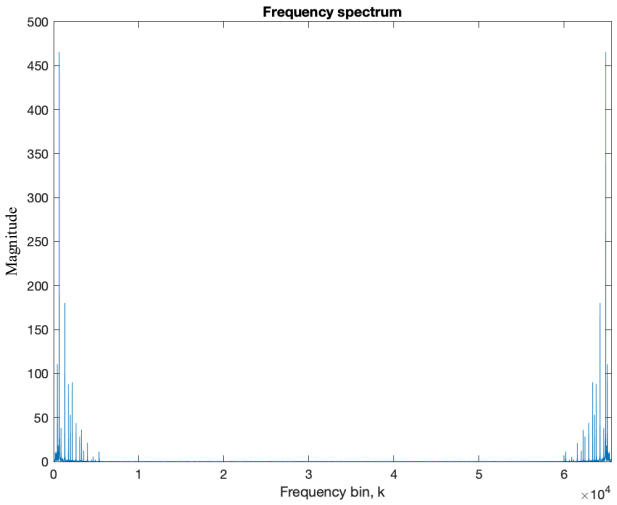
When calculating amplitude, note that Audacity uses F_s=44100 Hz



Peaks				
k	**Frequency** (Hz)	**Pitch**	**Magnitude**	**Amplitude**
436	293.390	**D4**	110.5139	0.0034
655	440.085	**A4**	465.5897	0.0142
873	587.453	D5	38.1401	0.0012
1308	880.170	A5	180.2467	0.0056
1749	1176.924	D6	88.0187	0.0027
1966	1322.946	E6	52.9893	0.0016
2190	1473.679	F#6	90.0701	0.0027
2627	1767.741	A7	43.7081	0.0013

Calculation time: 745.023s = 12 min 42 sec

Three Simultaneous Notes: D4 (293.66 Hz), F4 (349.23 Hz) and A4 (440 Hz)



Peaks				
k	**Frequency** (Hz)	**Pitch**	**Magnitude**	**Amplitude**
436	293.390	**D4**	529.3797	0.0162
519	349.242	**F4**	2146.804	0.0655
653	439.412	**A4**	459.5424	0.0140
873	587.453	D5	294.0187	0.0090
1039	699.156	F5	214.0631	0.0065
1310	881.516	A5	204.6125	0.0062
1561	1050.417	C6	102.0342	0.0031
1750	1177.597	D6	55.7549	0.0017
1968	1324.292	E6	91.9704	0.0028

Calculation time: 723.214s = 12 min 3 sec

The Fast Fourier Transform (FFT)

The FFT is an algorithm that computes the DFT faster, especially for large computations (high numbers of samples, *N*). Although the DFT and FFT both concern discrete signals, if calculations are fast enough, the DFT can be calculated as if there was a continuous input signal, enabling live calculations with minimal delay, which has especially useful applications for tuners and sonars. I decided to investigate the FFT after realising one of the weaknesses of using the DFT – that it takes a very long time to compute, especially for large numbers of samples. The sorting algorithm is best utilised when N is a power of two due to the subdivision of a DFT into two smaller DFTs, as will be explained later in this section. However, the input data can be appended with zeroes (zero-padding) to the next highest power of two in order to optimise the efficiency of the algorithm. This is more ideal than removing samples, which decreases the accuracy of results obtained instead as less samples are reviewed by the program.

F(k) with total N number of samples		
=	**Even inputs** $F_0(k)$ with total N/2 number of samples	+ **Odd inputs** $F_1(k)$ with total N/2 number of samples

FFT algorithms play on the idea that for each DFT, the order of computational complexity is the total number of samples squared, i.e. for a DFT with N samples, the order of computational complexity is $O(N^2)$. If we split the N-point DFT into two smaller DFTs, namely subdivisions for even and odd indices of the input function, each only requiring a total of N/2 samples now, the computational complexity is effectively halved.

$$O = \left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 = \frac{N^2}{4} + \frac{N^2}{4} = \frac{2N^2}{4} = \frac{N^2}{2}$$

Each time we split the DFT into even and odd components, the order of computational complexity is halved. Performing this recursively, the computational complexity can be reduced to $O(N \log_2 N)$. This is especially significant for large values of N such as my data set, translating to very large reductions in computational time, as will be demonstrated later.

Algorithm Derivation

Using the W_N notation explained on *Page 4*, the DFT equation (Eq. 2) can be rewritten as:

$$F(k) = \sum_{n=0}^{N-1} f(n) \times W_N^{kn} \qquad \text{(Eq. 3)}$$

which is represented in *Figure 9*

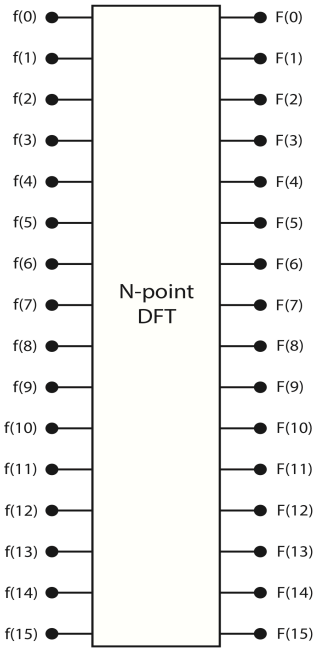


Figure 9: Original N-point DFT

STAGE 1

Expressing Equation 3 as the sum of two smaller DFTs each with half the total number of samples (N); with n = 2n representing the even-index inputs and n = 2n + 1 representing the odd-index inputs:

$$F(k) = \underbrace{\sum_{n=0}^{N/2-1} f(2n) \times W_N^{k(2n)}}_{\text{EVEN}} + \underbrace{\sum_{n=0}^{N/2-1} f(2n+1) \times W_N^{k(2n+1)}}_{\text{ODD}} \quad \text{where } n = 0 \ldots N/2 - 1$$

Simplifying this:

$$F(k) = \sum_{n=0}^{N/2-1} f(2n) \times W_N^{2kn} + \sum_{n=0}^{N/2-1} f(2n+1) \times W_N^{2kn+k}$$

$$F(k) = \sum_{n=0}^{N/2-1} f(2n) \times W_N^{2kn} + \sum_{n=0}^{N/2-1} f(2n+1) \times W_N^{2kn} \times W_N^{k}$$

Since $W_N^{2kn} = e^{-i\frac{2\pi kn}{N} \times 2} = e^{-i\frac{2\pi kn}{N/2}} = W_{N/2}^{kn}$

Rearranging so there are common elements allows for savings in computational time and storage!

$$\therefore F(k) = \sum_{n=0}^{N/2-1} f(2n) \times W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} f(2n+1) \times W_{N/2}^{kn}$$

$$\therefore F(k) = F_0(k) + W_N^k F_1(k) \qquad \text{(Eq. 4)}$$

where $F_0(k) = \sum_{n=0}^{N/2-1} f(2n) \times W_{N/2}^{kn}$ (Eq. 5) and $F_1(k) = \sum_{n=0}^{N/2-1} f(2n+1) \times W_{N/2}^{kn}$ (Eq. 6)

A DFT can be expressed as $F_0(k)$, the DFT of the even inputs, plus a constant multiplied by $F_1(k)$, the DFT of the odd inputs, where each smaller DFT has half the original number of samples.

Using the periodic property of DFTs G(k) and H(k), which are both N/2 periodic:

Since $k = k + N/2$,

$F_0(k) = F_0(k + N/2)$

$F_1(k) = F_1(k + N/2)$

$F(k) = F(k + N/2)$

Substituting the above into Equation 4,

$$F(k+N/2) = F_0(k+N/2) + W_N^{k+N/2} F_1(k+N/2)$$

$$= F_0(k) + W_N^{k+N/2} F_1(k)$$

This can be determined using Euler's Identity

$$W_N^{k+N/2} = e^{-i\frac{2\pi(k+N/2)}{N}} = e^{\frac{-i2\pi k}{N} + \frac{-i2\pi\frac{N}{2}}{N}} = e^{\frac{-i2\pi k}{N}} \times e^{-i\pi} = W_N^k \times (-1) = -W_N^k$$

$$\therefore F(k+N/2) = F_0(k) - W_N^k F_1(k) \qquad \text{(Eq. 7)}$$

And from Equation 4, $F(k) = F_0(k) + W_N^k F_1(k)$

Equations 4 and 7 can be expressed using a 'butterfly' diagram, which depicts the flow of a signal:

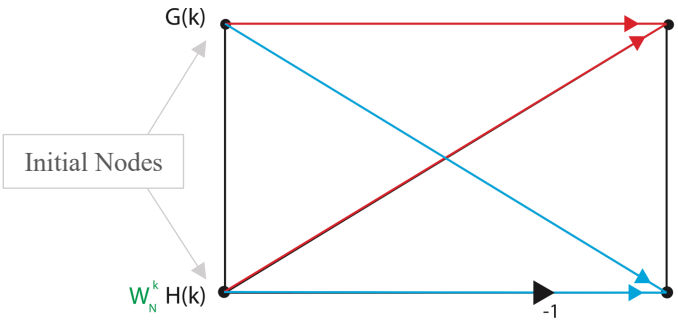


Figure 10: Explanation of a 'Butterfly'

The value at each node is the sum of the lines approaching it. The value of each line is given by the initial node, and if there is a value presented on a line connecting the two nodes (i.e. -1), the value at the initial node is multiplied by it.

Butterfly diagram expressing a N = 16 point DFT as the sum of two N = 8 point DFTs:

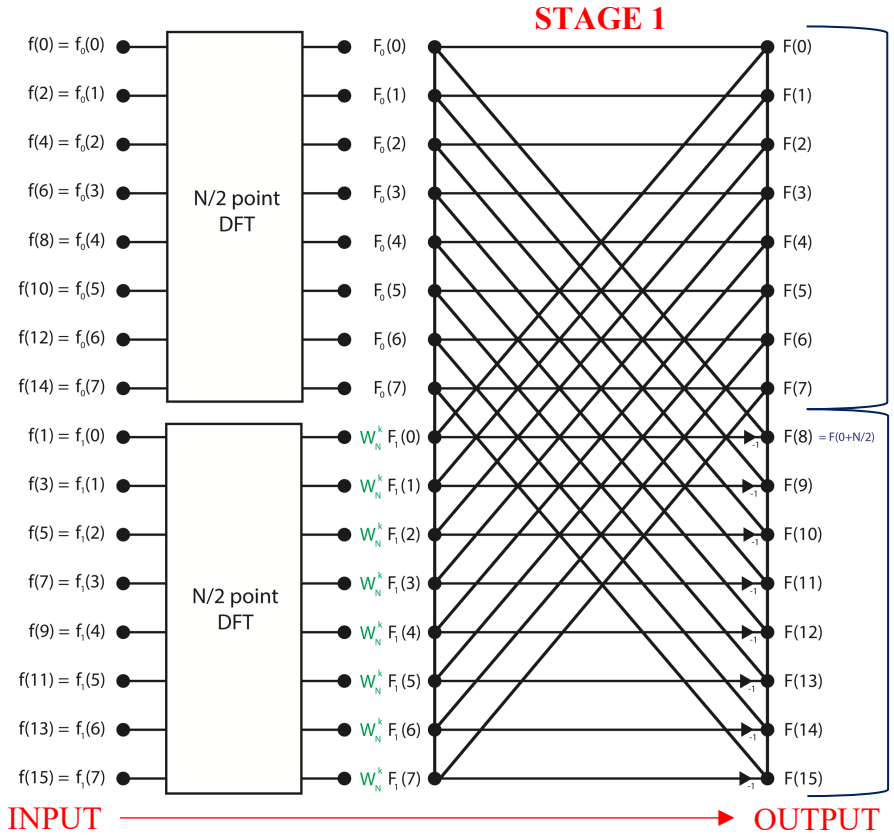


Figure 11: Stage 1 Butterfly Diagram

<u>Note:</u>

The k value in W_N^k is the k value in the function F(k) it is multiplied by.

i.e. $W_N^0 F_1(0)$

$W_N^1 F_1(1)$

STAGE 2

Each N/2 point DFT can be further simplified, split into its own even and odd arrays. Equation 5 can be expressed as:

$$F_0(k) = \sum_{n=0}^{N/4-1} f_0(2n) \times W_{N/2}^{k(2n)} + \sum_{n=0}^{N/4-1} f_0(2n+1) \times W_{N/2}^{k(2n+1)}$$

$$F_0(k) = \sum_{n=0}^{N/4-1} f_0(n) \times W_{N/4}^{kn} + W_{N/2}^{k} \sum_{n=0}^{N/4-1} f_0(n) \times W_{N/4}^{kn}$$

$$\therefore F_0(k) = F_{00} + W_{N/2}^{k} F_{01}$$

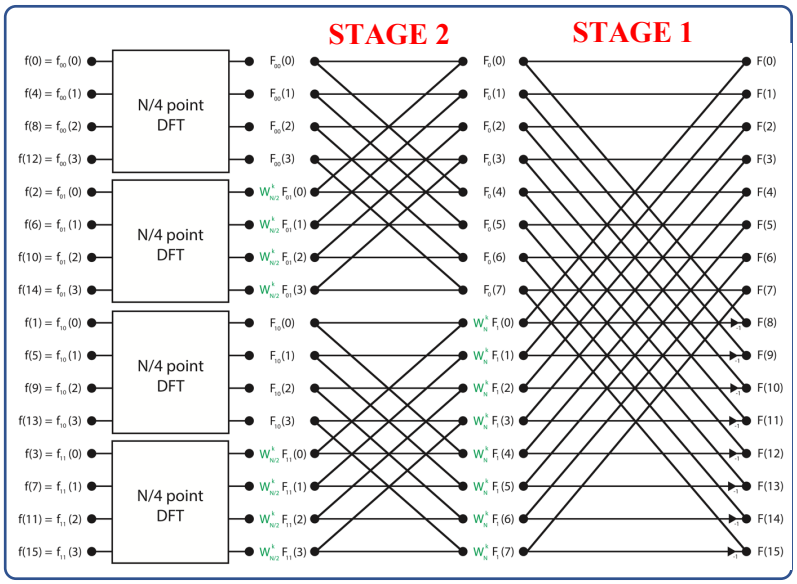


Figure 12: Stage 2 Butterfly Diagram

Since the functions are N/4 periodic, $F_0(k+N/4) = F_{00} - W_{N/2}^{k} F_{01}$

Similarly for Equation 6, $F_1(k) = F_{10} + W_{N/2}^{k} F_{11}$ and $F_1(k+N/4) = F_{10} - W_{N/2}^{k} F_{11}$

The 0's and 1's represent sorting into even and odd arrays. For example, $F_{01}(k)$ is every odd, or $(2n+1)^{th}$ term of $F_0(k)$. $F_{10}(k)$ is every even, or $(2n)^{th}$ term of $F_1(k)$. Please refer to Figure 14.

Recursion

This split can be recursively repeated for $\log_2 N$ stages, or until only the single butterflies of *Figure 10* remain, which cannot be further divided. At this point, there are N split-functions, and each unique with only one value of k as 0 in its array, as the final line in *Figure 14*, i.e. all the split-functions become $f_x(0)$ and $W_x^0 = 1$, where x is any possible value. For a N = 16 point DFT, there are 4 stages.

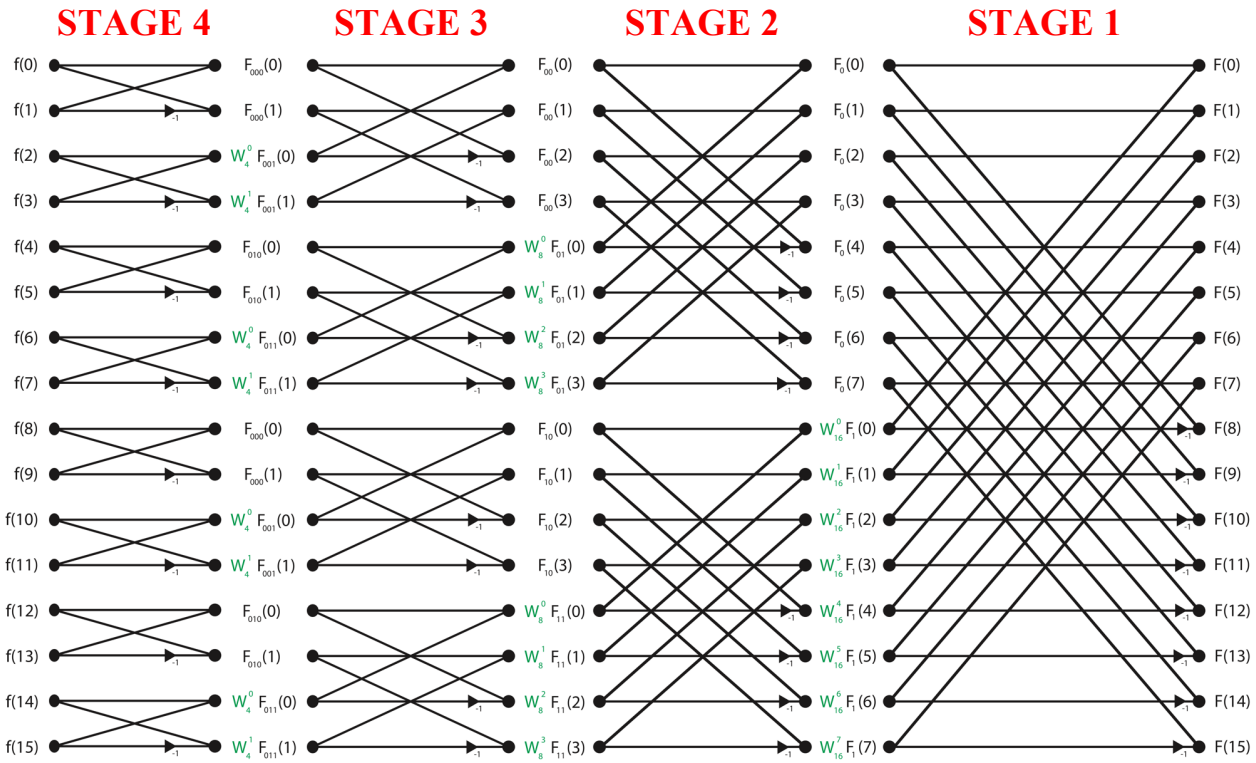


Figure 13: Complete Butterfly Diagram for a N = 16 point DFT

Due to the calculations, the inputs to each butterfly are reordered, but can be determined using the following diagram:

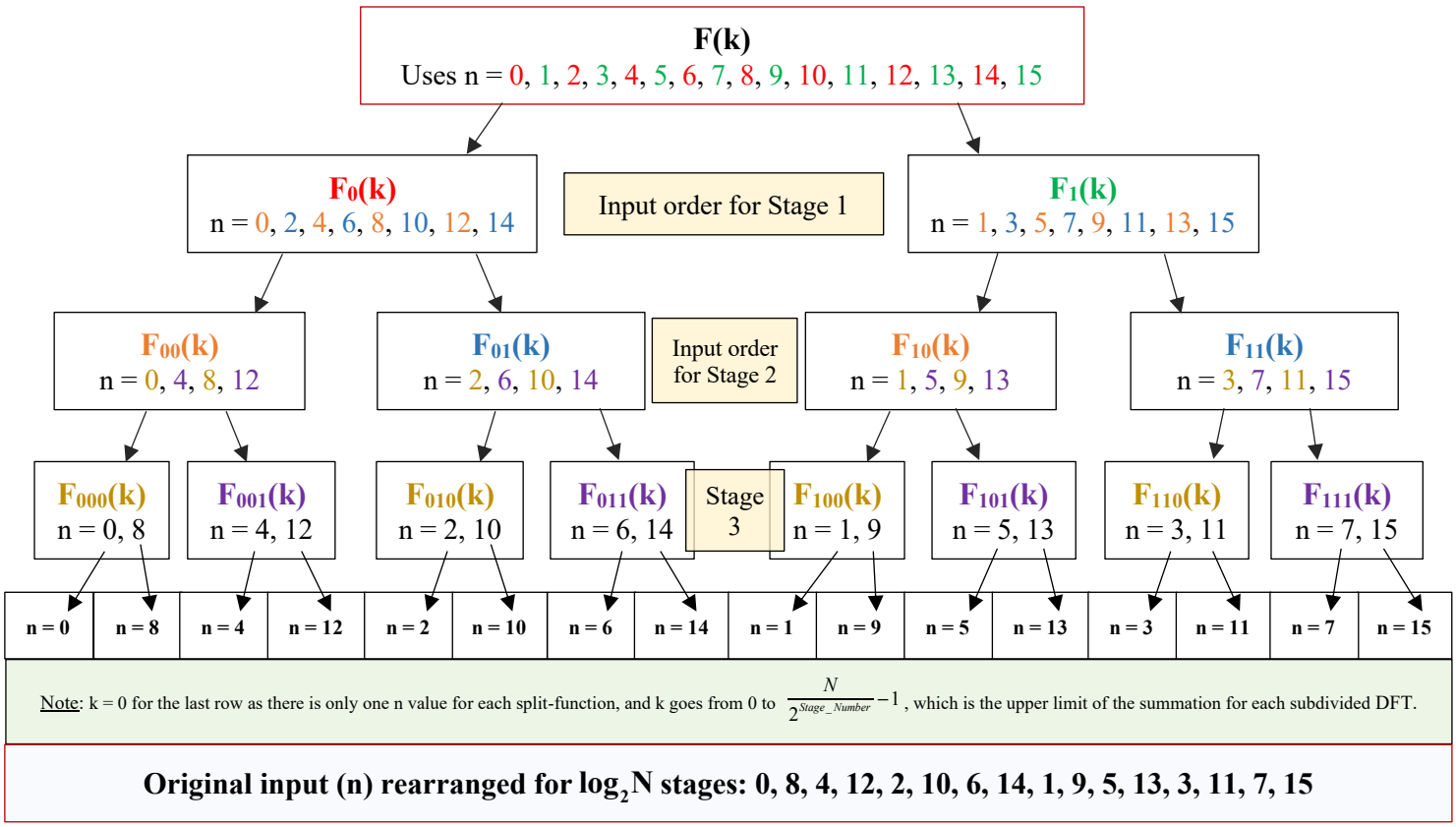


Figure 14: Rearranged inputs for each subdivided DFT (or each 'Butterfly')

The FFT algorithm depicted in *Figure 13* can be developed as a MATLAB program. Although MATLAB has its own in-built FFT function, I tried to make my own program as I could not determine which specific FFT algorithm MATLAB used, as there are several other algorithms that optimise calculation time for certain values of N, such as the Prime Factor Algorithm which splits DFT samples based on prime factors, Winograd Algorithm that factorises into polynomials (Kumar et al., 2013, pp. 24-25) and the RADIX 4 algorithm which splits into four smaller DFTs rather than two as I did (Jayacram & Arun, 2015, p. 5149). Although I attempted creating a completely original program, I could not successfully complete it due to the difficulty of the problem, coupled with limitations of experience and time. After reviewing many other algorithms, such as Dr Gylson Thomas' script which was similar to the script I had started working on but contained several errors (Thomas, 2006), I was able to finish writing my program and continue with my investigation to obtain the following results. I also added a zero-padding feature which enables FFT computations for input data where the total number of samples aren't powers of two. A copy of my program can be found in *Appendix 3*.

Results

As expected, the FFT algorithm computed the same frequency bin results and subsequent graphs as my DFT script. However, the most noticeable difference was in the calculation time:

1. **Single-Note Clip:** 0.820 seconds
2. **Two-Note Clip:** 0.898 seconds
3. **Three-Note Clip:** 0.861 seconds

The FFT computed the DFT with an average of 0.860 seconds, whilst the DFT script took an average of 12 minutes and 7.776 seconds. The FFT was 846.251 times as fast as the DFT script. This demonstrates the significant improvements in calculation time of using the FFT algorithm instead of the original DFT equation, especially for a large number of samples.

Conclusion

My investigation successfully answers my objective of using Fourier Transforms to discern musical pitches, including instances where multiple notes are played simultaneously. By transforming a time-domain signal into the frequency-domain, the magnitudes of frequency components can be analysed to determine what the fundamental frequencies are, which represent the notes played on my piano. Although I had initially planned to only explore the DFT, I was curious why the DFT for my 65536 samples took so long to compute, taking an average of around 12 minutes whilst Audacity's Spectrum Analysis feature, which also uses Fourier Transforms, computed it in less than a second. Upon research, I found out about the FFT algorithm it used and struggled with understanding its mathematics for a very long time, which was a concept that was often poorly explained. Since MATLAB did not have a built-in DFT program, I decided to write the DFT program myself, which turned out to be a rather time-consuming process as I had to also learn the syntax associated with the programming language. Although after the DFT I would be more experienced when writing my FFT program, realising the logic behind the FFT took a very long time, and trying to translate this to code proved to be an even more difficult task. In order to complete my investigation, after many months of research, I was forced to search for FFT programs that other MATLAB users had devised and use them as inspiration for my program. Upon reflection, I should have done this sooner because a lot of time was wasted in trying to devise a completely original solution.

Overall, the investigation was very successful, where I obtained expected results that matched the notes played on the piano for each clip. I also took the investigation a step further by trying to reconstruct the original time-domain signal using the complex data obtained, which provided an approximation for the signal, although the amplitudes were not very accurate since the Fourier Transform takes an average using all the samples. The research process in firstly understanding the DFT and then the FFT, was a major challenge that I managed to overcome after an extended period of time. The FFT brings significant computational time-saving benefits to the many applications of the DFT, and I am glad that I eventually managed to figure it out by myself, after watching countless hours of online lectures that did not explain the concept very well. Although I managed to learn a lot of related mathematical concepts in the process, I was not very satisfied with the amount of time invested into this investigation, which compromised my time spent on other subjects and put me under a lot of stress. There were also many revisions of the report, where I consistently removed sections not relevant to my objective, allowing me to produce a more coherent and concise report. There are many methods of explaining the mathematics behind the FFT. I had originally planned to derive the FFT using a matrices approach, as many universities did. However, I was unsuccessful in understanding that and used the approach described in the investigation, which I derived myself.

The magnitudes of frequencies in the data obtained could have not precisely reflected what was played on the piano as the strength of a particular frequency changes at various times due wave interference, dictating how different sound waves superimpose, which determines its strength. The piano notes could have also been out of tune, and the recorder (my laptop) could have picked up on fewer harmonic frequencies or was more sensitive towards certain frequencies, affecting the input data and consequently, the results as well. Calculations of the DFT and FFT depict the frequency components of the entire sampled signal but do not show how the relative strengths of frequency components change over time. A spectrogram, as an extension of the Fourier Transform, is a visual representation that conveys all three concepts, where the strength of a frequency is expressed on a frequency-time axis with different colours. This could, for example, enable a computer to notate a recording of a musical piece and include information about dynamics (volume), rhythm (time) and pitches or notes (frequencies).

References

Boston University Department of Physics. (n.d.). *Interference of Waves*. Retrieved 20 May 2019 from http://physics.bu.edu/~duffy/py105/WaveInterference.html

Jayaram, K, & Arun, C. (2015). Survey Report for Radix 2, Radix 4, Radix 8 FFT Algorithms. *International Journal of Innovative Research in Science, Engineering and Technology*, 4(7), 5149-5154.

Kumar, P. K. M., Jain, P., Kiran, R. S., Rohith, N., & Ramamani, K. (2013). FFT Algorithms: A Survey. *The International Journal of Engineering and Science*, 2(4), 22-26.

Nave, C. R. (2017). *Fundamental and Harmonics*. Retrieved 18 May 2019 from http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/funhar.html

Royakkers, L., Timmer, J., Kool, V., Est, R.V. (2018). Societal and ethical issues of digitization. *Ethics and Information Technology*, 20(2), 1.

Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, California: California Technical Publishing.

Suits, B. H. (2019). *Frequencies of Music Notes*. Retrieved 16 May 2019 from http://pages.mtu.edu/~suits/notefreqs.html

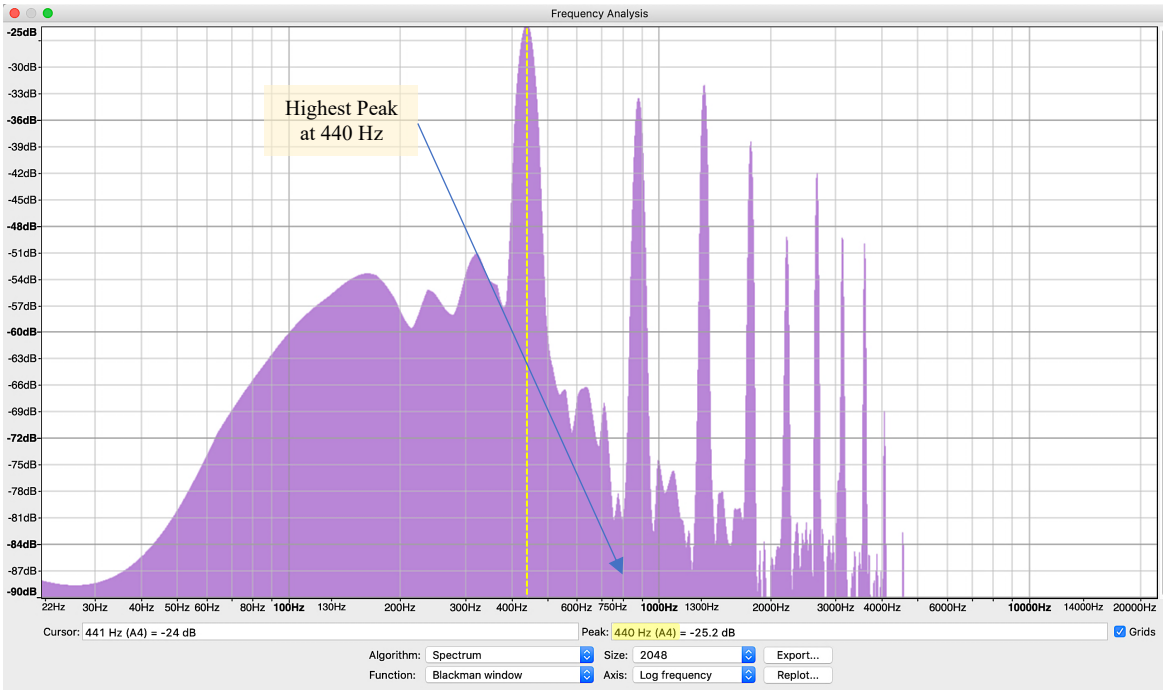
Teach Me Audio. (2019). *Fundamental & Harmonic Frequencies*. Retrieved 20 May 2019 from https://www.teachmeaudio.com/recording/sound-reproduction/fundamental-harmonic-frequencies/

Thomas, G. (2006). *Radix2 decimation in time 1D fast Fourier transform FFT*. Retrieved 1 August 2019 from https://au.mathworks.com/matlabcentral/fileexchange/13248-radix2-decimation-in-time-1d-fast-fourier-transform-fft

University of California Berkeley Department of Astronomy. (n.d.). *Applications for Fourier*. Retrieved from http://w.astro.berkeley.edu/~jrg/ngst/fft/applicns.html

Appendix

Appendix 1: Audacity's Frequency Analysis of Clip 1, using a sampling size of 2048 samples



Appendix 2: MATLAB Script Time-domain Reconstruction using the basic sine wave equation

```
% Reconstructing the original time-domain signal
S = zeros(N/2,1) % Setting up an array for the signal

% Adding multiple real sine curves together, substituting array values
into Equation 1
for t = 1:(N/2)
    for c = 1:(N/2)
        S(t) = S(t)+(2*(r(c))/N)*sin(2*pi*t*((44100*c)/N)+p(c));
    end
end

% Displays reconstructed time-domain graph
figure;
plot(S)
title('Reconstructed time-domain signal');
xlabel('Samples (n)');
ylabel('Amplitude');
```

Appendix 3: Fast Fourier Transform MATLAB Script

```
% Importing the data
[Y,X] = audioread('SingleNoteClip.wav');
X = Y(1:65536,1); % Selects and creates an array for the amplitudes

my_fft(X)  % Runs my FFT Function
r = abs(ans);  % Magnitude of the complex answers

% Displays frequency-domain graph
plot(r)
title('Frequency spectrum');
xlabel('Frequency bin, k');
ylabel('Amplitude');
xlim([0 length(X)]);



function f = my_fft(X)
N = length(X);
A = nextpow2(N);
X = [zeros((2^A - N)),1);X];  % Zero-padding
f = bitrevorder(X);  % Rearranges input
Stage_Number = log2(N);

k1 = 2;  % Number in each group
k2 = N/2;  % Subdivided DFT/Group Number
k3 = 1;  % Butterfly Number

for Stage = 1:Stage_Number  % Loop for Log2N Stages
    L = 1;  % Calculation number
    for Group = 1:k2  % Loop for each subdivided DFT
        k = 0;
        for Row = 1:k3  % Loop for each butterfly
            g = Row+L-1;  h = g+k3;
            W = complex(cos(-2*pi*k/N),sin(-2*pi*k/N));
            T = f(h)*W;
            f(h) = f(g)-T;  f(g) = f(g)+T;
            k = k+k2;
        end
        L = L+k1;
    end
    k1=k1*2;
    k2=k2/2;
    k3=k3*2;
end
end
```