Exeter Mathematics School

AUDIO TO MIDI CONVERSION

A-LEVEL COMPUTER SCIENCE NEA

April 4, 2019

Oisin Wellesley-Miller

Contents

Analysis . . . 3
  Overview . . . 3
  Research . . . 3
    WAVE and MIDI . . . 4
    DFT and FFT . . . 5
    Existing Solutions . . . 8
  End User . . . 9
  Solution . . . 10
  Objectives . . . 11
  Extensions . . . 12
  Limitations . . . 12
Documented Design . . . 12
  Preliminary Programming and Extra Research . . . 12
  Overview . . . 13
  Data Structures . . . 15
    Wave File . . . 16
    Midi File . . . 17
    Matrix . . . 18
    Fourier . . . 19
    GUIWindow . . . 20
  Algorithms and Functions . . . 21
    Fourier Transform . . . 21
    Median Filter . . . 26
    Blackman-Harris Window . . . 28
    Peak Finding Algorithm . . . 29
  Postprocessing Example Output . . . 29
  Final Overview . . . 31

Technical Solution . . . 32
Reference . . . 32
classes.py . . . 32
gui.py . . . 55
Testing . . . 61
The Matrix Class . . . 62
Addition . . . 62
Multiplication . . . 64
Slicing . . . 67
Concatenation . . . 68
The Wave Class . . . 69
The Fourier Class . . . 70
The Midi Class . . . 74
The GUI Class . . . 74
The Complete Solution . . . 76
Evaluation . . . 77
Objective Completion . . . 77
End User Feedback . . . 78
Further Work . . . 78
Bibliography . . . 80

Analysis

Overview

The Fast Fourier Transform (FFT) is one of the most important mathematical algorithms that exists and its applications have shaped our modern lives. In my project I will be using a Fast Fourier Transform in order to convert a recording of someone playing a piano into a MIDI file of the same recording.
A MIDI (.mid) file is a file format that is commonly used to store and edit compositions of music digitally. It stores which note is being played at which time and for how long, allowing the piece to be built up of the different notes being played at a time. On the other hand, a wave file (.wav) instead stores the magnitude of the wave it can hear through time. When writing and producing music it is often more beneficial to work with midi files as they allow for editing, as the characteristics of individual notes and sounds can be changed. Converting a midi file into a wave file is a simple task, as it only needs to be played through a midi synthesizer and a recording made. However converting in the other direction is a lot more complex, as instead of combining multiple sounds together they instead need to be separated from one another. If this conversion was able to be done easily, then it would be easier for people to edit music without having to use expensive midi instruments to produce the midi files.
In order to take this in and convert it to the note (or notes) being played at a time a Discrete Fourier Transform will have to be performed on the data. This will convert the amplitude over time graph to a graph that shows how much of each frequency is present in a sample. By analysing many samples over the course of the recording my program will be able to determine the note being played and write this to the MIDI file.

Research

My research was split into 3 different topics: The mathematics and algorithms behind the Fourier Transform, research into the exact format of MIDI and wave files and research into existing solutions that perform this conversion.

During this research phase of my project I found information on a wide variety of sites, here is list of many that I used (All links accessed September 2018):

- https://en.wikipedia.org/wiki/MIDI#MIDI_files

- https://en.wikipedia.org/wiki/WAV
- http://www.ccarh.org/courses/253/handout/vlv/
- https://youtu.be/qeb4Dc3gpdo
- https://youtu.be/_7U8hzBNyxk
- https://en.wikipedia.org/wiki/Fourier_transform
- https://www.ams.org/journals/mcom/1965-19-090/S0025-5718-1965-0178586-1/home.html
- https://www.recordingblogs.com/wiki/musical-instrument-digital-interface-midi
- https://www.recordingblogs.com/wiki/orinj-working-with-midi-files
- https://www.avrfreaks.net/forum/how-decode-wav-fileson
- https://www.midi.org/specifications/item/table-1-summary-of-midi-message
- https://stackoverflow.com/questions/13039846/what-do-the-bytes-in-a-wav-file-represent
- https://en.wikipedia.org/wiki/Piano_key_frequencies

WAVE and MIDI

The end result of this project will be a program capable of reading in a .wav file that contains a recording of somebody playing a piece on a piano, process it and then return a .mid file the same piece of piano music. In order to do this I will need to understand exactly how both the .wav and .mid file formats work.

For most applications a wave file is a perfectly adequate way of storing audio data, however its downfalls come in file size and editabilty. Because of the large amount of data they store they are slow to edit as many different changes must be made to the file to represent a small change in how the sound sounds. Midi on the other hand is extremely easy to edit and tweak as this format only stores what note is being played when. The problem I am trying to solve is turning a recording of music in wave format into midi format in order to edit the music.

So first of all what are MIDI and wave files and how are they different? Well, both are methods of digitally representing sound loosely based on the RIFF format but the similarities end there.

Wave (.wav or .wave) files directly store the uncompressed audio wave that they represent so that a computer may easily reconstruct it. They do this by taking a number of samples through time of what a microphone can hear, then storing these values one after another in a file. The most common sampling rate for audio is 44.1KHz (44100 samples a second) as this can be used accurately reconstruct a sound wave to the standards of human hearing, as shown by Nyquist's Theorem as the human hearing usually goes up to around 20KHz. This way of storing audio results in a very faithful representation of the original sound, but at the cost of very large file sizes.

MIDI (.mid) files represent audio in a very different way. They are specialised to store music and instead of representing the sound wave at a particular time they instead store a series of MIDI commands. These commands represent what instrument is playing what note at what time at what volume. This allows for MIDI files to be very easily edited and tweaked as well as allowing them to have incredibly small file sizes, as each command is only a few bytes. However in order to play MIDI files, the machine playing them must have a large number of samples of different instruments playing different notes stored on it so that it knows what to play for each MIDI command. This means that the final quality of the sound being played depends entirely on the device playing the file and not on how the file was recorded as is the case for most other audio formats.

Once the data is read from the wave file and a Fourier Transform performed, the note being played at a particular time must be determined by comparing the found frequencies to the known frequencies of different notes. Once this is done and the start and stop time of the note is known, it can be converted to a MIDI start note command and a MIDI stop note command that can then be written to the MIDI file with the correct delta times. This delta time is a variable length value that stores the time that should be waited before processing the next command in sequence.

DFT and FFT

To begin my research I looked into the Fourier Transform. This is a mathematical transform capable of splitting complicated periodic functions into their components for easy analysis. A very common application of this is to split a wave through time into it's component frequencies, which will be how I plan to use it in this project. There are many other applications of this however, such as in audio editing, image editing, compression and even in our mobile phones. For this project I will need to determine the frequency of a note (or notes) being played at a particular time, so I will need to split the sound wave stored in the .wav file into its frequencies with a Fourier Transform and then analyse these before writing them to the .mid file.

The general Fourier Transform of a function $f(x)$ if usually defined as $\hat{f}(\xi)$, as shown by

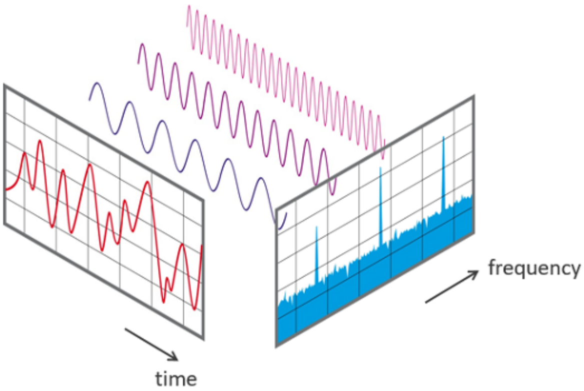

Figure 1: View of a signal through both frequency and time domains [7]

equation 1. In this case x would represent time and ξ represents the frequency of the wave.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) \cdot e^{-2\pi i x \xi} dx \tag{1}$$

However, the above version of the Fourier Transform only applies to continuous functions. In order to perform a Fourier Transform on a series of data, for example the amplitudes stored in a .wav file, instead a Discrete Fourier Transform (DFT) must be used. This will have the same effect as a Fourier Transform but can be performed on a series of equally spaced samples instead of a function. In order to transform a series of N complex numbers $\{x_n\} := x_1, x_2, ..., x_{n-1}$ into their Fourier Transform $\{X_n\} := X_1, X_2, ..., X_{n-1}$ the following function can be used:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-2\pi i k n / N} \tag{2}$$

When actually computed, the DFT is often implemented using matrices to improve efficiency. The samples being transformed are put into a vector and transformation matrix is then constructed, when multiplied together they then result in an output vector of frequencies as shown below in figure 3. When performed in this way the computational complexity of the is $O(n^2)$. What this means is that in order to transform n samples, n^2 calcula'tions must be performed and 2x increase in the number of samples will result in a 4x increase in the number of calculations. This is quite good as far as efficiency goes however it can be improved through the use of a different algorithm for computing the DFT.

$$Let \quad \omega_N = e^{-2\pi i/N}$$

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{(N-1)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega_N & \dots & \omega_N^{(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{(N-1)} & \dots & \omega_N^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{(N-1)} \end{bmatrix} \tag{3}$$

Instead a Fast Fourier Transform (FFT) can be used which will improve the computational complexity to $O(n \log n)$. This FFT algorithm is one of the most important algorithms ever developed and it is ubiquitous in our modern lives. The FFT was first developed in 1805 by Carl Friedrich Gauss in an unpublished manuscript[8], however it was not until 1965 when it was independently rediscovered by Cooley and Tukey[14] that it saw widespread usage. The main idea behind the algorithm is work with a number of samples equal to a power of two. This allows for the Fourier Transform matrix $\underline{\underline{\mathbb{F}}}$ to be split into separate operations for the odd and even indexed samples, and then split again multiple times. The dramatically improves efficiency as many values in the matrices are replaced with zeros meaning that less computations have to be carried out. This process can be seen below:

$$\underline{\hat{X}} = \underline{\underline{\mathbb{F}}}_{2^p} \ \underline{X}$$

$$\underline{\hat{X}} = \begin{bmatrix} I & -D \\ I & -D \end{bmatrix} \begin{bmatrix} F_{2^{p-1}} & 0 \\ 0 & F_{2^{p-1}} \end{bmatrix} \begin{bmatrix} \underline{X}_{even} \\ \underline{X}_{odd} \end{bmatrix}$$

$$Where \quad D = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & \omega_N & 0 & \dots & \vdots \\ 0 & 0 & \omega_N^2 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & \omega_N^{2^p-1} \end{bmatrix}$$

$$F_{2^p} = \begin{bmatrix} F_{2^{p-1}} & 0 \\ 0 & F_{2^{p-1}} \end{bmatrix}$$

With this FFT algorithm each time F_{2^p} is broken down, the efficiency increase by a factor of two, as half of the resulting matrix becomes zeros.

Interestingly even if the number of samples is not exactly a power of 2, it is still dramatically more efficient to pad the samples vector with zeros until a power of two is reached and then perform a FFT than it would be to perform a DFT.

Existing Solutions

Bear File Converter

https://www.ofoct.com/audio-converter/convert-wav-or-mp3-ogg-aac-wma-to-midi.html

The first solution I found was this website, which allows the user to convert one of a variety (WAV, MP3, OGG, AAC, WMA) of file formats into a MIDI file. Most of this website is covered in advertisements, but a screenshot of the actual conversion UI can be seen below.

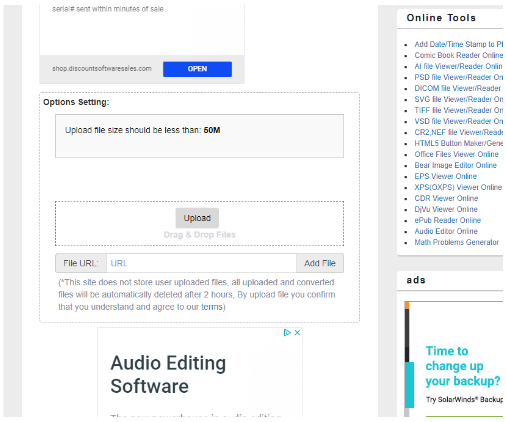

Figure 2: The Bear File Converter

The website is very simple to use once the options are found on the page, there is an input box to add the URL of web-hosted file or to upload a local file. Once a file has been added options will then appear to add additional files for a batch conversion, convert all added files, to download all converted files or to download an individual file.

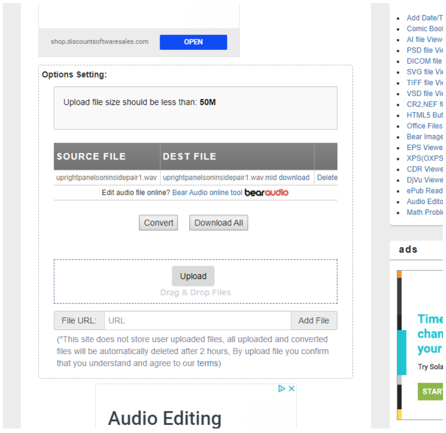

Figure 3: Bear File Converter Options

Once the converted MIDI files are downloaded they can then be easily played by a MIDI sequencer. This site has a very easy to use interface (once you find it amongst the ads) that makes it very simple to convert a file, however the quality of the final MIDI file varies significantly with the song being converted. Some pieces convert almost flawlessly, with only a few minor glitches, whereas the results of other sound like someone randomly bashing the keys of a piano. From the testing I have done it seems the more chords being played at once and the faster the tempo of the music the more likely it is to "fail" in the conversion, though overall this solution is very robust.

I was unable to test any other existing solutions during my research as the only other solutions I could find were part of expensive software packages (such as Ableton[1], Widisoft[3], Wavesum[2])

End User

In order to determine the exact features that that my end user will find important I conducted an interview with them, I have included the questions and their responses below:

- Question: What are you going to be using this application for?
  "I need this program to help me when I am writing music, I like to try out different melodies that I hear on TV or the internet ect to experiment. The problem is my pitch is not very good so I struggle to work out what notes are playing, hopefully this program should help me with this. I'd also like to be able to edit and see the code myself, I don't want something that runs on a server somewhere I'd like to know exactly what is happening to the music that goes through it."

- Question: Are there any particular features that will be important?
  "I need the ability to run this on a multitude of devices, such as my desktop that runs Windows and my laptop that runs Linux. I also would like it to be able to identify different chords, but this isn't that important for me."
- Question: How important is the speed at which it works?
  "I'd like it to be relatively fast, but I don't mind too much if it isn't as I can be working on other ideas when I wait for it to complete."
- Question: Is a GUI an important feature?
  "It would be nice to see but not essential, I use some other command line tools so I would be used to working without a GUI."
- Question: How important is the accuracy?
  "I think getting the different pitches correct is important, I don't mind about the timings being exactly correct as I can more easily edit these from the midi file and what I can hear from the original. Obviously if it get's it completely correct that would be best."

From this I have determined that the end user for this project is a musician who needs a simple solution to music transcription, current .wav to .mid are either part of expensive software packages, are sketchy links on file hosting sites or require uploading the file to a 3rd party server. My client wants a solution that will combat all three of these issues: It must be available to them for free, and not require the installation of an .exe from a potentially unsafe source and be use-able without uploading their new music to a 3rd party. As part of their requirements they need the program to be able to run on both Windows and Linux, be easy to use and quite quick to process.

Solution

My solution will be a GUI based program capable of performing the file conversion of a Wave file to a MIDI file through the use of a Fourier Transform. I will read in a Wave file extract the sampling rate and sample data from it and store it as an object within my program. The program will then move through the file and perform multiple Fourier Transforms on different slices to build up a model of which notes are probably being played at each time. This will be calculated through the use of custom matrix classes. Once this is done, the resulting probabilities will be converted into a MIDI object that will hold the MIDI commands needed to represent the notes. Finally this will be written to a file and a success message displayed to the user.

The GUI will look something like the below image, but this is just a rough mock-up of the UI elements that are needed.

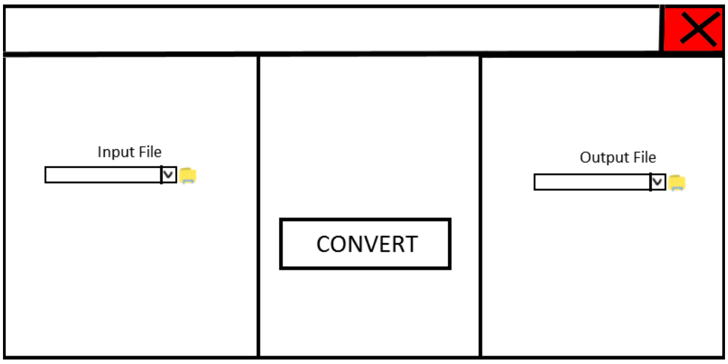

Figure 4: GUI Mock-up

Objectives

1. The program must be able to read the samples from a wave file into an object that represents the samples and information of the file, providing the input file is under a length of 3 minutes.
2. The program should be able to read the amplitudes of the wave object to approximate volume.
3. The program must be able to perform the matrix operations:
   (a) Addition
   (b) Multiplication
   (c) Slicing
   (d) Concatenation
4. The program must be able to correctly write the results to a midi file.
5. The user must be able to select a wave file to convert using a GUI.
6. The user must be able to select a wave file to convert without using a GUI.
7. The user must receive conformation that the conversion has taken place successfully, or that an error of some kind has occured.

8. During conversion, the user should have visual feedback in the form of a timer or progress bar to ensure the program is functioning.

9. The program must be able to convert files that have only one note played at a time.

Extensions

As an extension to the above basic functionality, some of the following features could be added to improve the program:

1. The program should be able to correctly pick up on the velocity with which notes are played, relative to each other.

2. The program could be able to convert files that have multiple notes being played at a time.

3. The user could be able to queue the conversion on multiple files.

Limitations

The limitations of this program will likely be in the conversion of wave files that are not of sufficient quality or include other instruments than a piano. These will not be able to be converted as the frequencies they contain will not correctly map to piano notes. The program may also struggle with the conversion of multiple notes being played at the same time in quick succession, as there may not be enough data to work out if a sound is more than one note or not.

DOCUMENTED DESIGN

Preliminary Programming and Extra Research

Before I started work on my final solution I started by making some different test versions of my converter. For this I used the signal processing functions from Numpy instead of implementing them myself a this meant I could easily switch in and out different steps and observe the changes these made to the output without having to re-write each one each time. Before I could start coding anything I had to do some further research into methods that had been tried before, I began by reading through several papers on this topic [9] [10] [11] [12]. I found that there were several different approaches that I could try but there wasn't a know perfect method, so anything that I produce couldn't be perfect in operation. A few methods used auto-correlation whereas others instead just used the FFT and some post-processing so I decided I would need to try out both and see what I could get working. To begin with I tried to determine the pitch at a time using a process called auto-correlation, this involved taking a sample and offsetting it with itself and computing the Pearson correlation coefficient and then storing them into a result list. This process can be sped up by using Fourier transforms by utilising the Weiner-Khinchin Theorem[13]. After implementing this I found that it was very slow and did not give results that were as clear as just using a FFT. Although theoretically this use of autocorrelation should be able to cope with chords better I did not find it as easy to extract the notes from its results, so I ended up not using this method in favour of a pure FFT approach.

Overall I believe that I should be able to match then effectiveness of the Bear file converter, however exceeding its accuracy and speed will be a very difficult task due to the complexity involved in the project I have chosen. There is no known method for transcribing files will 100% accuracy in a reasonable compute time, but my approximation should be enough to meet the needs of my user and therefore enough for the scope of this project.

Overview

This program will be written in Python 3.7 it provides an easy way of dealing with files and binary, as well as easy rapid prototyping. Although this may not produce a result that is as efficient to run as say Java, it will a lot easier to produce and extend. Additionally, efficiency is not an absolute priority for this project as evidenced by the answers gained from the questions in the Analysis section.

The main algorithm that my program is using is the FFT, the mathematics of which which I have briefly explained in my analysis section. This is a recursive algorithm where each step reduces the computational complexity by a factor of two. When applied to a signal through time, it will transform it into a representation of the different frequencies present as mentioned in the analysis. Once a user has selected a file to convert to midi using the GUI, the file will be read in and one track will be separated and stored into a matrix. This matrix will then be iterated over with a sliding window to determine the note at each position. This data will then be turned into midi events, sorted, and written to a midi file. This process is shown in the image overleaf (Figure 5).

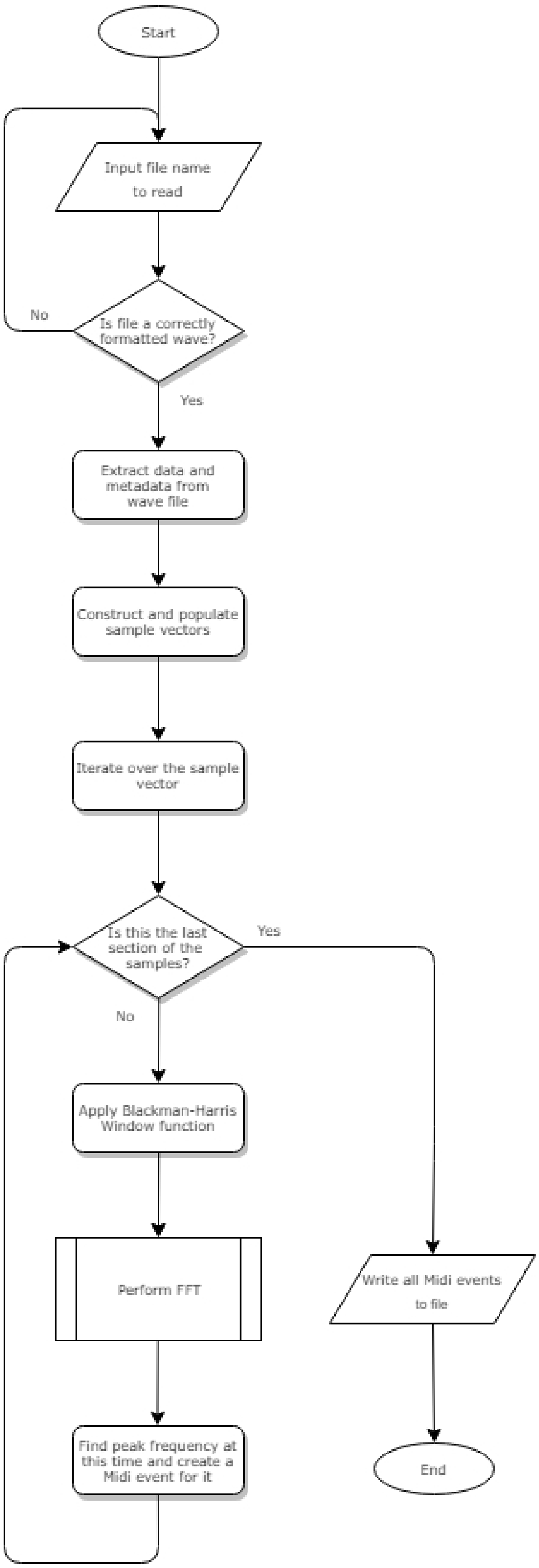

Figure 5: Program Flow

Data Structures

I have used an object orientated design structure for this project as it easily allows me to see how the functionality of different sections should interact with each other. I have tried to ensure each class completely encapsulates all of the data it need to work on and that they only interact with each other in ways that I have designed through the used of getter and setter methods. I have also used the convention of underscores to denote the attributes which should remain private to each class in order to ensure any other programmers can easily understand the structure of my code.

This section will now briefly discuss the purpose of each class and the methods that they can perform, Figure 6 shows the inheritance relationships between the different classes, along with their methods and attributes.

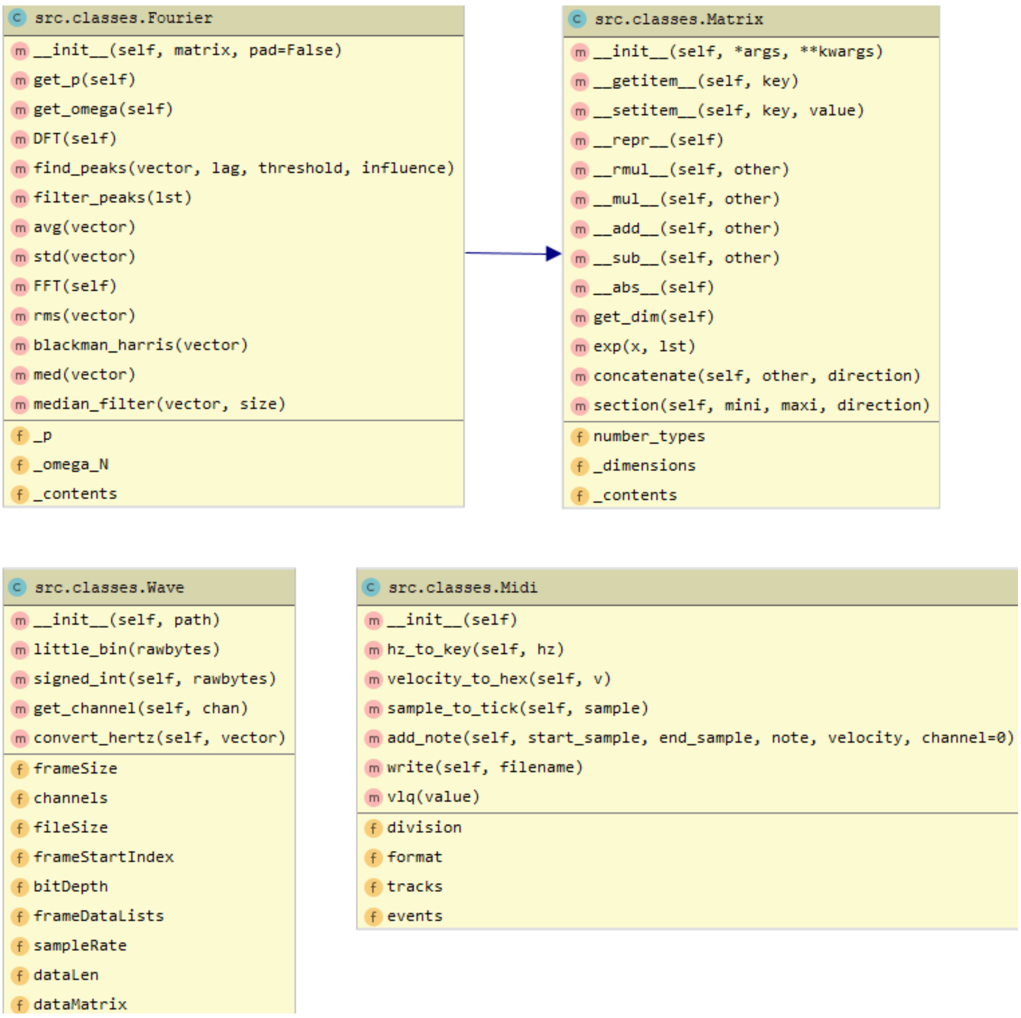

Figure 6: Hierarchy Chart

Wave File

The wave file class deals with the initial input of a .wav file to the program. It reads in the file as binary and carefully parses the file's metadata to ensure it follows a format that the program can understand. Once the relevant information such as sample rate, filesize, number of channels and bit depth have been extracted, each different channel of audio is read and transferred into a separate column within a matrix. In order to understand the exact layout of a wave file I used a reference page [6] from an old Stanford University website. The main information I needed was the layout of the file's header, as this would contain the metadata about the file that I would need to know in order to read in the samples. The wave file format is a subset of the RIFF file format and so consists of a file header followed by a number of data chunks, depending on the amount of data stored. This makes the job of reading the file very easy, as the needed values can be found in set locations defined in the header chunk.

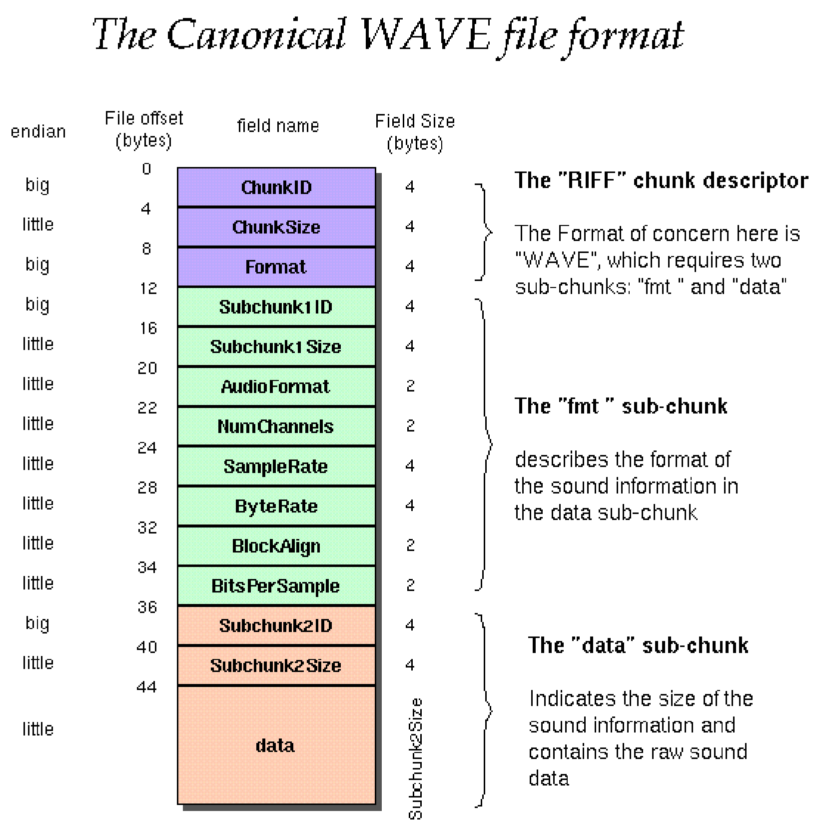

Figure 7: Wave File Format

When the data is needed for a conversion, the specified audio channel is then transferred to its own matrix and sent for processing. This class is quite simple in terms of functionality as it need to only read in wave files, so the only complex methods it has are to do with the conversion of the little-endian hex to signed integers.

Function Name	**Inputs**	**Description**
__init__	A file path	The Wave's constructor, loads the wave file from the specified path into an object. Each audio channel is stored into a column in a Matrix.
little_bin	Some Bytes	Converts the provided bytes into an unsigned 32 bit integer, using little endian form.
signed_int	Some Bytes	This converts the bytes into a signed integer, using the bit depth of the file.
get_channel	A channel number	This gets a specific channel of audio, from the Matrix where they are stored.
convert_hertz	A Matrix	This converts a Matrix from an FFT output into the corresponding values in Hz.

Midi File

The midi file class is used to collect midi events and write them to a specified output file. It can be configured to write with different numbers of tracks and time signatures, although for this project they are set to be 1 and 96 respectively. As the conversion progresses the class will accumulate a list of midi events which it will create from a given start time, length and pitch. Once the process is finished, the events must be prepared for writing by converting their start times and lengths into note on and note off events. Since a midi file is read event by event, each note on/off event also need to be given a delta time so the interpreter/synthesizer knows how long to wait before progressing to the next event. In a midi file these delta times are stored as VLQs (Variable Length Quantity) as opposed to a traditional fixed length binary number. This means that they can use as many bytes as needed to represent an arbitrarily large number without wasting extra bits for leading 0s when storing smaller numbers. However, because each number is represented with a different number of bits, this makes reading in midi files a difficult task as special care must be taken to separate the different values from each other. Luckily for this project only the ability to write midi files is required, so this simplifies things considerably.

The main functions of this class are type conversion, for example the Hz to midi note formula, sample to tick and variable length quantities. The Hz to midi note conversion makes use of the below formula in order to map a frequency in Hertz to on of the 127 notes that the midi file format supports.

$$\text{Midi note} = 69 + 12 * \log_2(Hz/440) \tag{4}$$

Function Name	**Inputs**	**Description**
__init__	None	Constructs a blank Midi object.
hz_to_key	A value in Hz	This converts a frequency in Hz to the corresponding midi key.
velocity_to_hex	A velocity value	This converts a notes strength to a midi velocity in hex, this is how loud the note is.
sample_to_tick	A sample number	Converts a sample number to a time in midi ticks.
add_note	A start sample, end sample, note, velocity and channel	Adds a note to the list of notes to be written, as a midi event.
write	A filename	Sorts the different midi events by their time and writes them to an output file.
vlq	An integer value	Converts a value to its VLQ equivalent.

Matrix

The matrix class is a very important data structure as it is used to perform operations required to do the Fast Fourier Transform. Matrices are used to store the samples from the wave file, the intermediary FFT products as well as the final results so they are critical to this project. I have implemented most standard operations that can be done with matrices such as addition, multiplication and subtraction, as well as things such as getting and setting data a specific positions. I have note-ably not implemented matrix inversion however, as this was not needed for the applications within this project I would be using the matrices for.

There are also some other functions within the matrix class, such as the ability to get a specific vertical or horizontal slice from a matrix using the section method, or to concatenate two matrices into one which are used throughout the program for ease of access.

Function Name	Inputs	Description
__init__	A dimension to fill, or a list of values	The Matrix's constructor, handles forming new Matrix instances and ensuring that the type of data stored in them is valid.
__getitem__	An index	Gets the item from the Matrix at a specified index.
__setitem__	An index and an item	Sets the specified index of the Matrix to an item
__rmul__	Another object	This handles multiplication when the object on the right does not have a method to multiply with a Matrix. This is used for multiplication with integers, floats and complex.
__mul__	Another Matrix	This handles all other types of multiplication, mainly multiplying together two Matrices.
__add__	Another Matrix	This adds another Matrix to the Matrix, item by item.
__sub__	Another Matrix	This subtracts another Matrix from the Matrix, item by item.
__abs__	None	This takes the modulus of each item in the Matrix.
get_dim	None	This returns the dimensions of the Matrix.
concatenate	Another Matrix and a direction	This concatenates another Matrix onto the Matrix, either horizontally or vertically.
section	A start, stop and direction	This takes a slice from the Matrix between the start and stop values in the given direction.

Fourier

This class inherits from matrix and is used to store the specific data that the transform is applied to just before it is. No methods from the matrix class were overridden, however a few new ones were added in the form of the DFT and FFT functions, Blackman-Harris window and median filter functions. Each of these are needed for a step in the final conversion process, so whilst not all are strictly Fourier related they are close enough to be put into this class instead of requiring the creation of an additional class.

Function Name	**Inputs**	**Description**
__init__	A Matrix	Constructs a Fourier object from a Matrix object, also inherits from Matrix.
DFT	None	Performs the DFT over the contents of itself.
find_peaks	A Matrix	Finds the indexes of any peaks in the data of the Matrix and whether they are concave or convex upwards.
filter_peaks	A Matrix	Filters out some peaks from the data if they seem irregular.
avg	A Matrix	Finds the mean average of the data.
std	A Matrix	Finds the standard deviation of the data
FFT	None	Performs a recursive FFT on the Fourier object.
rms	A Matrix	Finds the root mean squared average of the data.
blackman_harris	A Matrix	Performs a Blackman-Harris window on the data.
med	A Matrix	Finds the median average of the data.
median_filter	A Matrix and a size	Perfoms a median filter over the contents of the Matrix.

GUIWindow

This class creates and handles the GUI for the project, using the PyQt5 library, it inherits from a QtWidget which allows it to easily create and manage a layout to represent itself. It handles the orchestration of the rest of the program, as well as allowing the user to select an input file and displaying a progress bar so they can see how close the program is to completion.

This class has methods that are called when a user clicks buttons on the GUI, as GUIs are event driven. For example when the browse button is clicked a new file browser window is opened and the path of the file the user selects using it is returned. It also has a "main" method that controls the creation and use of the other classes. The structure of this main method follows the program flow chart seen in Figure 5. The GUI also has a progress bar to show how far along the conversion is and the reassure the user that the program has not crashed. In order to achieve this, I have used threading to allow the GUI to run on one thread and the actual conversion to run on a different thread. This means that these two parts of the program can run concurrently in parallel, with the progress bar updating when each note has been written.

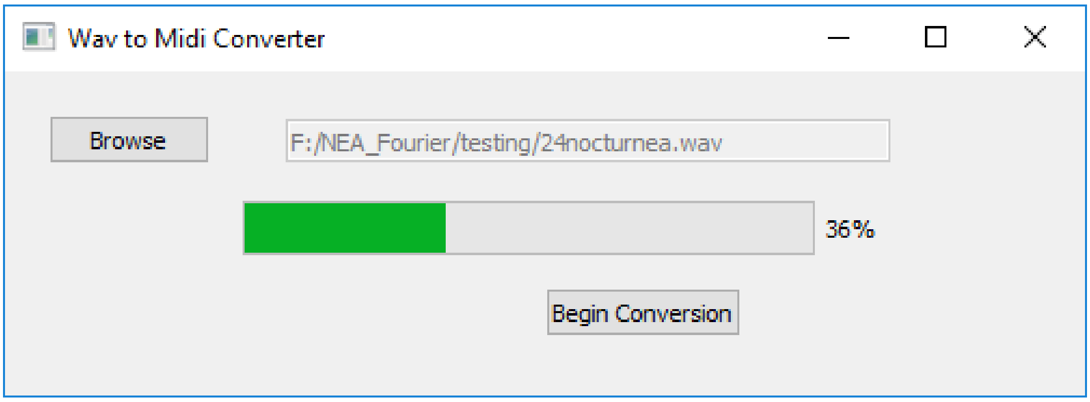

Figure 8: The GUI for Running the Converter

Function Name	**Inputs**	**Description**
__init__	None	This creates the GUI window.
get_files	None	This opens a file explorer window and returns the path of the file they select.
begin	None	This begins the process of converting the file once the button has been clicked and displays a predicted time of completion. The actual conversion is handled on a different thread to the GUI, to allow it access to more RAM.
main	A path	This handles the actual conversion itself, the different Wave, Midi and Fourier objects are all created and managed here.

Algorithms and Functions

Fourier Transform

As mentioned, the backbone of this project relies on the Fast Fourier Transform. In order to make use of this I have implemented the Cooley-Tukey algorithm that I described in my analysis section.The flowchart for this algorithm can be seen in Figure 9 and the psuedocode for this can be seen as Algorithm 2, it makes use of recursion in order to simplify the problem and turn it into something that is much easier to compute. I have implemented this in my code with the use of my custom Matrix classes.

One key point to understand is that the DFT simply represents the method of applying the Fourier Transform shown in Equation 1 to a discrete set of data and is simply a mathematical function, whereas an FFT is a family of algorithms that speed up this process significantly. In this project I have used the Cooley-Tukey algorithm as my FFT and how this works is shown both in the flowchart in Figure 9 and in Algorithm 2. Algorithm 1 shows how the formula for the DFT is applied to a sample containing vector using matrix multiplication, this is taken from the form shown in Equation 3.

Algorithm 1 Discrete Fourier Transform

procedure DFT(vector)
    $N \leftarrow$ length of *vector*
    $omegaN \leftarrow e^{-2\pi i/N}$
    $matrixDFT \leftarrow$ blank N by N sized matrix
    for $x \leftarrow 0, 1, 2, .., (N-1)$ **do**
        for $y \leftarrow 0, 1, 2, .., (N-1)$ **do**
            $matrixDFT[y][x] \leftarrow omegaN^{x*y}$
    return $matrixDFT * vector$

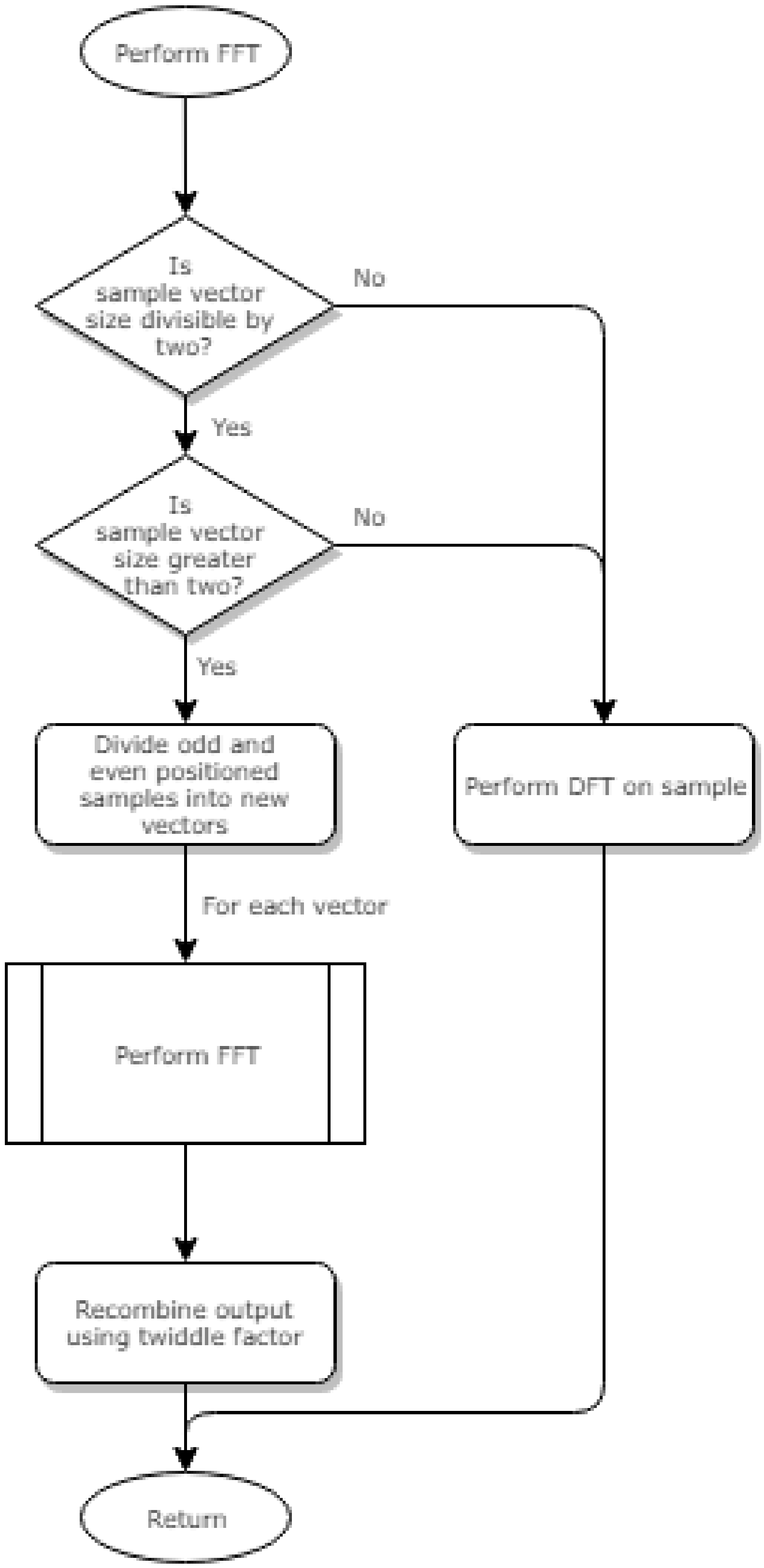

Figure 9: FFT Flowchart

Algorithm 2 Fast Fourier Transform

```
procedure FFT(vector)
    N ← length of vector
    if N <= 2 then return DFT(vector)
    else
        even ← even positioned values of vector
        odd ← even positioned values of vector
        even ← FFT(even)
        odd ← FFT(odd)
        powers ← Matrix(0, 1, 2, .., N-1)
        factor ← −2πi/N to the power of each element of powers
        firstHalf ← blank N / 2 by 1 sized matrix
        secondHalf ← blank N / 2 by 1 sized matrix
        for i ← 0,1,2,..,N/2 do
            firstHalf[i] ← even[i] + odd[i] ∗ factor[i]
            secondHalf[i] ← even[i] + odd[i] ∗ factor[N/2 + i]
        return concatenation of secondHalf onto firstHalf
```

To better understand this, an example of the output of this algorithm can be seen below. Figure 10 shows the input to the function in green, this comprises to two sine waves of different frequencies added together. When passed into the Fourier transform it is decomposed into the second function shown in Figure 11. As you can see the Fourier transform has produced two peaks, as the wave it was passed was composed of two different frequencies. The location of these peaks do not exactly match the frequencies of the original wave however, as it depends on the sampling rate of the audio where the peaks show up. In order to get the actual frequency of the wave (in Hz) out, the below formula (Formula 5) must be used for conversion:

$$\textit{frequency in Hz} = \frac{\textit{samplerate}}{\textit{size of sample}} * \frac{\textit{bin number}}{2} \tag{5}$$

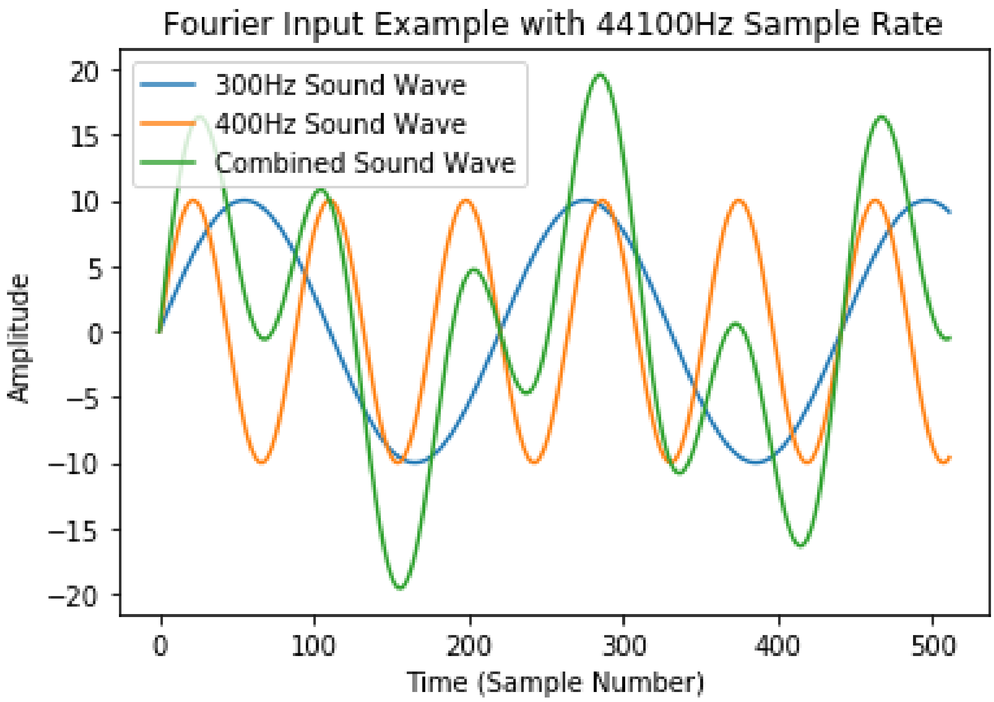

Figure 10: Input Wave

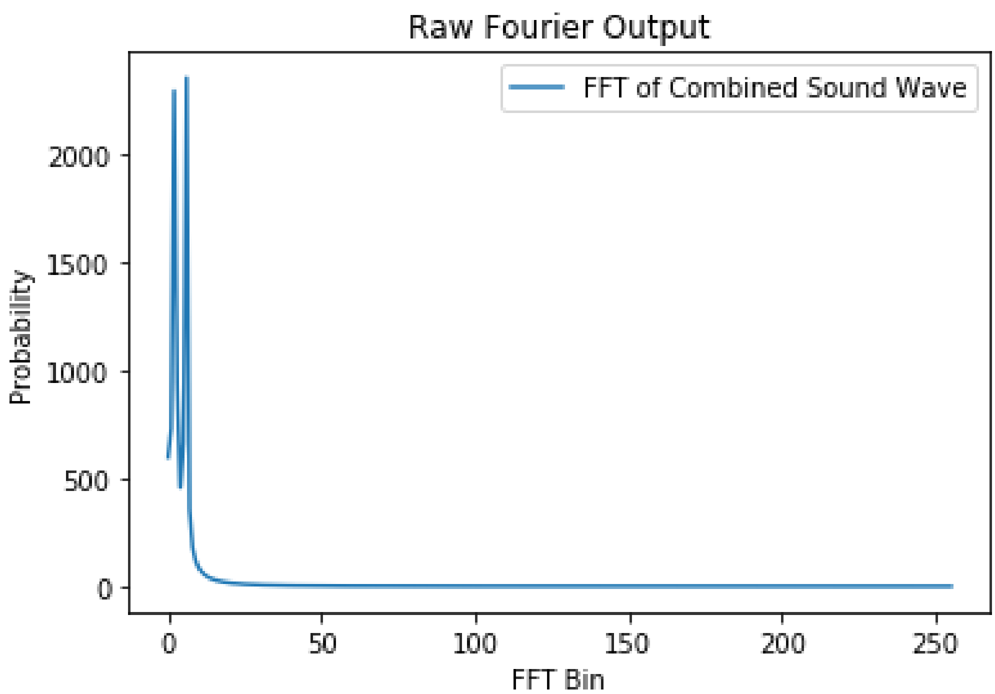

Figure 11: FFT Output

Median Filter

A median filter is used on the raw results from the FFT in order to help smooth out any smaller peaks due to signal noise, this then helps with identifying the positions of peak frequencies as there is a lesser chance of false positives. A median filter works by sliding a odd-length sized window across the data and replacing the middle value with the median of the numbers in the window. The psuedocode for this can be seen below in Algorithm 3.

Algorithm 3 Median Filter

procedure MEDIAN FILTER(vector, size)
  $N \leftarrow$ length of *vector*
  output $\leftarrow$ blank N by 1 sized matrix
  $output[0][0] \leftarrow vector[0][0]$
  $output[N-1][0] \leftarrow vector[N-1][0]$
  $end \leftarrow size/2$
  for $i \leftarrow 0, 1, .., (N - end) - size$ **do**
    window $\leftarrow$ *vector* positions i to $i + size/2$
    $output[end + i][0] \leftarrow$ median of *vector*
  return *output*

As an example, if passed in the list [0, 3, 2, 6, 4, 0, 12, 21, 8, 0, 0, 22, 24, 26, 0, 30, 32, 0, 54, 38] with a window size of 3, then the output would be [0, 2, 3, 4, 4, 4, 12, 12, 8, 0, 0, 22, 24, 24, 26, 30, 0, 0, 0, 38]. The smoothing that this provides can be seen in the plots overleaf.

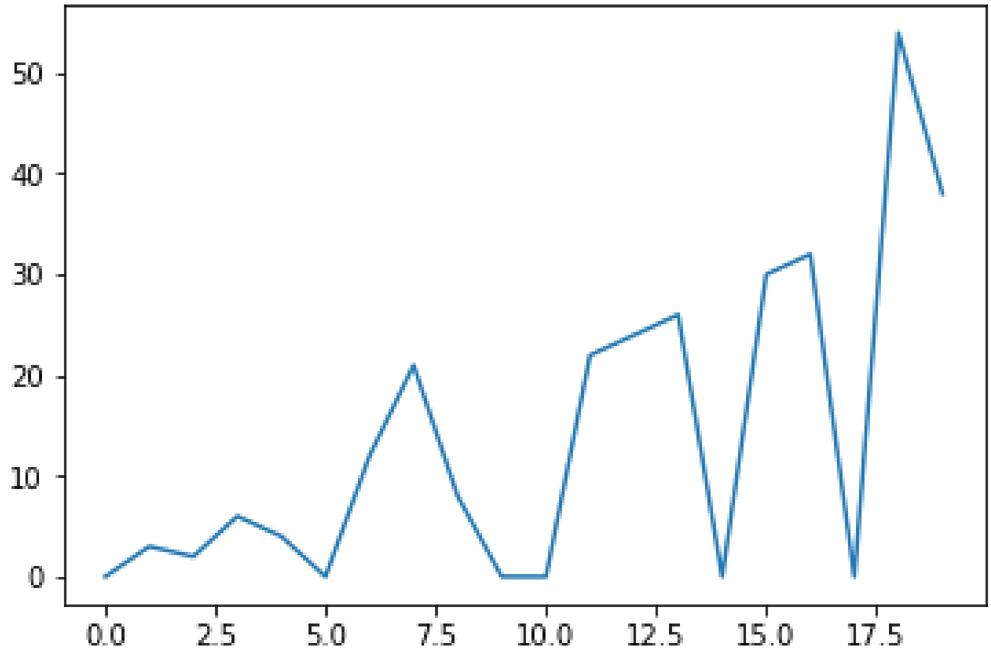

Figure 12: Input List of Random Numbers

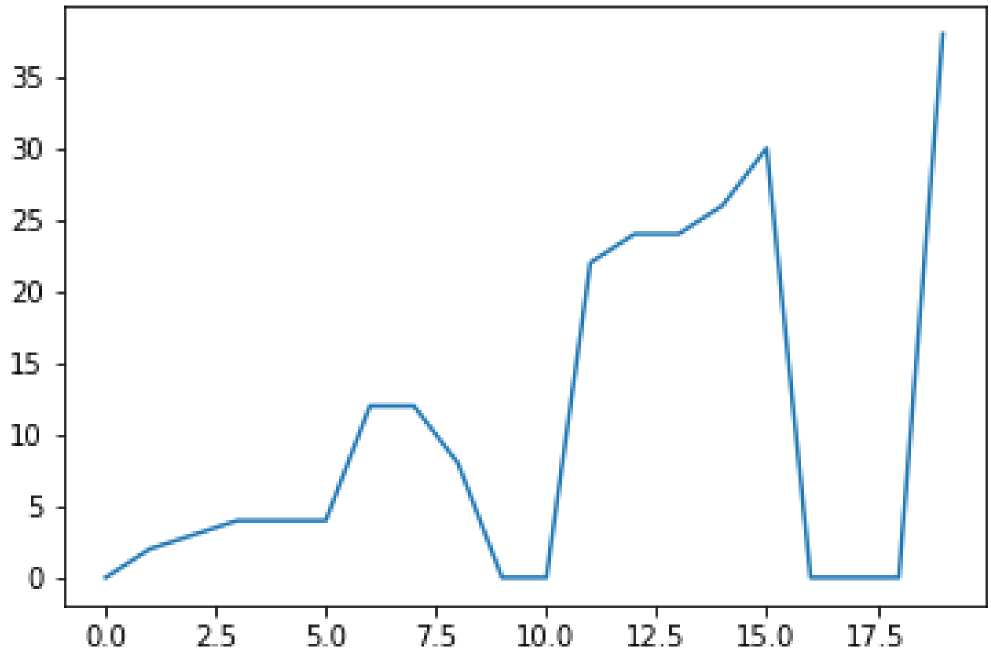

Figure 13: Output List After Median Filter

Blackman-Harris Window

Windowing is a signal processing technique whereby only specific sections of data are processed at a time. For example in the median function algorithm a square window is used to look through the input vector, this involves simply looking through and taking the section of data as-is to be processed. A square window is the most simple kind of window as no further processing is required to make one, it is also sufficient for most applications such as finding a median. However with an FFT the results can be improved by using a more complicated windowing function such as the Blackman-Harris as it puts more emphasis on data towards the middle of the window. Using the Blackman-Harris the final conversion does noticeably improve in quality as it is more easily able to distinguish low-frequency notes from signal noise.

The formula for this window function can be seen below in Figure 14, each value in the raw window is multiplied by the result of this function, where N is its position in the window. The result of applying this function to a window of 1s can also be seen in Figure 15.

$$w[n] = a_0 - a_1 \cos\left(\frac{2\pi n}{N}\right) + a_2 \cos\left(\frac{4\pi n}{N}\right) - a_3 \cos\left(\frac{6\pi n}{N}\right)$$

$$a_0 = 0.35875; \quad a_1 = 0.48829; \quad a_2 = 0.14128; \quad a_3 = 0.01168.$$

Figure 14: Blackman-Harris Function Formula

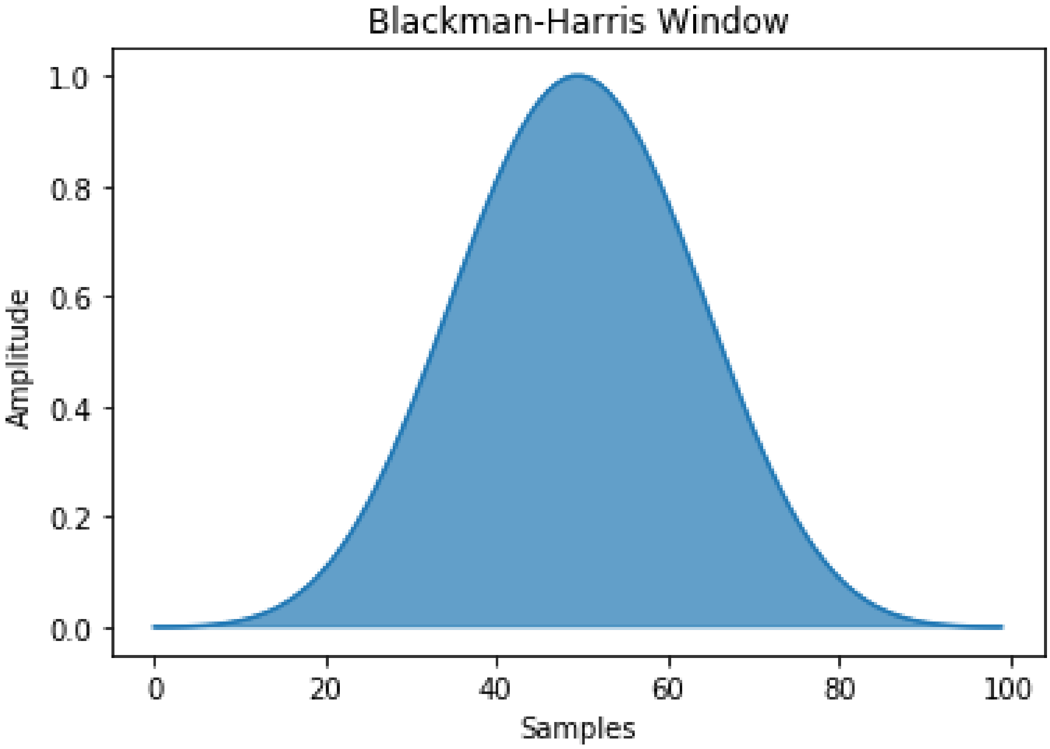

Figure 15: Blackman-Harris Plot

Peak Finding Algorithm

In order to find all of the peaks in my FFT output I've implemented a peak-finding algorithm. This algorithm will look through the values presented and signal that there is a peak in the data when the new value it sees is more than a certain threshold away from the data it has already seen, with a prioritisation of more recent data points. This algorithm is quite robust and allows for fine-tuning of the different threshold values so I was able to set it up to work best with the data I would be feeding it. Interestingly, the origin of this algorithm was not a scientific paper or something similar, but instead an answer on StackOverflow and can be found here *https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362*.

Postprocessing Example Output

When these extra algorithms are used in conjunction with the FFT, a more clear result can be obtained. For example when provided with the same input wave as shown in Figure 10 on page 19, the difference in outputs can be seen below as 16. The Blackman-Harris window

Algorithm 4 Peak Finding Algorithm

procedure FIND PEAKS(vector, lag, threshold, influence)
  $N \leftarrow$ length of *vector*
  signals $\leftarrow$ blank N by 1 matrix
  filteredY $\leftarrow$ *vector*
  avgFilter $\leftarrow$ blank N by 1 matrix
  stdFilter $\leftarrow$ blank N by 1 matrix
  avgFilter$[lag-1] \leftarrow$ average of the first *lag-1* items in *vector*
  stdFilter$[lag-1] \leftarrow$ standard deviation of the first *lag-1* items in *vector*
  for $i \leftarrow (lag+1), .., (N-1)$ **do**
    if $|(vector[i][0] - avgFilter[i-1][0])| > threshold * stdFilter[i-1][0]$ **then**
      if $vector[i][0] > avgFilter[i-1][0]$ **then**
        $signals[i][0] \leftarrow 1$
      else
        $signals[i][0] \leftarrow$ -1
      $filteredY[i][0] \leftarrow influence * vector[i][0] + (1 - influence) * filteredY[i-1][0]$
    else
      $signals[i][0] \leftarrow 0$
      $filteredY[i][0] \leftarrow vector[i][0]$
    $avgFilter[i][0] \leftarrow$ the average of *filteredY* from i-lag to i
    $stdFilter[i][0] \leftarrow$ the standard deviation of *filteredY* from i-lag to i
  return *signals*

removes some of the smoothing of the raw Fourier output, allowing a computer to more easily determine the peaks in the data.

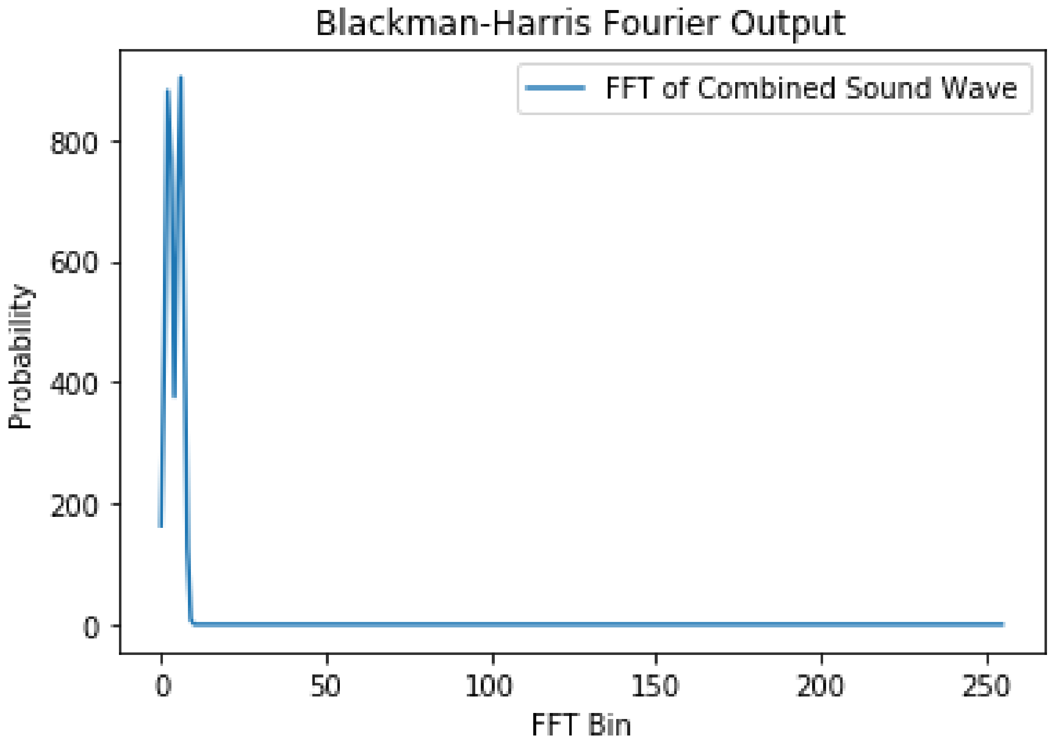

Figure 16: FFT Output with Blackman-Harris

In order to then determine the frequency from this I use the peak finding algorithm. In order to try and identify any chords in the recording, I must look for when there are two different peaks that are not multiples of each other. This is again hard to achieve as algorithmically determining the difference between a chord and noise in the FFT result is very difficult.

Final Overview

Using all of the above algorithms my program should be able to successfully complete its task of converting a wave file to a midi file. Once the samples have been read in, they will first be iterated over to determine the boundaries between where there is and is not a note. Then each section will be put through a Blackman-Harris window to better represent the frequencies that will be heard, before finally being Fast Fourier Transformed. A median filter will then be used to smooth out the results and then a peak finding algorithm used to determine the peaks. The position of these will finally be converted into a midi note and then written as a midi event to a file.

Technical Solution

Reference

The below table shows where key features of the program are located in the program. All line numbers refer to the first file *classes.py*, as *gui.py* simply holds the code to run the classes and to display the GUI.

Feature	Line Number	Page Number
Matrix Multiplication	137	37
Wave File Reading	291	42
Discrete Fourier Transform	423	46
Peak Finding Algorithm	433	46
Fast Fourier Transform	486	48
Blackman-Harris Window	512	49
Median Filter	534	49
Midi File Writing	581	51

classes.py

```python
1 import cmath
2 import math
3 import pickle
4 import time
5
6
7 class Matrix:
8     """A  n*m matrix class, can be constructed from a list of objects or a 2d list of objects
9         e.g.
10        a = Matrix([[1,2], [3,4]])
11        a = Matrix(m=10, n=10)
12        Methods:
```

```
            __init__
            __rmul__
            __add__
            __sub__
    """

    def __init__(self, *args, **kwargs):
        # Setup initial variables
        self._contents = []
        self._dimensions = [0, 0]
        self.number_types = (int, float, complex)
        if len(args) == 0 and len(kwargs) == 2:
            # Construct from given m by n dimensions
            if kwargs.keys() == {"m": 0, "n": 0}.keys():
                if isinstance(kwargs["m"], int) and isinstance(kwargs["n"], int):
                    if kwargs["m"] > 0 and kwargs["n"] > 0:
                        self._dimensions = [kwargs["m"], kwargs["n"]]
                        self._contents = [[0 for x in range(kwargs["n"])] for y in
                                          range(kwargs["m"])]
                    else:
                        raise TypeError("Matrix dimension cannot be less than 0")

                else:
                    raise TypeError(f"""Invalid type for dimensions:
                                    [{type(kwargs['m'])}, {type(kwargs['n'])}]""")

            else:
                raise TypeError(f"""Invalid kwargs {kwargs} must be 'm' and 'n'""")

        elif len(args) == 1 and len(kwargs) == 0:
```

```
            # Construct from values
            if isinstance(args[0], list):
                if len(args[0]) > 0:
                    # Can construct from list of objects OR
                    # List of lists of objects
                    if isinstance(args[0][0], list):
                        # If given a list of lists, then it must be valid
                        n = len(args[0][0])
                        for x in range(len(args[0])):
                            if not isinstance(args[0][x], list):
                                raise TypeError(
                                    "Invalid values for Matrix, must be only of type list")
                            for y in args[0][x]:
                                if isinstance(y, list):
                                    raise TypeError(
                                        "Invalid values for 2D Matrix, must not be list")
                                if len(args[0][x]) != n:
                                    raise TypeError(
                                        "Invalid values for 2D Matrix, must not of equal width")
                        # At this point, valid list of lists so matrix is constructed
                        self._contents = args[0]
                        self._dimensions = [len(args[0]), len(args[0][0])]

                    else:
                        # At this point a 1D list is detected
                        for x in args[0]:
                            if isinstance(x, list):
                                raise TypeError(
                                    "Invalid values for Matrix, must be only of type: list")
```

```

                        self._dimensions = [1, len(args[0])]
                        self._contents = [list(args[0])]
                        if all(issubclass(type(x), type(self)) or issubclass(type(self), type(x)) for x in args[0]):
                            self._contents = self._contents[0]
                        # And constructed

            elif issubclass(type(args[0]), type(self)) or issubclass(type(self), type(args[0])):
                # This if for creating a copy of a matrix, or matrix subclass
                self._contents = args[0]._contents
                self._dimensions = args[0]._dimensions
            else:
                raise TypeError(f"Invalid type for Matrix: {type(args[0])}, must be list or matrix")

        else:
            # Don't construct, incorrect information given
            raise TypeError(f"""Invalid input length for Matrix: {type(args)},
                                must be exactly 1 list""")

    def __getitem__(self, key):
        # Gets the item at a specified index
        if isinstance(key, int):
            return self._contents[key]
        elif isinstance(key, slice):
            if self.get_dim()[1] == 1:
                data = self._contents[key]
                return Matrix(data)
            else:
```

```python
                raise KeyError("Slice not supported for matrices, only vectors")
        else:
            raise KeyError

    def __setitem__(self, key, value):
        # Sets the item at a specified index
        if isinstance(key, int):
            if self._dimensions[0] >= key:
                self._contents[key] = value
            else:
                raise KeyError
        else:
            raise KeyError

    def __repr__(self):
        return str(self._contents)

    def __rmul__(self, other):
        # This should handle scalar multiplication only
        if isinstance(other, list):
            if len(other) == 1:
                other = other[0]
        if isinstance(other, self.number_types):
            result_matrix = Matrix(m=self._dimensions[0], n=self._dimensions[1])
            for y in range(len(self._contents)):
                if isinstance(self[0], list):
                    for x in range(len(self[y])):
                        result_matrix[y][x] = self[y][x] * other
                else:
                    result_matrix[y] = self[y] * other

        elif issubclass(type(other), type(self)) or issubclass(type(self), type(other)):
```

```
            # Matrix multiplication should be handled by mul not rmul,
            #  if being found here then an error has occurred
            raise NotImplementedError("Matrix multiplication should be handled by mul")
        else:
            raise TypeError
        return result_matrix

    def __mul__(self, other):
        if issubclass(type(other), type(self)) or issubclass(type(self), type(other)):
            if self._dimensions[1] != other.get_dim()[0]:
                raise ValueError(f"Cannot multiply matrices of incorrect dimensions, "
                                 f"self n ({self._dimensions[1]}) != other "
                                 f"m ({other.get_dim()[0]})")
            else:
                # Multiply two matrices with the correct dimensions
                x = self.get_dim()[0]
                y = other.get_dim()[1]
                result_matrix = Matrix(m=x, n=y)

                # This nested loop will perform all of the multiplication required
                for i in range(self._dimensions[0]):
                    for c in range(other.get_dim()[1]):
                        num = 0
                        for j in range(other.get_dim()[0]):
                            a = self[i][j] * other[j][c]
                            if num == 0:
                                num = a
                            else:
                                num += a
                        result_matrix[i][c] = num
```

```python

        elif isinstance(other, self.number_types):
            result_matrix = self.__rmul__(other)
        else:
            raise TypeError
        return result_matrix

    def __add__(self, other):
        if issubclass(type(other), type(self)) or issubclass(type(self), type(other)):
            if self.get_dim() != other.get_dim():
                raise ValueError(
                    f"Cannot add matrices of different dimensions, self ({self.get_dim()}) != other ({other.get_dim()})")
            else:
                x = self.get_dim()[0]
                y = self.get_dim()[1]
                result_matrix = Matrix(m=x, n=y)
                # This simply adds all corresponding matrix indices together
                for i in range(x):
                    for j in range(y):
                        result_matrix[i][j] = self[i][j] + other[i][j]
                return result_matrix

        else:
            raise NotImplementedError(
                f"unsupported operand type(s) for +: '{type(self)}' and '{type(other)}'")

    def __sub__(self, other):
        if issubclass(type(other), type(self)) or issubclass(type(self), type(other)):
            if self.get_dim() != other.get_dim():
                raise ValueError(
```

```
                    f"Cannot multiply matrices of different dimensions, self ({self.get_dim()}) != other ({other.get_dim()})")
            else:
                x = self.get_dim()[0]
                y = self.get_dim()[1]
                result_matrix = Matrix(m=x, n=y)
                # This simply subtracts all corresponding matrix indices
                for i in range(x):
                    for j in range(y):
                        result_matrix[i][j] = self[i][j] - other[i][j]
                return result_matrix
        else:
            raise NotImplementedError(
                f"unsupported operand type(s) for +: '{type(self)}' and '{type(other)}'")

    def __abs__(self):
        # This replaces each index in the matrix with its absolute value
        # It does not represent the determinant of the matrix
        x = self.get_dim()[0]
        y = self.get_dim()[1]
        result_matrix = Matrix(m=x, n=y)
        for i in range(x):
            for j in range(y):
                result_matrix[i][j] = abs(self[i][j])
        return result_matrix

    def get_dim(self):
        return self._dimensions

    @staticmethod
    def exp(x, lst):
        # This utility function will take every item in a list and raise e
```

```
        # To the power of that item
        result = []
        for i in lst:
            result += [cmath.exp(x * i)]
        return Matrix([result])

    def concatenate(self, other, direction):
        # This function concatenates two matrices together, if they are matching
        # dimensions, in either a horizontal or vertical direction
        other_m, other_n = zip(other.get_dim())
        other_m, other_n = other_m[0], other_n[0]
        if not (issubclass(type(other), type(self)) or issubclass(type(self), type(other))):
            raise ValueError(f"unsupported type for concatanate: {type(other)}")
        if direction in ["vertical", "v"]:
            if not self.get_dim()[1] == other.get_dim()[1]:
                raise ValueError

            result = Matrix(m=self.get_dim()[0] + other.get_dim()[0],
                            n=self.get_dim()[1])

            for y in range(self.get_dim()[0]):
                for x in range(result.get_dim()[1]):
                    result[y][x] = self[y][x]

            for y in range(self.get_dim()[0], result.get_dim()[0]):
                for x in range(result.get_dim()[1]):
                    result[y][x] = other[y - self.get_dim()[0]][x]

        elif direction in ["horizontal", "h"]:
            if not self.get_dim()[0] == other.get_dim()[0]:
                raise ValueError

```

```
            result = Matrix(m=self.get_dim()[0],
                            n=self.get_dim()[1] + other.get_dim()[1])

            for x in range(self.get_dim()[1]):
                for y in range(result.get_dim()[0]):
                    result[y][x] = self[y][x]

            for x in range(self.get_dim()[1], result.get_dim()[1]):
                for y in range(result.get_dim()[0]):
                    result[y][x] = other[y][x - self.get_dim()[1]]

        else:
            raise ValueError

        return result

    def section(self, mini, maxi, direction):
        # Takes either a horizontal or vertical slice of the matrix between
        # mini and maxi inclusive
        if direction in ["vertical", "v"]:
            result = Matrix(m=self.get_dim()[0], n=maxi-mini+1)
            for i in range(result.get_dim()[0]):
                result[i] = self[i][mini:(maxi+1)]

        elif direction in ["horizontal", "h"]:
            result = Matrix(m=maxi-mini+1, n=self.get_dim()[1])
            for i in range(result.get_dim()[0]):
                result[i] = self[i+mini]
        else:
            raise ValueError(f"Incorrect direction for sectioning: {direction}")

        return result

```

```python

class Wave:
    """A representation of a Wave file, must be created from a string containing the location of a
    wave file on disk"""

    def __init__(self, path):
        with open(path, "rb") as raw_wave_file:
            contents = raw_wave_file.read()

        # Check the header chunk for correctly set values
        # https://blogs.msdn.microsoft.com/dawate/2009/06/23/intro-to-audio-program
        # The above article is nicely formatted but has a bunch of mistakes
        if contents[0:4] != b"RIFF" or contents[8:12] != b"WAVE":
            raise TypeError("Specified file is not in wave format")

        fileSize = contents[4:8]
        self.fileSize = int(Wave.little_bin(fileSize), 2)  # This correctly calculates filesize
        headerChunk = contents[:12]

        fmtSizeRaw = contents[16:20]
        fmtSize = int(Wave.little_bin(fmtSizeRaw), 2)
        # formatChunk = contents[12:20+fmtSize]
        # bytes 12:16 'fmt '
        sampleRate = contents[24:26]
        self.sampleRate = int(Wave.little_bin(sampleRate), 2)

        channels = contents[22:24]
        self.channels = int(Wave.little_bin(channels), 2)

        frameSize = contents[32:34]
        self.frameSize = int(Wave.little_bin(frameSize), 2)
```

```python

        bitDepth = contents[34:36]
        self.bitDepth = int(Wave.little_bin(bitDepth), 2)

        # bytes 36:40 = "data"
        dataLen = contents[40:44]
        self.dataLen = int(Wave.little_bin(dataLen), 2)

        # Read in data from array
        self.frameStartIndex = 44

        framesNum = self.dataLen / self.frameSize
        if framesNum.is_integer():
            framesNum = int(framesNum)
        else:
            raise ValueError("Non integer frame number")

        self.frameDataLists = [[] for i in range(self.channels)]
        for frame in range(framesNum):
            # Loops through every frame in the wave file and splits apart the different samples
            start = self.frameStartIndex + frame * self.frameSize
            end = self.frameStartIndex + (frame + 1) * self.frameSize
            data = contents[start:end]
            if not len(data) == self.channels * self.bitDepth // 8:
                raise ValueError("Invalid bit depth")
            n = self.bitDepth // 8
            samples = [data[i:i + n] for i in range(0, len(data), n)]
            for i in range(self.channels):
                self.frameDataLists[i].append([self.signed_int(samples[i])])

        self.dataMatrix = Matrix(self.frameDataLists[0])
        for channel in range(len(self.frameDataLists)): # Finally stores the sample filled channels into a Matrix
```

```
                self.dataMatrix.concatenate(Matrix(self.frameDataLists[channel]), "h")

    @staticmethod
    def little_bin(rawbytes):
        """Returns the binary reprsesentation of an unsigned 32 bit integer,
            from little endian hex"""
        bytez = []
        for i in rawbytes:
            bytez.append(hex(i)[2:].zfill(2))
        hexstr = "".join(bytez[::-1])
        result = ""
        for x in hexstr:
            digits = bin(int(x, 16))[2:].zfill(4)
            result += digits
        return result

    def signed_int(self, rawbytes):
        """Returns the integer representation of a signed integer,
            from binary"""
        if self.bitDepth == 8 or self.bitDepth == 16:
            binary = Wave.little_bin(rawbytes)
            if binary[0] == "1":
                res = -32768 + int(Wave.little_bin(rawbytes)[1:], 2)
            else:
                res = int(Wave.little_bin(rawbytes), 2)
            return res

        elif self.bitDepth == 32:
            # Data is a float (-1.0f ro 1f)
            raise NotImplementedError("Cannot read 32 bit wave file")

    def get_channel(self, chan):
```

```
        return self.dataMatrix.section(chan, chan, "v")

    def convert_hertz(self, vector):
        """Converts a fourier transform output index to its value in
         ↪ Hz"""
        N = vector.get_dim()[0]
        T = N / self.sampleRate
        df = 1 / T
        result = Matrix(m=N, n=1)
        for n in range(N):
            if n < N / 2:
                result[n][0] = df * n // 2
            else:
                result[n][0] = df * (n - N) // 2
        return result


class Fourier(Matrix):
    """Performs a fourier transform on one Matrix of time domain values
     ↪ and returns a Matrix of
    frequency domain values"""

    def __init__(self, matrix, pad=False):
        super().__init__(matrix)
        self._p = math.ceil(math.log(matrix.get_dim()[0], 2))
        # This sets up the inheritence from the marix class and pads the
         ↪ matix
        # if required

        if pad:
            length = 2**self._p - matrix.get_dim()[0]
            if length > 0:
                left = math.ceil(length/2)
                right = length//2
```

```
                self._contents = Matrix(m=left, n=1).concatenate(self,
                 ↪  "v").concatenate(Matrix(m=right, n=1), "v")._contents
            self._dimensions[0] = 2**self._p

        self._omega_N = cmath.exp(-2j * math.pi / self.get_dim()[0])

    def get_p(self):
        return int(self._p)

    def get_omega(self):
        return self._omega_N

    def DFT(self):
        # Performs a DFT on itself
        N = self.get_dim()[0]
        operator = Matrix(m=N, n=N)
        for x in range(N):
            for y in range(N):
                operator[y][x] = self._omega_N ** (x * y)
        return operator * self

    @staticmethod
    def find_peaks(vector, lag, threshold, influence):
        """Peak detection algorithim from

 ↪  https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-ti
        signals = Matrix(m=vector.get_dim()[0], n=1)
        filteredY = vector
        avgFilter = Matrix(m=vector.get_dim()[0], n=1)
        stdFilter = Matrix(m=vector.get_dim()[0], n=1)
        avgFilter[lag-1] = Fourier.avg(vector.section(0, lag-1, "h"))
        stdFilter[lag-1] = Fourier.std(vector.section(0, lag-1, "h"))

        for i in range(lag+1, vector.get_dim()[0]):
```

```
            if abs(vector[i][0] - avgFilter[i-1][0]) > threshold * stdFilter[i-1][0]:
                if vector[i][0] > avgFilter[i-1][0]:
                    signals[i][0] = 1
                else:
                    signals[i][0] = -1
                filteredY[i][0] = influence*vector[i][0] + (1-influence)*filteredY[i-1][0]

            else:
                signals[i][0] = 0
                filteredY[i][0] = vector[i][0]

            avgFilter[i][0] = Fourier.avg(filteredY.section(i-lag, i, "h"))
            stdFilter[i][0] = Fourier.std(filteredY.section(i-lag, i, "h"))

        return signals

    @staticmethod
    def filter_peaks(lst):
        # This function filters peaks based on difference from preceding peaks
        lst = list(lst)
        result_list = []
        while len(lst) > 0:
            temp = [[lst.pop(0)]]
            while len(lst) > 0 and abs(Fourier.avg(Matrix(temp)) - lst[0]) < 6*len(temp):
                temp.append([lst.pop(0)])
            result_list.append(int(Fourier.avg(Matrix(temp))))
        return result_list

    @staticmethod
```

```
    def avg(vector):
        # Calculates the mean of a vector
        n = vector.get_dim()[0]
        total = sum([i[0] for i in vector._contents])
        return total / n

    @staticmethod
    def std(vector):
        # Calculates the standard deviation of a vector
        n = vector.get_dim()[0]
        total = sum([i[0]**2 for i in vector])
        return math.sqrt(total / n - Fourier.avg(vector)**2)

    def FFT(self):
        N = self.get_dim()[0]
        if N <= 2:
            return self.DFT()
        else:
            even, odd = self[::2], self[1::2]
            even, odd = Fourier(Matrix(even)), Fourier(Matrix(odd))
            even, odd = even.FFT(), odd.FFT()

            factor = Fourier.exp(-2j * math.pi / N, list(range(N)))

            first = Matrix(m=even.get_dim()[0], n=even.get_dim()[1])
            second = Matrix(m=even.get_dim()[0], n=even.get_dim()[1])

            for i in range(even.get_dim()[0]):
                first[i][0] = even[i][0] + factor[0][:N // 2][i] * odd[i][0]
                second[i][0] = even[i][0] + factor[0][N // 2:][i] * odd[i][0]

            return Fourier(first.concatenate(second, "v"))

```

```
    @staticmethod
    def rms(vector):
        # Determines the root-mean-square of a vector
        return math.sqrt(sum([i[0] ** 2 for i in vector]) / vector.get_dim()[0])

    @staticmethod
    def blackman_harris(vector):
        # Applys a blackman-harris window function to the vector
        a = [0.35875, 0.48829, 0.14128, 0.01168]
        N = vector.get_dim()[0]
        result = Fourier(Matrix(m=N, n=1))
        for i in range(N):
            window = a[0] - a[1] * math.cos((2 * math.pi * i) / (N - 1)) + a[2] * math.cos(
                (4 * math.pi * i) / (N - 1)) - a[3] * math.cos((6 * math.pi * i) / (N - 1))
            result[i][0] = window * vector[i][0]
        return result

    @staticmethod
    def med(vector):
        # Finds the median of a vector
        values = sorted([i[0] for i in vector])
        if len(values) % 2 == 0:
            x = values[len(values) // 2 - 1:len(values) // 2 + 1]
            return sum(x) / 2
        else:
            return values[len(values) // 2]

    @staticmethod
    def median_filter(vector, size):
        # Performs a median filter over the vectcor
        y = Matrix(m=vector.get_dim()[0], n=1)
        y[0][0] = vector[0][0]
```

```
        y[vector.get_dim()[0] - 1][0] = vector[vector.get_dim()[0] -
         ↪ 1][0]
        end = size / 2
        for i in range(int(vector.get_dim()[0] - end) - size):
            # print(vector.get_dim()[0], i+size-1, i,
             ↪ int(vector.get_dim()[0]-end))
            window = vector.section(i, i + size - 1, "h")
            y[int(end + i)][0] = Fourier.med(window)
        return y


class Midi:
    """A representation of a midi file,
     can be written to an actual file through use of .write(filename)"""

    def __init__(self):
        # These values are constant for the midi file output
        self.format = 0
        self.tracks = 1
        self.division = 96
        self.events = [(0, 0)]

    def hz_to_key(self, hz):
        x = int(69 + 12 * math.log(hz / 440, 2))
        if x not in list(range(128)):
            print(f"broken {x}")
        return hex(x)[2:]

    def velocity_to_hex(self, v):
        return "40" # Different velocities have been disbabled as they
         ↪ did not
        # improve sound quality

    def sample_to_tick(self, sample):
        return int(int(sample) // (44100 / (2 * self.division)))
```

```

    def add_note(self, start_sample, end_sample, note, velocity, channel=0):
        # At 120 BPM, 1s = 2b
        # 96 ticks per 1/4 note
        # 230 samples per tick
        note_on = "9" + hex(channel)[2:] + self.hz_to_key(note) + self.velocity_to_hex(velocity)
        note_off = "8" + hex(channel)[2:] + self.hz_to_key(note) + "40"
        if int(end_sample) - int(start_sample) > 1000: # This filters out very short notes as noise
            self.events.append((self.sample_to_tick(start_sample), note_on))
            self.events.append((self.sample_to_tick(end_sample), note_off))

    def write(self, filename):
        # Prepare file header
        header = "4d546864000000006"
        header += hex(self.format)[2:].zfill(4)
        header += hex(self.tracks)[2:].zfill(4)
        header += hex(self.division)[2:].zfill(4)

        # Prepare track data
        track_data = ""
        ordered_events = list(sorted(self.events, key=lambda tup: tup[0]))
        delta_times = [ordered_events[i][0] - ordered_events[i - 1][0] for i in
                           range(1, len(ordered_events))]
        delta_vlq = [Midi.vlq(i) for i in delta_times]
        for index, event in enumerate(ordered_events):
            if index != 0:  # Empty event to begin
                track_data += delta_vlq[index - 1] + event[1]

```

```
        track_data += "19ff2f0000ff2f00"  # End of track event

        # Prepare track header
        track_header = "4d54726b"
        track_header += hex(len(track_data) // 2)[2:].zfill(8)

        # Write file
        final_hex_string = header + track_header + track_data
        with open(filename, "wb") as midi_file:
            midi_file.write(bytearray.fromhex(final_hex_string))

    @staticmethod
    def vlq(value):
        # This converts an integer into a binary variable length quantity
        bits = list(bin(value)[2:])
        while len(bits) % 7 != 0:
            bits = ["0"] + bits
        rev_bits = bits[::-1]
        result = []
        for i, value in enumerate(rev_bits):
            result.append(value)
            if (i + 1) == 7:
                result.append("0")
            elif (i + 1) % 7 == 0:
                result.append("1")
        binary_str = "".join(result)[::-1]
        hex_result = [hex(int(binary_str[i:i + 4], 2))[2:] for i in range(0, len(binary_str), 4)]
        return "".join(hex_result)


if __name__ == '__main__':
    filename = "blind.wav"

```

```python
    # Sets the fourier size and increment values, experimentally these seem good
    FOURIER_SIZE = 2048
    FOURIER_INCREMENT = 256

    print(f"\nProcessing begun on file '{filename}', this will take a while.\n")

    # This creates a cache of the file, if it needs to be converted again
    loadStartTime = time.time()
    try:
        with open(filename[:-4] + ".pickle", "rb") as file:
            print("Cached file version found!\n")
            wave_file = pickle.load(file)
    except FileNotFoundError:
        print("No cache found.\n")
        wave_file = Wave(filename)
        with open(filename[:-4] + ".pickle", "wb") as file:
            pickle.dump(wave_file, file, protocol=pickle.HIGHEST_PROTOCOL)
    loadEndTime = time.time()
    print(f"* Wave load complete. Elapsed time {loadEndTime - loadStartTime} seconds.")

    wave_channel = wave_file.get_channel(0)

    results_lst = []
    for offset in range((int(wave_channel.get_dim()[0]) - (
            FOURIER_SIZE - FOURIER_INCREMENT)) // FOURIER_INCREMENT):
        signal = Fourier(wave_channel.section(offset * FOURIER_INCREMENT,
                                              (offset * FOURIER_INCREMENT + FOURIER_SIZE) - 1,
                                              "h"), pad=True)
        results_lst.append(Fourier.rms(signal))

```

```
    v = Matrix([[i] for i in results_lst])
    x = [i[0] for i in Fourier.find_peaks(v, 10, 3, 0.1)]
    dividers = []
    prev = 0
    for i in range(1, len(x)):
        if x[i] == 1 and x[i - 1] == 0:
            if i - prev > 25:
                prev = i
                dividers.append(i)
    dividers.append(len(x))

    noteEndTime = time.time()
    print(f"* Note partitioning complete. Elapsed time {noteEndTime -
     ↪ loadEndTime} seconds.")

    midi_file = Midi()


    if len(dividers) > 0:
        start = 0
        total = len(dividers)
        # Splits the file into each seperate note section
        for j in dividers:
            current = dividers.index(j)
            end = j * FOURIER_INCREMENT
            # Moves over each note and determines its pitch
            if start != end:
                signal = Fourier(wave_channel.section(start, (end) - 1,
                 ↪ "h"), pad=True)
                signal = Fourier.blackman_harris(signal)
                corr = abs(Fourier.FFT(signal))
                post = Fourier.median_filter(corr, 15).section(0,
                 ↪ corr.get_dim()[0] // 2, "h")

                value = max([i[0] for i in post])
```

```
                pos = post._contents.index([value])
                hz_post = wave_file.convert_hertz(post)
                if hz_post[pos][0] > 0:
                    midi_file.add_note(start, end, hz_post[pos][0], 40)
            start = end

    else:
        length = 2 ** int(math.log(wave_file.get_data()[0].get_dim()[0] - 1, 2))
        signal = Fourier(wave_channel.section(0, length - 1, "h"), pad=True)
        corr = abs(Fourier.autocorrelation(signal))
        post = Fourier.median_filter(corr, 15).section(0, corr.get_dim()[0] // 2, "h")

    fourierEndTime = time.time()
    print(
        f"* Fourier transforms complete. Elapsed time {fourierEndTime - noteEndTime} seconds.")

    # Completes the conversion and writes out the results
    midi_file.write(filename[:-4] + ".mid")
    endEndTime = time.time()
    print(f"* Midi file write complete. Elapsed time {endEndTime - fourierEndTime} seconds.")
    print(f"Total elapsed time {endEndTime - loadStartTime} seconds.")
```

gui.py

```
import os
import pickle
import sys
import time
import math
import threading
```

```

from classes import Matrix, Fourier, Wave, Midi
from PyQt5 import QtWidgets, QtCore


class Window(QtWidgets.QWidget):
    def __init__(self):
        QtWidgets.QWidget.__init__(self)
        self.queue = []

        # Setup Browse Button
        self.btn = QtWidgets.QPushButton('Browse', self)
        self.btn.move(20, 20)
        self.btn.clicked.connect(self.get_files)
        # Setup Enqueue Button
        self.btn = QtWidgets.QPushButton('En-Queue', self)
        self.btn.move(20, 60)
        self.btn.clicked.connect(self.enqueue)
        # Setup Dequeue Button
        self.btn = QtWidgets.QPushButton('De-Queue', self)
        self.btn.move(20, 100)
        self.btn.clicked.connect(self.dequeue)
        # Setup File Queue Indicator
        self.label = QtWidgets.QLabel("File(s) in queue: 0", self)
        self.label.move(360, 105)
        # Setup Begin Button
        self.start_btn = QtWidgets.QPushButton('Begin Conversion', self)
        self.start_btn.move(230, 100)
        self.start_btn.clicked.connect(self.begin)
        # Setup Path Display Box
        self.le = QtWidgets.QLineEdit(self)
        self.le.move(130, 22)
        self.le.resize(315, 20)
        self.le.setDisabled(True)
        # Setup Progress Bar
```

```
        self.progress = QtWidgets.QProgressBar(self)
        self.progress.setGeometry(130, 60, 350, 25)
        self.progress.setMaximum(100)
        # Create Window
        self.setGeometry(300, 300, 500, 150)
        self.setWindowTitle('Wav to Midi Converter')
        self.show()

    def get_files(self):
        # Opens a dialog box to select a file
        fileName, _ = QtWidgets.QFileDialog.getOpenFileName(self, 'Single File', QtCore.QDir.rootPath() , '*.wav')
        self.le.setText(fileName)

    def enqueue(self):
        path = self.le.text()
        self.queue.append(path)
        self.label.setText(f"File(s) in queue: {len(self.queue)}")

    def dequeue(self):
        if len(self.queue) > 0:
            self.queue.pop(-1)
        self.label.setText(f"File(s) in queue: {len(self.queue)}")

    def begin(self):
        # Begins the conversion, using a seperate thread to increase avalible memory
        if len(self.queue) == 0:
            path = self.le.text()
            filename = path[-path[::-1].index("/"):]
            filesize = os.path.getsize(path)
            time_est = int(filesize * 0.0004161731354229695)
```

```python
            QtWidgets.QMessageBox.question(self, 'Alert', f"Coversion has begun on {filename}, this may take a long time.\nEstimate of {time_est}s", QtWidgets.QMessageBox.Ok, QtWidgets.QMessageBox.Ok)
            thread = threading.Thread(target=self.main, args=(path,))
            thread.start()
        else:
            for p in self.queue:
                path = p
                filename = path[-path[::-1].index("/"):]
                filesize = os.path.getsize(path)
                time_est = int(filesize * 0.00041617313542296 95)
                QtWidgets.QMessageBox.question(self, 'Alert', f"Coversion has begun on {filename}, this may take a long time.\nEstimate of {time_est}s", QtWidgets.QMessageBox.Ok, QtWidgets.QMessageBox.Ok)
                thread = threading.Thread(target=self.main, args=(path,))


    def main(self, path):
        filename = path[-path[::-1].index("/"):]

        # Sets the fourier size and increment values, experimentally these seem good
        FOURIER_SIZE = 2048
        FOURIER_INCREMENT = 256

        print(f"\nProcessing begun on file '{filename}', this will take a while.\n")

        # This creates a cache of the file, if it needs to be converted again
        loadStartTime = time.time()
        try:
            with open(filename[:-4] + ".pickle", "rb") as file:
```

```
                print("Cached file version found!\n")
                wave_file = pickle.load(file)
        except FileNotFoundError:
            print("No cache found.\n")
            wave_file = Wave(path)
            with open(filename[:-4] + ".pickle", "wb") as file:
                pickle.dump(wave_file, file, protocol=pickle.HIGHEST_PROTOCOL)
        loadEndTime = time.time()
        print(f"* Wave load complete. Elapsed time {loadEndTime - loadStartTime} seconds.")

        wave_channel = wave_file.get_channel(0)

        results_lst = []
        for offset in range((int(wave_channel.get_dim()[0]) - (
                FOURIER_SIZE - FOURIER_INCREMENT)) // FOURIER_INCREMENT):
            signal = Fourier(wave_channel.section(offset * FOURIER_INCREMENT,
                                                  (offset * FOURIER_INCREMENT + FOURIER_SIZE) - 1,
                                                  "h"), pad=True)
            results_lst.append(Fourier.rms(signal))

        v = Matrix([[i] for i in results_lst])
        x = [i[0] for i in Fourier.find_peaks(v, 10, 3, 0.1)]
        dividers = []
        prev = 0
        for i in range(1, len(x)):
            if x[i] == 1 and x[i - 1] == 0:
                if i - prev > 25:
                    prev = i
                    dividers.append(i)
        dividers.append(len(x))
```

```python

        # Updates the progress bar
        self.progress.setValue(5)
        noteEndTime = time.time()
        print(f"* Note partitioning complete. Elapsed time {noteEndTime - loadEndTime} seconds.")

        midi_file = Midi()


        if len(dividers) > 0:
            start = 0
            total = len(dividers)
            # Splits the file into each seperate note section
            for j in dividers:
                current = dividers.index(j)
                self.progress.setValue(int((current*95)/total) + 5)
                end = j * FOURIER_INCREMENT
                # Moves over each note and determines its pitch
                if start != end:
                    signal = Fourier(wave_channel.section(start, (end) - 1, "h"), pad=True)
                    signal = Fourier.blackman_harris(signal)
                    corr = abs(Fourier.FFT(signal))
                    post = Fourier.median_filter(corr, 15).section(0, corr.get_dim()[0] // 2, "h")

                    value = max([i[0] for i in post])
                    pos = post._contents.index([value])
                    hz_post = wave_file.convert_hertz(post)
                    if hz_post[pos][0] > 0:
                        midi_file.add_note(start, end, hz_post[pos][0], 40)
                start = end

```

```
        else:
            length = 2 ** int(math.log(wave_file.get_data()[0].get_dim()[0] - 1, 2))
            signal = Fourier(wave_channel.section(0, length - 1, "h"), pad=True)
            corr = abs(Fourier.autocorrelation(signal))
            post = Fourier.median_filter(corr, 15).section(0, corr.get_dim()[0] // 2, "h")

        fourierEndTime = time.time()
        print(
            f"* Fourier transforms complete. Elapsed time {fourierEndTime - noteEndTime} seconds.")

        # Completes the conversion and writes out the results
        self.progress.setValue(100)
        midi_file.write(filename[:-4] + ".mid")
        endEndTime = time.time()
        print(f"* Midi file write complete. Elapsed time {endEndTime - fourierEndTime} seconds.")
        print(f"Total elapsed time {endEndTime - loadStartTime} seconds.")


if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    ex = Window()
    sys.exit(app.exec_())
```

Testing

In order to ensure the functionality on my program I will test different aspects of its performance against my original specification. Throughout the development process I have been testing the functionality of my program using an agile programming paradigm, to ensure that each stage of the conversion process could be used to verify the next was working correctly. I have broken down the testing documentation into the components of the program of which they test.

The Matrix Class

The tests on the Matrix class are to ensure that Objective 3 has been met by the program, to recap this means the program must be able to correctly add, multiply, slice and concatenate matrices together.

Addition

To test the addition function I will try 3 different input cases, one with positive integers, one with a mix of positive and negative integers and one with complex numbers. The class should be able to handle a matrix containing any object types that support addition, but these 3 combinations will be encountered during the run of the program. I have also included a test to ensure the program will not attempt to perform addition on matrices of different dimensions.

Test 1, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

```
In [8]: A = Matrix([[1, 2], [3, 4]])

In [9]: B = Matrix([[1, 1], [1, 1]])

In [10]: A + B
Out[10]: [[2, 3], [4, 5]]
```

Figure 17: Test 1, Success

Test 2, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} -1 & -2 \\ -4 & -3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix}$$

```
In [12]: A = Matrix([[1, 2], [3, 4]])

In [13]: B = Matrix([[-1, -2], [-4, -3]])

In [14]: A + B
Out[14]: [[0, 0], [-1, 1]]
```

Figure 18: Test 2, Success

Test 3, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} i & -2i \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 1+i & 2-2i \\ 2 & 6 \end{bmatrix}$$

```
In [16]: A = Matrix([[1, 2], [3, 4]])

In [17]: B = Matrix([[1j, -2j], [-1, 2]])

In [18]: A + B
Out[18]: [[(1+1j), (2-2j)], [2, 6]]
```

Figure 19: Test 3, Success

Test 4, Input and Expected Output

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = ERROR$$

```
In [22]: A = Matrix([[1], [2]])

In [23]: B = Matrix([[1j, -2j], [-1, 2]])

In [24]: A + B
Traceback (most recent call last):

  File "<ipython-input-24-151064de832d>", line 1, in <module>
    A + B

  File "E:/Users/oisin/Documents/RREEPPOOSS/NEA_Fourier/tests/test_01.py", line
171, in __add__
    f"Cannot add matrices of different dimensions, self ({self.get_dim()}) !=
other ({other.get_dim()})")

ValueError: Cannot add matrices of different dimensions, self ([2, 1]) != other
([2, 2])
```

Figure 20: Test 4, Success

Multiplication

To test the multiplication functionality, I need to ensure that the class can handle all types of multiplication between vectors, scalars and matrices correctly. Some combinations of matrix multiplication are not mathematically valid, so I need to ensure my program correctly determines when it can and cannot perform a multiplication.

Test 1, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$$

```
In [41]: A = Matrix([[1, 2], [3, 4]])

In [42]: B = Matrix([[1, 1], [1, 1]])

In [43]: A * B
Out[43]: [[3, 3], [7, 7]]
```

Figure 21: Test 1, Success

Test 2, Input and Expected Output

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ 4 & 6 \end{bmatrix}$$

```
In [48]: A = Matrix([[1, 2], [3, 4]])

In [49]: B = Matrix([[1, 1], [1, 1]])

In [50]: B * A
Out[50]: [[4, 6], [4, 6]]
```

Figure 22: Test 2, Success

Test 3, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 22 \end{bmatrix}$$

```
In [52]: A = Matrix([[1, 2], [3, 4]])

In [53]: B = Matrix([[2], [4]])

In [54]: A * B
Out[54]: [[10], [22]]
```

Figure 23: Test 3, Success

Test 4, Input and Expected Output

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = ERROR$$

```
In [56]: A = Matrix([[1, 2], [3, 4]])

In [57]: B = Matrix([[2], [4]])

In [58]: B * A
Traceback (most recent call last):

  File "<ipython-input-58-36ca14b77ca1>", line
1, in <module>
    B * A

  File "E:/Users/oisin/Documents/RREEPPOOSS/
NEA_Fourier/tests/test_02.py", line 141, in
__mul__
    raise ValueError(f"Cannot multiply matrices
of incorrect dimensions, "

ValueError: Cannot multiply matrices of
incorrect dimensions, self n (1) != other m (2)
```

Figure 24: Test 4, Success

Test 5, Input and Expected Output

$$3\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}3 & 6\\9 & 12\end{bmatrix}$$

```
In [60]: A = 3

In [61]: B = Matrix([[1, 2], [3, 4]])

In [62]: A * B
Out[62]: [[3, 6], [9, 12]]
```

Figure 25: Test 5, Success

Slicing

Test 1, Input and Expected Output

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}.section(1,1,"vertical")->\begin{bmatrix}2\\5\\8\end{bmatrix}$$

```
In [5]: A = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

In [6]: A.section(1, 1, "vertical")
Out[6]: [[2], [5], [8]]
```

Figure 26: Test 1, Success

Test 2, Input and Expected Output

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}.section(1,5,"vertical")->\begin{bmatrix}2 & 3\\5 & 6\\8 & 9\end{bmatrix}$$

```
In [14]: A = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

In [15]: A.section(1, 5, "vertical")
Out[15]: [[2, 3], [5, 6], [8, 9]]
```

Figure 27: Test 2, Success

Test 3, Input and Expected Output

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}.section(1,2,"horizontal")->\begin{bmatrix}4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

```
In [11]: A = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

In [12]: A.section(1, 2, "horizontal")
Out[12]: [[4, 5, 6], [7, 8, 9]]
```

Figure 28: Test 3, Success

Concatenation

Test 1, Input and Expected Output

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}.concatenate(\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix},"vertical")->\begin{bmatrix}1 & 2\\3 & 4\\1 & 1\\1 & 1\end{bmatrix}$$

```
In [17]: A = Matrix([[1, 2], [3, 4]])

In [18]: B = Matrix([[1, 1], [1, 1]])

In [19]: A.concatenate(B, "vertical")
Out[19]: [[1, 2], [3, 4], [1, 1], [1, 1]]
```

Figure 29: Test 1, Success

Test 2, Input and Expected Output

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}.concatenate(\begin{bmatrix}1\\1\end{bmatrix},"horizontal")->\begin{bmatrix}1 & 2 & 1\\3 & 4 & 1\end{bmatrix}$$

```
In [21]: A = Matrix([[1, 2], [3, 4]])

In [22]: B = Matrix([[1], [1]])

In [23]: A.concatenate(B, "horizontal")
Out[23]: [[1, 2, 1], [3, 4, 1]]
```

Figure 30: Test 2, Success

Test 3, Input and Expected Output

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}.concatenate(\begin{bmatrix} 1 \\ 1 \end{bmatrix}, "vertical") -> ERROR$$

```
In [25]: A = Matrix([[1, 2], [3, 4]])

In [26]: B = Matrix([[1], [1]])

In [27]: A.concatenate(B, "vertical")
Traceback (most recent call last):

  File "<ipython-input-27-fb07de0ba0a1>", line 1, in <module>
    A.concatenate(B, "vertical")

  File "E:/Users/oisin/Documents/RREEPPOOSS/NEA_Fourier/tests/test_03.py", line 228, in
concatenate
    raise ValueError

ValueError
```

Figure 31: Test 3, Success

The Wave Class

The wave class only performs the task of reading in a file to begin with, but to do this it makes use of several methods that need to be tested. In order to test the overall result of this is correct, the wave file is opened using a program called audacity to view the raw waveform.

Test 1, Input and Expected Output The input was a wave recording of 3 blind mice, which can be heard in the testing video (link on page 76).

```
In [8]: A = Matrix([[1, 2], [3, 4]])

In [9]: B = Matrix([[1, 1], [1, 1]])

In [10]: A + B
Out[10]: [[2, 3], [4, 5]]
```

Figure 32: Test 1, Success

The Fourier Class

This class is the most important to the success of the project, so ensuring it worked correctly was a top priority. Both the FFT and DFT should produce the same results for the same input, but for large inputs the FFT will be faster.

Test 1, Input and Expected Output

$$DFT(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}) -> \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

```
In [79]: A = Matrix([[1], [0], [0], [0], [0], [0], [0], [0]])

In [80]: B = Fourier(A).DFT()

In [81]: B
Out[81]: [[(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)]]

In [82]: plt.plot([i for i in B])
E:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:501: ComplexWarning: Casting
complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
Out[82]: [<matplotlib.lines.Line2D at 0x2b3848b06a0>]
```

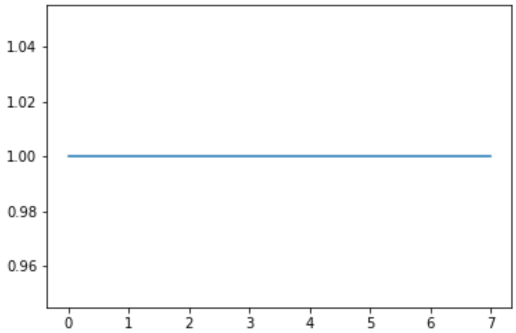

Figure 33: Test 1, Success

Test 2, Input and Expected Output

$$DFT(\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}) -> \begin{bmatrix}1\\7.07-7.07i\\0-i\\-7.07-7.07i\\-1\\-7.07+7.07i\\0+i\\7.07+7.07i\end{bmatrix}$$

```
In [85]: B = Fourier(A).DFT()

In [86]: B
Out[86]: [[(1+0j)], [(0.7071067811865476-0.7071067811865476j)], [-1.0000000000000002j],
[(-0.7071067811865477-0.7071067811865477j)], [(-1.0000000000000004+0j)],
[(-0.7071067811865479+0.7071067811865479j)], [1.0000000000000007j],
[(0.707106781186548+0.707106781186548j)]]

In [87]: plt.plot([i for i in B])
E:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:501: ComplexWarning: Casting
complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
Out[87]: [<matplotlib.lines.Line2D at 0x2b38497f978>]
```

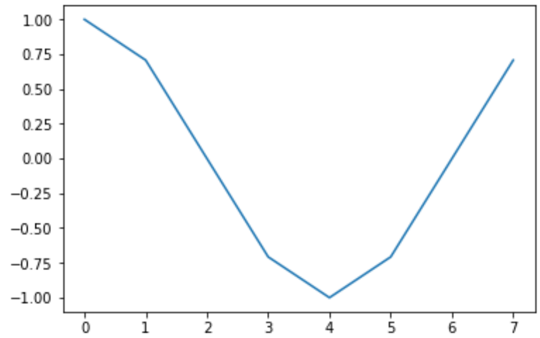

Figure 34: Test 2, Success

Test 3, Input and Expected Output

$$FFT(\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}) -> \begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}$$

```
In [94]: A = Matrix([[1], [0], [0], [0], [0], [0], [0], [0]])

In [95]: B = Fourier(A).FFT()

In [96]: B
Out[96]: [[(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)], [(1+0j)]]

In [97]: plt.plot([i for i in B])
E:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:501: ComplexWarning: Casting
complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
Out[97]: [<matplotlib.lines.Line2D at 0x2b384bb9400>]
```

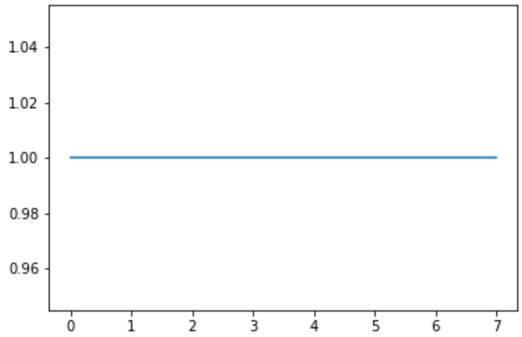

Figure 35: Test 3, Success

Test 4, Input and Expected Output

$$FFT(\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix})->\begin{bmatrix}1\\7.07-7.07i\\0-i\\-7.07-7.07i\\-1\\-7.07+7.07i\\0+i\\7.07+7.07i\end{bmatrix}$$

```
In [89]: A = Matrix([[0], [1], [0], [0], [0], [0], [0], [0]])

In [90]: B = Fourier(A).FFT()

In [91]: B
Out[91]: [[(1+0j)], [(0.7071067811865476-0.7071067811865476j)], [(6.123233995736766e-17-1j)],
[(-0.7071067811865475-0.7071067811865476j)], [(-1-1.2246467991473532e-16j)],
[(-0.7071067811865477+0.7071067811865475j)], [(-1.8369701987210297e-16+1j)],
[(0.7071067811865474+0.7071067811865477j)]]

In [92]: plt.plot([i for i in B])
E:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:501: ComplexWarning: Casting
complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
Out[92]: [<matplotlib.lines.Line2D at 0x2b384a60ef0>]
```

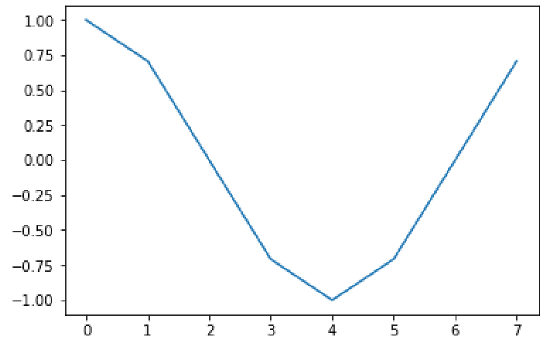

Figure 36: Test 4, Success

The Midi Class

The midi class writes the final file out to disk, so it needs to always correctly format the output file to ensure that it is readable by any synthesizer the user wants to use to play back the file.

Test 1, Input and Expected Output By inputting the results of the conversion of the 3 Blind Mice file, the midi class should write a correctly formatted file to disk.

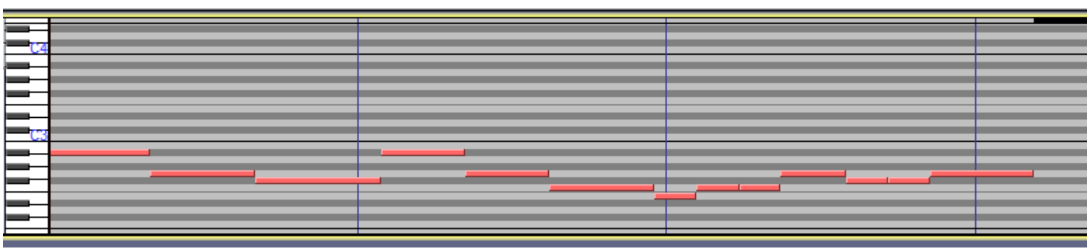

Figure 37: Test 1, Success

The GUI Class

The GUI itself is quite simple, as there are few options for the user to configure. The main features are the time estimation, progress bar and the ability to queue up files.

Test 1, Input and Expected Output The GUI should launch.

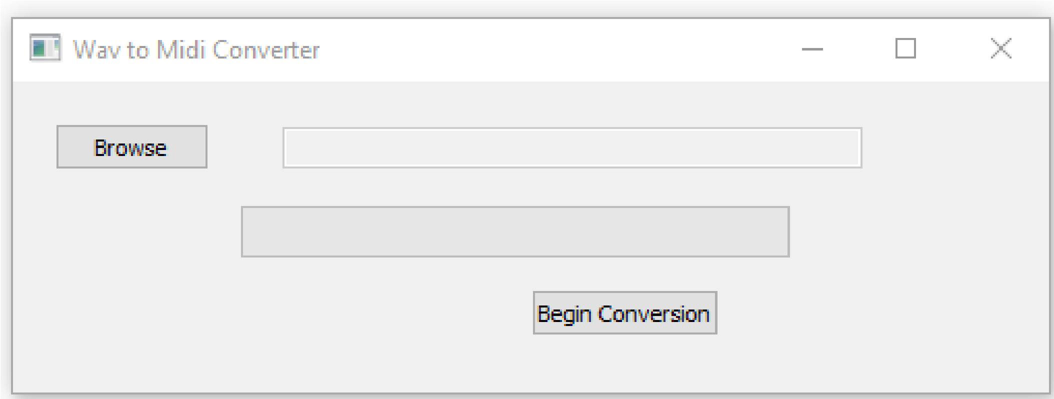

Figure 38: Test 1, Success

Test 2, Input and Expected Output When clicked, the browse button should open a file explorer window allowing the user to select a file.

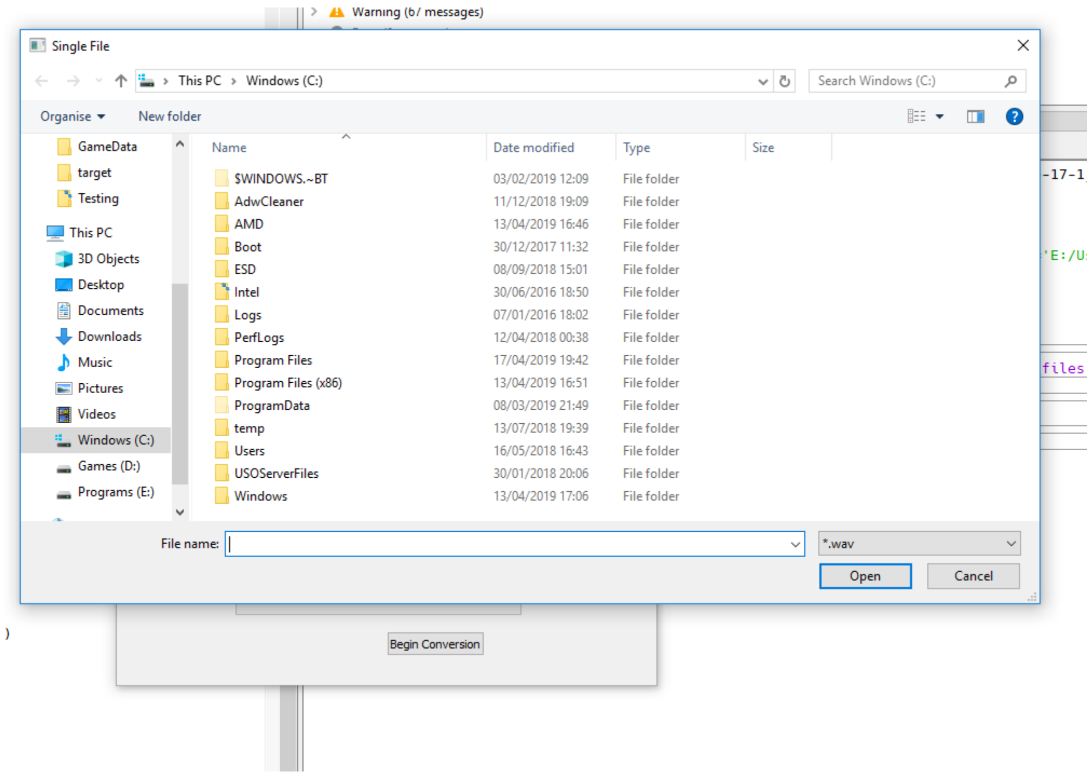

Figure 39: Test 2, Success

Test 3, Input and Expected Output Before conversion, a popup should appear with an estimated length of time until the conversion is completed. This time is a slight overestimate and represents a worse case scenario.

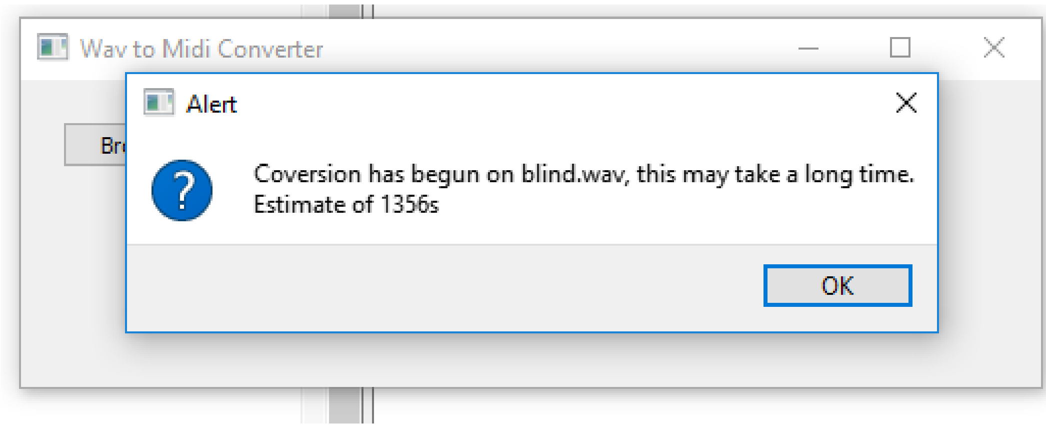

Figure 40: Test 3, Success

Test 4, Input and Expected Output During a conversion a progress bar should be shown to the user, this should update as the conversion progresses.

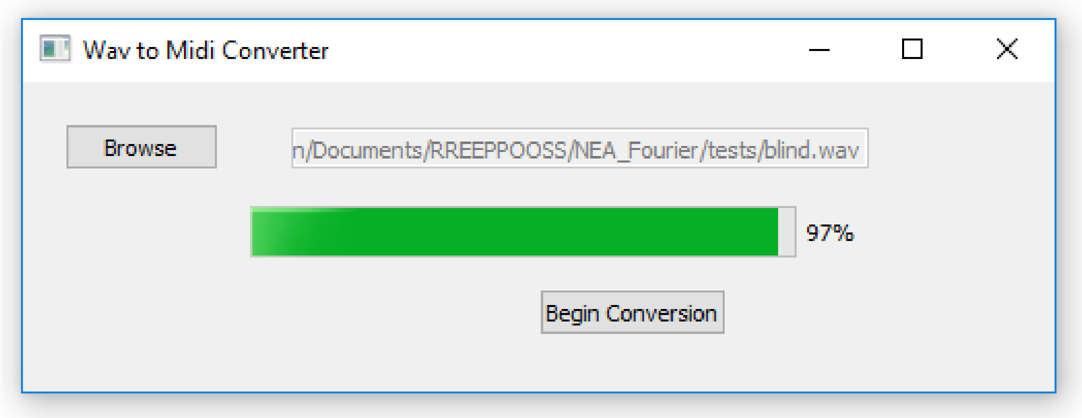

Figure 41: Test 4, Success

The Complete Solution

As the result of this program is an audio file instead of visual, I have created several recordings of the program operating to showcase this. Each of the files that can be found in the following Google Drive folder *https://youtu.be/9qqKU9sJCDg* contains a video walking through the tests and their outcomes. In summary however the program performs as expected in all areas, the final quality of the conversion were not as good as the online Bear converter however. I believe this was due to not using autocorrelation, although it may have been other factors. Despite this however, the program has still met its objectives as I did not expect the conversion to be perfect. In addition feedback from the end user describes the final result as good enough for their purposes, no converter can be 100% accurate but I am overall pleased at how mine turned out.

EVALUATION

In conclusion my solution does solve the problem that it set out to do. Whilst it is not as fast or accurate as the online solution provided by Bear, it does fulfil the needs of my user by running offline and having the code publicly available and editable so it was a success.

Objective Completion

All objectives set out in my Analysis section were successfully completed, no objectives that I set out to fulfil based on the needs of my user were left incomplete. Of the extension tasks, one was completed allowing for the user to queue multiple files when using the GUI. The velocity of notes was partially implemented, however this feature is disabled by default as it did not add a noticeable change to the audio quality whilst still increasing compute time.

1. The program must be able to read the samples from a wave file into an object that represents the samples and information of the file. - This has been completed as required, any type of wave file can be read in for conversion.

2. The program must be able to perform the matrix operations:

   (a) Addition - Complete

   (b) Multiplication - Complete

   (c) Slicing - Complete

   (d) Concatenation - Complete

3. The program must be able to correctly write the results to a midi file, providing the input file is under a length of 3 minutes. - This has been completed, with a file size greater than 3 minutes long the program can become unstable although this happens

somewhat randomly. By staying below the 3 minute mark the performance is much more consistent.

4. The user must be able to select a wave file to convert using a GUI. - Complete, as mentioned files can also be queued.

5. The user must be able to select a wave file to convert without using a GUI. - Complete

6. The user must receive conformation that the conversion has taken place successfully, or that an error of some kind has occurred. - Complete, a series of notification boxes ran on separate threads will alert the user if anything has gone wrong.

7. During conversion, the user should have visual feedback in the form of a timer or progress bar to ensure the program is functioning. - Complete

8. The program must be able to convert files that have only one note played at a time. - Complete

End User Feedback

After showcasing my final solution to both the end user interviewed in my Analysis and one of their friends I am pleased with the responses that they had to it. I was happy to see that they thought it met their needs enough that they could use it when required and that the accuracy was good enough for this as well. The program's UI was simple enough that they needed no instruction on how to use the program and they especially liked the inclusion of the progress bar. They did mention that the speed would be a prime area for improvement although it was not a major issue as they were able to queue up multiple songs to be converted overnight or whilst they were working on something else. Overall the feedback has been positive.

Further Work

As this topic still represents a fairly open problem, an almost endless amount of optimisation could be performed on the code to increase its accuracy and speed. In particular there are two main areas I would like to expand upon, these being the use of autocorrelation and the speed of the program. Theoretically autocorrelation should yield much better results than the "pure" FFT that I am currently using, although I could not get it working to do so with the time available, so it would be interesting to spend more time and implement that successfully. I would also like to try porting the codebase into a different language, such as

Java or MATLAB to try and improve the efficiency of it. Python has allowed me to rapidly prototype and develop my solution, but it is unideal for running performance critical code as it cannot be compiled and is relatively optimised for it. Further optimisation could likely be made in the existing Python codebase, although I am unsure how effective they could be versus the time required to implement them. By far the most complex and time-consuming part of this project has been the research and understanding required to solve the problem, so with this now out of the way I could probably start in another language and implement a better second version much more rapidly.

Bibliography

[1] Ableton Music Editor
`https://www.ableton.com/en/manual/converting-audio-to-midi/`

[2] Wavemid Converter
`http://wavesum.net/wavemid-audio-to-midi.html`

[3] Widisoft Music Processing Package
`https://www.widisoft.com/english/mp3-midi-products.html`

[4] The Taiwan National University, The Midi File Format
`https://www.csie.ntu.edu.tw/ r92092/ref/midi/`

[5] Center for Computer Assisted Research in the Humanities at Stanford University, The MIDI Standard
`http://www.ccarh.org/courses/253/handout/smf/`

[6] Unknown Stanford Unvirsity Website
`http://soundfile.sapp.org/doc/WaveFormat/`

[7] NTI Audio, Fast Fourier Transform FFT
`https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft`

[8] Gauss, Carl Friedrich (1876)
`Theoria Interpolationis Methodo Nova Tractata, page 265-327`

[9] G. Heinzel, A. Rudiger, and R. Schilling, Max Planck Society (2002)
`Spectrum and spectral density estimation by the discrete fourier transform (DFT), including a comprehensive list of window functions and some new at-top windows`

[10] Giuliano Monti, Mark Sandler, Department of Electronic Engineering, King's College London (2000)
MONOPHONIC TRANSCRIPTION WITH AUTOCORRELATION

[11] Lizhe Tan and Montri Karnjanadecha, Prince of Songkla University (2003)
PITCH DETECTION ALGORITHM : AUTOCORRELATION METHOD AND AMDF

[12] William A. Yost, Arizona State University (2009)
Pitch Perception

[13] L. Cohen (1998)
The generalization of the Wiener-Khinchin theorem

[14] Cooley, James W.; Tukey, John W. (1965)
An algorithm for the machine calculation of complex Fourier series

All web links were accessed and correct as of April 2019