Presented to the Department of Computer Technology

De La Salle University - Manila

Term 2, A.Y. 2020-2021

In partial fulfillment

of the course

In CEDISP1, S11

Music Note Detector:
Harmonic Product Spectrum Utilization

Submitted by:

AGULTO, Juliana Marie B.

NOROÑA, Yeohan Lorenzo M.

Submitted to:

Prof. Ronald Pascual

May 24, 2021

TABLE OF CONTENTS

BACKGROUND THEORY AND APPLICATION **3**
- Fundamental Frequency 3
- Harmonics 3
- Harmonic Product Spectrum 3
- Octave 4
- Overtones 4
- Pitch 6

FILTER OR SYSTEM SPECIFICATION **7**
- Fast Fourier Transform 7
- Hanning Window 7
- Harmonic Product Spectrum 8

SYSTEM BLOCK DIAGRAM **10**

DESIGN METHODOLOGY **11**

MATLAB IMPLEMENTATION AND CODE **12**
- MATLAB implementation 12
- Code 15
- Test cases 19

RESULTS AND DISCUSSION **20**

SUMMARY AND CONCLUSION **24**

REFERENCES **25**

BACKGROUND THEORY AND APPLICATION

A musical note is made up of a series of peaks known as harmonics. Harmonics are integer or whole number multiples of the fundamental tone or fundamental frequency, or the lowest frequency of the vibration. The overtone is the natural frequency of vibration that is higher than the fundamental tone (Towell, 2020).

This project aims to develop an efficient and user-friendly **music note detector** capable of identifying notes by extracting the correct fundamental frequency from the input note. When a note is played, the program's output is expected to produce the correct note output by interpreting basic theories of musical notes such as octaves, pitch, fundamental frequency, harmonics, overtones, and the Harmonic Products Spectrum (HPS) and MATLAB's signal-processing features.

Fundamental Frequency

The fundamental tone or frequency is the lowest frequency of a vibrating material or object. It is also referred to as the first harmonic. In terms of audibility, it is the most powerful pitch reference. The harmonics and overtones that are part of the wave distinguish different instruments even when they play the same note which is the timbre (Walker, 2020).

Harmonics

Harmonics are audio frequencies that are higher or higher than the fundamental frequency but have a lower amplitude than the fundamental frequency. For example, if a fundamental frequency is 55 Hz, the first harmonic, the second harmonic is 110, the third harmonic is 165, and so on, multiplying by N iterations (Walker, 2020).

Harmonic Product Spectrum

This algorithm is utilized for audio signals with harmonics. It downsamples the spectrum with a certain amount of iterations. The peaks of every downsampled or compressed spectrum by its respective factor will be multiplied together, and form the fundamental frequency’s peak. It is

resistant to noise that is multiplicative and additive. It is adaptable to different inputs. It is also efficient. The human pitch perception is logarithmic; therefore, the low pitches may be less accurate when tracking them in comparison to high pitches (Middleton, 2003).

Octave

In Western Music, it has 12 identifiable pitches. The distance of a note and its other notes from either direction of the scale bears the same name with different octave intervals. In terms of physics, it is the distance between a note and another note whose frequency is either doubled or halved depending on the direction. For instance, the note A4 has a frequency of 440Hz, and its respective pitch by one octave up its scale is A5 which is 880Hz; it is twice its frequency. If it goes one octave down the scale from A4, it is half of its original pitch or frequency which is 220Hz (MasterClass staff, 2020).

Overtones

Overtones, on the other hand, are waveform frequencies that are above or higher than the fundamental frequency, but it is not the case that it is directly related to it (Walker, 2020).

Table 1

Frequencies of Musical Notes (Suits, 1998)

Octave Scale	Identifiable Notes	Frequency (Fundamental Frequency)
3	C	130.81
	C# | D^b	138.59
	D	146.83
	D# | E^b	155.56
	E	164.81
	F	174.61
	F# | G^b	185.00

	G	196.00
	G# | A^b	207.65
	A	220.00
	A# | B^b	233.08
	B	246.94
4	C	261.63
	C# | D^b	277.18
	D	293.66
	D# | E^b	311.13
	E	329.63
	F	349.23
	F# | G^b	369.99
	G	392.00
	G# | A^b	415.30
	A	440.00
	A# | B^b	466.16
	B	493.88
5	C	523.25
	C# | D^b	554.37
	D	587.33
	D# | E^b	622.25
	E	659.25
	F	698.46
	F# | G^b	739.99
	G	783.99

	G# | A^b	830.61
	A	880.00
	A# | B^b	932.33
	B	987.77

Note. Frequencies of Musical Notes by Suits, B. H. (1998). Retrieved from https://pages.mtu.edu/~suits/notefreqs.html

Pitch

A higher frequency corresponds to a higher pitch, whereas a lower pitch corresponds to a lower frequency. The note is the fundamental musical unit that associates a pitch's classification or name with its frequency (Chase, 2020).

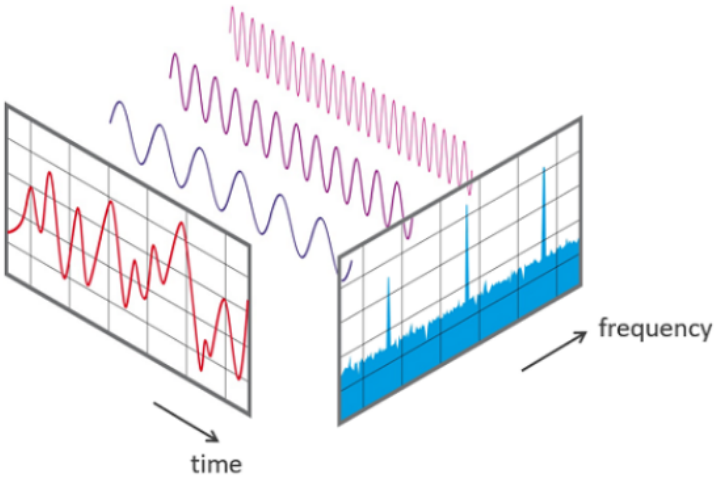
FILTER OR SYSTEM SPECIFICATION

Fast Fourier Transform

The FFT is a critical measurement method in the science of audio and acoustic measurement. As it breaks down a signal into individual spectral components, it provides frequency information about the signal (*Fast Fourier Transformation FFT*, n.d.). It is an O(N*log (N)) efficient algorithm for computing the Discrete Fourier transform as it employs a divide-and-conquer algorithm (Maklin, 2019).

Figure 1

Signal in the time and frequency domain



Note. From "Fast Fourier Transform FFT", n.a, n.d, NTI Audio. Retrieved from https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft

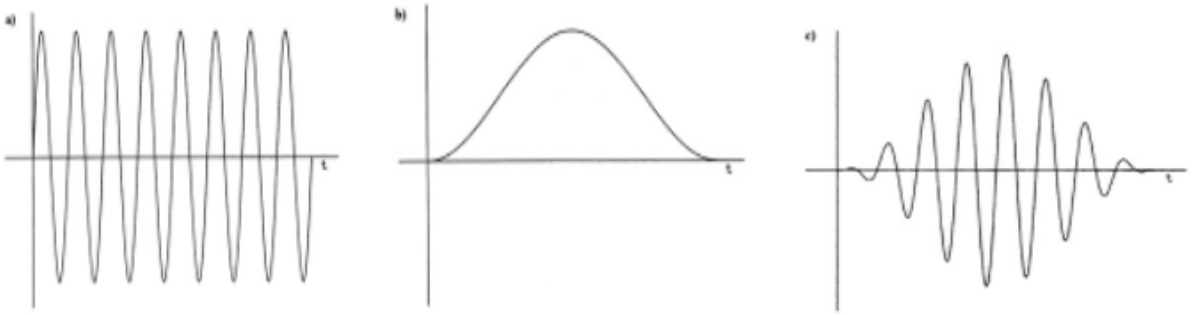
Hanning Window

A Hanning window or the apodization function is often utilized to reduce the discrete Fourier transforms leakage (Weisstein, n.d.). The main advantage of controlling leakage is that it broadens the dynamic range of the analysis, as leakage can swamp signal components with low frequencies and small magnitudes. This is highly recommended for signals with closely spaced

line spectra (Braun, 2001). As a result, the HPS will make use of the window function in the implementation of the code.

Figure 2

Hanning Window (Witte, 2001)

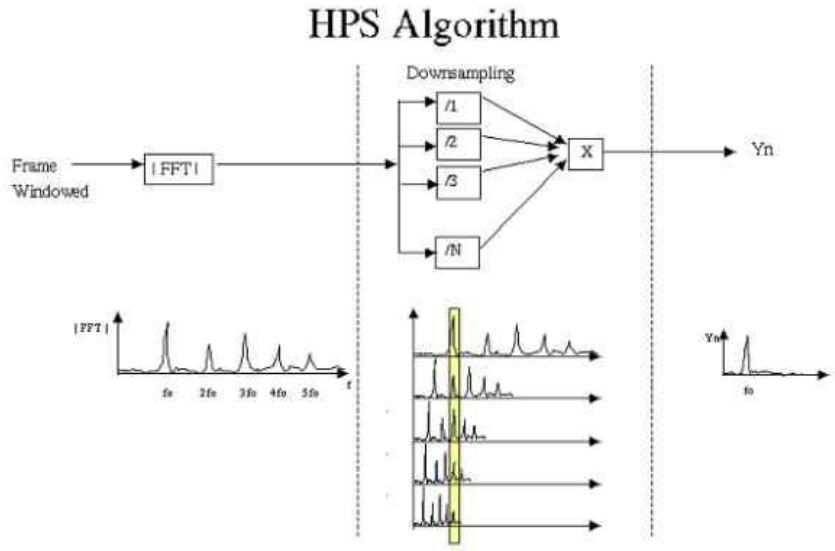


Note. a) represents the time record, b) represents Hanning Window, c) represents windowed time record. From "Hanning Window" by Witte, 2001, Spectrum and Network Measurements. (https://www.globalspec.com/reference/77966/203279/4-10-hanning-window)

Harmonic Product Spectrum

Figure 3

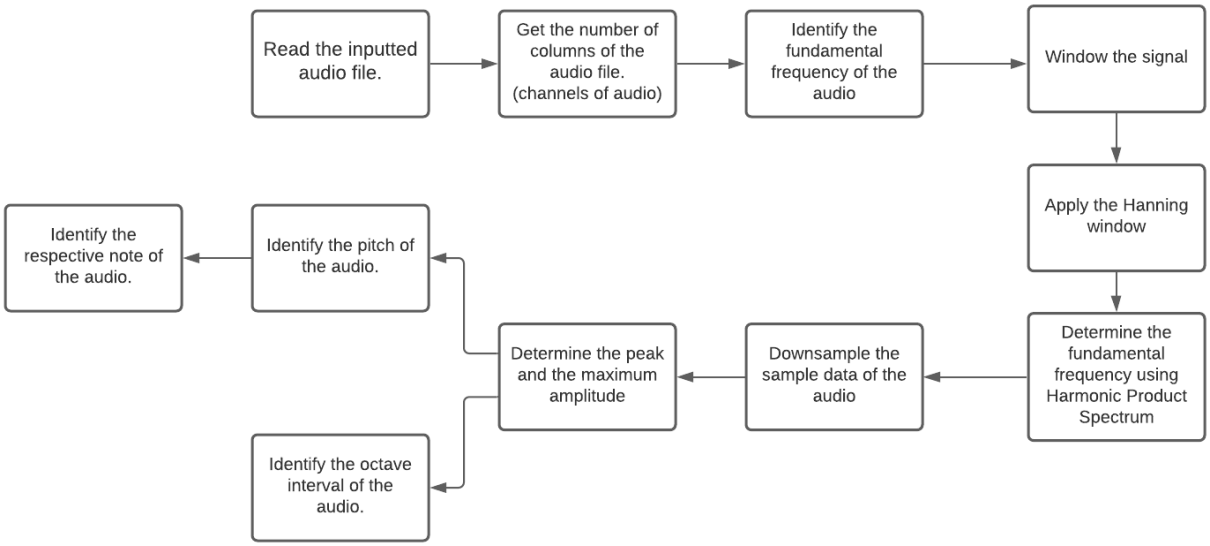
HPS Algorithm (Middleton, 2003)



Note. From "Pitch Detection Algorithms" by Middleton, 2003, ECE 301 Projects Fall 2003, (https://cnx.org/contents/i5AAkZCP@2/Pitch-Detection-Algorithms). CC BY 1.0.

The HPS method is utilized if the goal is to find the fundamental frequency of a music note as the input signal since its spectrum contains harmonics, which are the whole number multiples of the fundamental frequency represented in the series of peaks. As Figure 3. shows, the method is to divide the signal into sections by utilizing the Hanning window. Every window will have a Short Time Fourier Transform (STFT) of converting the time domain to the frequency domain. In this case, the Fast Fourier Transform (FFT) was utilized instead since the STFT has a limited number of discrete frequencies, therefore, a longer FFT was utilized to get a higher resolution in the result (Middleton, 2003). Each window's spectrum will be compressed twice by resampling; for example, if it is compressed twice, it will then go through two iterations, compressing the spectrum by two in the first iteration and three in the second. The input signal will be downsampled N amount of times of its iteration (Middleton, 2003). The N is determined by the spectrum's pitch, its lowest frequency or pitch, and the length of the FFT. If the number of downsamples or N is large in the spectrum's downsampling, the low pitch's overtone can end in the very same bin. In this case, if N is too little, it may not be able to catch harmonics of higher frequency which have a fraction of the energy of a pitch spectrum (Nicholson, 2013). In this case, the project chose to compress by 5 times. After that, multiply the spectra together, and the output is the peak value which is the window's fundamental frequency (Middleton, 2003).

SYSTEM BLOCK DIAGRAM



Note. System block diagram by Agulto & Noroña, 2021. Figure made in lucidchart.

DESIGN METHODOLOGY

The input signals would be all 12 notes including the semitones from octave range intervals 3 to 5 starting at C4 as the midpoint. The program can go as far as any octave interval, but to set constraints due to multiple inputs, only notes from octave intervals 3 and 5 were chosen to test if the program can detect the note of a different interval. The source for the piano sounds would be from a website: MediaFire (n.d.). The respective name files in the attached folder are formatted to “a3.mp3” to “b5.mp3” with a dash in the center of the text file to indicate a semitone; for instance, the C#4 audio files are named “c-4.mp3”. In case an audio file is two-channel or stereo audio, there is a segment in the code that will remove the second column of the vector to make it mono-channel since the data that would be read should only contain one column. The chosen N signal length is 8192 to allocate a higher approximation length for the windowing of the signal. After that, the Hanning function will be applied, and the FFT of the windows will be halved and passed to the HPS function. The HPS will then downsample the signal by 5 times, then multiplying the spectra together. It will then contain the peak values of the original signal, and match the frequency found in the downsampled. Lastly, it will be converted to the fundamental frequency, then find its respective classified note, along with its octave interval.

MATLAB IMPLEMENTATION AND CODE

It is expected to display the correct note and frequency from the inputted audio file. Since the HPS was beyond the scope of the class lessons, a search for an example of an HPS program was necessary for the basis of the code structure which is from Ederwander (2013) and is also attributed within the m-script file. Simultaneously, the feedback for the code's implementation stated that it did not work successfully for the user. Despite all that, the code was still used as the foundation. There were also codes for the HPS from other sources: they had similar but different implementations and still somewhat represented the original algorithm, but the validity or success of their program remained uncertain. The difference between the implemented code is that the program has pitch, note, and octave functions, with its HPS modified into having no threshold factor for fixing the octave error and no correct Factor for increasing results. Another step is to check the validity of the algorithm by using the HPS system block diagram and description by Middleton (2003), which does not show any parameters such as threshold and correct factor in the HPS algorithm by Ederwander (2013).

MATLAB implementation

1. Read audio file
    - In the command window, the user should enter the file name, including the file extension. If the file name is found, the audio data is obtained using the function **audioread()**.
    - Syntax: **[y, Fs] = audioread(filename)** reads data of the file path and returns sampled data, y, as well as the sample rate for that data, Fs. (*Read Audio File - MATLAB Audioread*, n.d.)
2. Determine the fundamental frequency
    - This determines the fundamental frequency of the audio file that would distinguish the instrument being played. Using the sample data and sample rate (y, Fs) to get the fundamental frequency.
    - Syntax: **fundamental_frequency = ff(y, Fs)**, the parameters are the sample data and sample rate (y, Fs) and returns the fundamental frequency.

i. Window the signal
   - Frame the signal to a certain length to remove extra information.
   - Syntax: **window = y(1:N)**

ii. Apply Hanning to window
   - Hanning functions as a data procession of the window for HPS.
   - Syntax: **window = window .* hann(W)**

iii. Fast Fourier Transform
   - Get the frequency domain of the input signal and get the components.
   - Syntax: **newFourier = fft(window, N)** the n-point DFT is returned. Y is the same size as X if no value is specified (*Fast Fourier Transform - MATLAB Fft*, n.d.)**.**

iv. Harmonic Product Spectrum
   - The method is used to find the fundamental frequency of the note as input signal. The sample data was down sampled 5 times and will then be multiplied by each other in order to get the maximum peak value of the downsampled with the peaks of the original signal.
   - Syntax: **fundamental_frequency = hps(newFourier, Fs)**

3. Determine the pitch
   - Using the value of the fundamental frequency, it will then determine the pitch depending on the given range.
   - Syntax: **pitch = findPitch(F)**
     i. Pitch = 1: **F >= 261.63 && F < 277.18**
     ii. Pitch = 2: **F >= 277.18 && F < 293.66**
     iii. Pitch = 3: **F >= 293.66 && F < 311.13**
     iv. Pitch = 4: **F >= 311.13 && F < 329.63**
     v. Pitch = 5: **F >= 329.63 && F < 349.23**
     vi. Pitch = 6: **F >= 349.23 && F < 369.99**
     vii. Pitch = 7: **F >= 369.99 && F < 392.00**
     viii. Pitch = 8: **F >= 392.00 && F < 415.30**

ix. Pitch = 9: **F >= 415.30 && F < 440.00**

x. Pitch = 10: **F >= 440.00 && F < 466.16**

xi. Pitch = 11: **F >= 466.16 && F < 493.88**

xii. Pitch = 12: **F >= 493.88 && F < 523.25**

4. Determine the respective note
    - Finds the classification of the fundamental frequency of the signal given the parameter pitch it will then return the note.
    - Syntax: **note = detectNote(P)**
        i. Pitch = 1: **note = "C"**
        ii. Pitch = 2: **note = "C#"**
        iii. Pitch = 3: **note = "D"**
        iv. Pitch = 4: **note = "D#"**
        v. Pitch = 5: **note = "E"**
        vi. Pitch = 6: **note = "F"**
        vii. Pitch = 7: **note = "F#"**
        viii. Pitch = 8: **note = "G"**
        ix. Pitch = 9: **note = "G#"**
        x. Pitch = 10: **note = "A"**
        xi. Pitch = 11: **note = "A#"**
        xii. Pitch = 12: **note = "B"**
5. Determine the octave interval
    - Get the note's specific interval from the octave scale given the frequency and octave (default interval) it will then return the octave of the audio file.
    - Syntax: **octave = oct(F, O)**, uses the frequency and the default interval is 4 which is passed in the parameter O in its first call and returns the octave.

Code

Main script (flow of the program)

```
% Script for the flow of the program.

% Authors: AGULTO, Juliana Marie B. & NORONA, Yeohan Lorenzo M.
% Course: CEDISP1 S11
% Date: 2021, May

clear all
clc

% Source (Audio file): https://www.mediafire.com/file/zdlmqtazulgv28a/mp3_Notes.rar/file
isRun = true;

while isRun

    isFileName = true;

    disp('Instruction: ');
    disp('   Enter file name with its file extension.');
    disp('   Type `exit.` to close the program.');
    disp('- - - - - - - - - - - MUSIC NOTE DETECTOR - - - - - - - - - - -');
    while isFileName
      filename = input('File name: ', 's'); % File name of the audio
      filepath = strcat('../mp3 Notes/', filename);  % Adds the file path of the mp3 Notes to the filename
      checkRun = strcmp(filename, 'exit.');  % Compares the String of the filename with `exit`

      if exist(filepath, 'file') && ~exist(filepath, 'dir') && checkRun == 0 % To check if file name exist in the folder and file name is not `exit`
        isFileName = false;
        %---
        [y,Fs] = audioread(filepath); % read audio file
        sound(y, Fs);   % simple playback of the audio file
        [~,n] = size(y); % get the number of columns (channels of audio)

        %-- To plot the audio file --
        subplot(3,1,1), plot(0:1/Fs:(length(y(:,1))*1/Fs)-1/Fs, y(:,1)), xlabel('Seconds'), ylabel('Amplitude'), title("Signal of the audio file");

        if n == 2 % This program only works for monophonic / not stereo channels
            y = y(:, 2); % Converts to mono channel (remove 2nd column / channel)
        end

        F = ff(y, Fs); % Gets the fundamental frequency
        pitch = findPitch(F); % Find the Pitch
        note = detectNote(pitch); % Find the respective note
        octave = oct(F,4); % Find the octave interval
        disp("   Note Detected: " + note);
        disp("   Octave Scale: " + octave);
      elseif checkRun == 1  % To check if the user input is `exit`
          isRun = false;
          isFileName = false;
      else
        disp('Sorry, file name not found.');
      end
    end
    disp('- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -');
    fprintf('\n\n'); % New line
end
```

oct function

```
% Script for detecting the octave.

function octave = oct(F, O)
% Function: oct - Musical notes including the semitones (sharps / flats)
% Parameters: (F) - frequency, (O) -  default 4 in its first call as the default interval or center
% Returns: octave
    if (F < 261.63) % just like pitch func, recursively call the function, )
        F = F*2; % except octave is (4-N), for N increases every call after its first call
        O = O-1;
        octave = oct(F, O);
    elseif (F >= 523.25) % the B4 covers the first half of the midpoint between C5 and itself
        F = F/2;
        O = O+1;  % just like pitch func, recursively call the function, )
        octave = oct(F, O);  % except octave is (4-N), for N increases every call after its first call
    else
        octave = O;
    end
end
```

hps function

```
% Script for the Harmonic Product Spectrum.

% This code was adapted from "MATLAB code for Harmonic Product Spectrum" by Ederwander
%     Title: MATLAB code for Harmonic Product Spectrum
%     Author: ederwander
%     Date: 2013, November 5
%     Code version: 3.0
%     Availability: https://stackoverflow.com/questions/19765486/matlab-code-for-harmonic-product-spectrum
%     License: CC BY-SA 3.0 (Free to Share & Adapt)

function HPS = hps(y, Fs)
% Function: hps
% Parameter: (y) - sample data, (Fs) - sample rate
% Return: HPS
    ds_1 = downsample(y, 1);  % downsample by 1
    ds_2 = downsample(y, 2);  % downsample by 2
    ds_3 = downsample(y, 3);  % downsample by 3
    ds_4 = downsample(y, 4);  % downsample by 4
    ds_5 = downsample(y, 5);  % downsample by 5

    N = length(ds_5); % get the lowest length since error will produce for a length higher than HPS
    y = []; % array for containing the multiplication of the downsamples
    % get the length of the smallest downsample, to prevent indexing errors
    for x = 1 : N
        y(x) =  ds_1(x) * ds_2(x) * ds_3(x)* ds_4(x) * ds_5(x);
    end
    % loop multiplying the downsamples
        % ignore row, get the peaks in the second parameter
    [~, n] = findpeaks(y, 'SortStr', 'descend'); % insert ~ in the first param to ignore row

    % Peak sorting by descendance, largest values first to get the first few
    % peak values
    Maximum = n(1); % get the first as a harmonic or f0

    subplot(3,1,3), plot(y), xlabel('Seconds'), ylabel('Amplitude'), title("Harmonic Product Spectrum");

    HPS = ( (Maximum*Fs) / 8192 );  % calculate the frequency Max Value * Sampling Rate / Signal Length
end
```

findPitch function

```
% Script for finding the pitch.

function pitch = findPitch(F)
% Function: findPitch - find the pitch [numbers correspond to the cases in function note()]
% Parameter: (F) - frequency, from the (f0) - HPS
% Returns: pitch
    if (F >= 261.63 &&  F < 277.18)
        pitch = 1;
    elseif (F >= 277.18 && F < 293.66)
        pitch = 2;
    elseif (F >= 293.66 && F < 311.13)
        pitch = 3;
    elseif (F >= 311.13 && F < 329.63)
        pitch = 4;
    elseif (F >= 329.63 && F < 349.23)
        pitch = 5;
    elseif (F >= 349.23 && F < 369.99)
        pitch = 6;
    elseif (F >= 369.99 && F < 392.00)
        pitch = 7;
    elseif (F >= 392.00 && F < 415.30)
        pitch = 8;
    elseif (F >= 415.30 && F < 440.00)
        pitch = 9;
    elseif (F >= 440.00 && F < 466.16)
        pitch = 10;
    elseif (F >= 466.16 && F < 493.88)
        pitch = 11;
    elseif (F >= 493.88 && F < 523.25)
        pitch = 12;
    else
        if (F < 261.63) % if frequency is lower than the default octave 4,
            F = F * 2; % recursively call the function while doubling since
            pitch = findPitch(F); % its double is the same note, just higher octave
         else
             F = F / 2; % recursively call the function while halving since
             pitch = findPitch(F); % its half is the same note, just lower octave
         end
     end
end
```

ff function

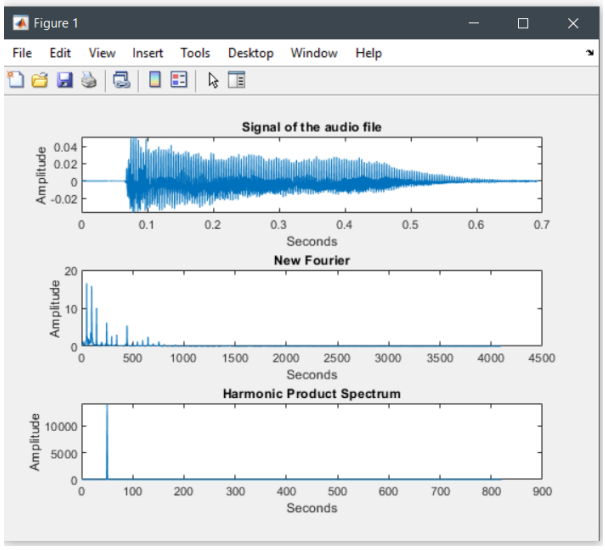
```
% Script for the Fundamental Frequency.

% This code was adapted from "MATLAB code for Harmonic Product Spectrum" by Ederwander
%     Title: MATLAB code for Harmonic Product Spectrum
%     Author: ederwander
%     Date: 2013, November 5
%     Code version: 3.0
%     Availability: https://stackoverflow.com/questions/19765486/matlab-code-for-harmonic-product-spectrum
%     License: CC BY-SA 3.0 (Free to Share & Adapt)

function fundamental_frequency = ff(y, Fs)
% Function: ff - Gets the fundamental frequency
% Parameters: (y) - sample data, (Fs) - sample rate
% Return: fundamental_frequency
    N = 8192; % CHOSEN SIGNAL LENGTH
    window = y(1:N); % window the signal
    W = length(window); % get the length
    window = window .* hann(W); % apply hanning to window
    newFourier = fft(window, N); % Fast Fourier Transform of the Window
    newFourier = newFourier(1 : size(newFourier, 1) / 2); % Half the DFT
    newFourier = abs(newFourier); % absolute value
    fundamental_frequency = hps(newFourier, Fs); % call function for Harmonic Product Spectrum

    %-- Plot --
    subplot(3,1,2), plot(newFourier), xlabel('Seconds'), ylabel('Amplitude'), title("New Fourier");
end
```

detectNote function

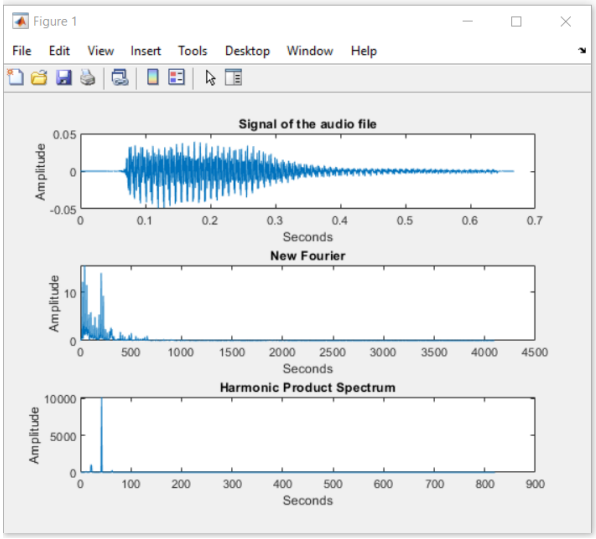
```
% Script for detecting the note.

function note = detectNote(P)
% Function: detectNote - Musical notes including the semitones (sharps / flats)
% Parameter: (P) - pitch from findPitch(F)
% Returns: note
    switch (P)
        case 1
            note = "C";
        case 2
            note = "C#";
        case 3
            note = "D";
        case 4
            note = "D#";
        case 5
            note = "E";
        case 6
            note = "F";
        case 7
            note = "F#";
        case 8
            note = "G";
        case 9
            note = "G#";
        case 10
            note = "A";
        case 11
            note = "A#";
        case 12
            note = "B";
    end
end
```

Test cases

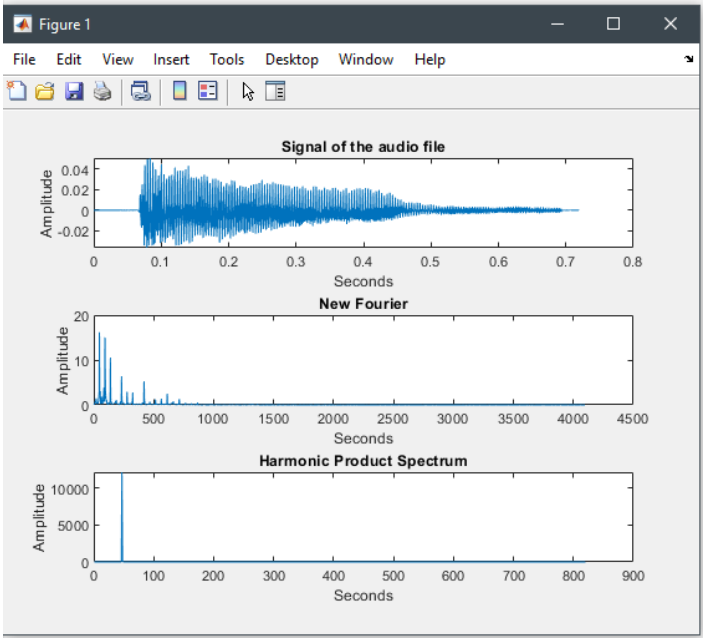
```
Instruction:
   Enter file name with its file extension.
   Type `exit.` to close the program.
- - - - - - - - - - - MUSIC NOTE DETECTOR - - - - - - - - - - -
File name: c4.mp3
   Note Detected: C
   Octave Scale: 4
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```




```
Instruction:
   Enter file name with its file extension.
   Type `exit.` to close the program.
- - - - - - - - - - - MUSIC NOTE DETECTOR - - - - - - - - - - -
File name: a3.mp3
   Note Detected: A
   Octave Scale: 3
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```




```
Instruction:
   Enter file name with its file extension.
   Type `exit.` to close the program.
- - - - - - - - - - - MUSIC NOTE DETECTOR - - - - - - - - - - -
File name: b4.mp3
   Note Detected: B
   Octave Scale: 3
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```



RESULTS AND DISCUSSION

Table 2

Results of matched/mismatched notes and octave intervals from the program

Notes (Given)	Note (Output)	Octave (Output)
C3	C	3
C#3	C#	3
D3	D	3
D#3	D#	3
E3	E	3
F3	F	3
F#3	F#	3
G3	G	3
G#3	G#	3
A3	A	3
A#3	B	2
B3	B	3
C4	C	4
C#4	C#	4
D4	D	4
D#4	D#	4
E4	E	4
F4	F	4
F#4	F#	4
G4	G	4
G#4	G#	4

A4	A	3
A#4	A#	3
B4	B	3
C5	C	5
C#5	C#	5
D5	D	5
D#5	D#	5
E5	E	5
F5	F	5
F#5	F#	4
G5	G	5
G#5	E	3
A5	A	4
A#5	A#	4
B5	B	4

Note. Equal means either the given note and the detected note is matched. It applies to the octave interval as well.

Table 3

Result of the correctness of the program

	EQUAL	NOT EQUAL	TOTAL	PERCENTAGE
NOTE	34	2	34 / 36	94.44%
OCTAVE	27	9	27/ 36	75%
BOTH	26	2	26 /36	72.22%

Note. The total category is the number of equals over 36 (total notes). Both rows are both note and octave being correct together.

According to Table 3, the correctness of the notes is 94.44% (34/36), while the octave is 75% (27/36). There are 8 correct notes with incorrect octave intervals, and 1 mismatch for both note and octave. According to Table 2, only two of the 36 notes had a detection mismatch, and it was found in G#5 giving detection of an E3 note. As stated previously, the low pitches may be improperly tracked (Middleton, 2003), so if notes from the even lower octave than interval 3 were tested, there might be frequent mismatches in the notes. The notes with wrong octave intervals are probably the harmonics of the fundamental frequency. It was also stated that the fundamental frequency is frequently absent, particularly in pitched sources of sound, thick string instruments, and lower or male voices in general (Nicholson, 2010). As such, it is for this reason that different algorithms, such as the HPS, are used in a program like this. There were almost no correct notes before the application of the HPS. In comparison to its previous result, the music detection's total correctness is now 94.22% higher.

The limitation of this program is that it used audio files that did not contain noise such as the background noise, and the sounds tested are probably from a digital piano or synthesizer. It is possible that using recorded live audio of instruments played in any device may factor in the determination of the frequency. Another limitation is the consideration of timbre in this program since it is the one that makes the distinction between different instruments despite playing the same note; reading audio files of a note played from any instrument other than the digital piano might release different results. Although there were very few audio files of a guitar tested in the program, some were able to match both the notes and the intervals. To be clear, this success could have happened by chance, and the program is ostensibly aimed solely at the instrument piano; the program has not taken into account other piano instruments such as grand pianos, digital pianos, and many others. In theory, if background noise in the input signal is ignored, the program could work for any instrument because the HPS is there to deal with harmonics; as previously stated, harmonics or overtones are what distinguishes the sound of different instruments even if they all played the same note (Walker, 2020).

The most challenging part was implementing the HPS because it falls outside of the modules covered in class. Many sources contain HPS codes, but the vast majority of them are incomplete or invalid, and some are in programming languages such as Python or Java. Ederwander (2013) created a surface-level example of the HPS for the foundation of the code in

which was attributed within the m-script file, it was completed and implemented very close to its algorithm based on the description of its implementation and the HPS system block diagram (Middleton, 2003). Other users' HPS implementations were also checked for the validity of their structure and algorithm of the program modified for the project's target, including the mentioned attributions. Although it was able to get at least many correct notes and two-thirds of the note's octave is correct. It may not be perfect, but it follows the closest to the description of the algorithm.

At least 34 of the 36 notes were correctly identified by the program, with some having an offset in their octave interval detection. If the octave deviates even further below the scale than it does above, the program's note mismatch may become apparent. There may be discrepancies in the results if real-time tuning is used and a repetitive live recording of an instrument playing a specific note is used. If the same audio file with the same note is used, the program will continue to produce the same fundamental frequency. Finally, the program captured more than 94% of the music notes from octave intervals 3-5, with some incorrect octave outputs having an offset value of 1 either above or below the scale.

SUMMARY AND CONCLUSION

Without the use of specific, advanced concepts such as the HPS, determining the fundamental frequency of a sound is difficult. A simple FFT without other signal processing algorithms will most likely result in an error. Sounds produced by string instruments, such as the piano or guitar, still have harmonics to deal with and a slew of other factors, particularly those related to music and physics, must be considered. At the same time, the fundamental frequency may not even be detected or may become missing. The software MATLAB has given the ease of implementing such a program for musical note detection systems. It is suggested that in future research and projects involving concepts such as musical notes or fundamental frequency detection systems, efficient and effective algorithms other than the HPS be used to test whether there is a significant difference in comparative results or to implement even better programs using HPS. To summarize, musical note detection has many factors to consider, particularly in its implementation.

REFERENCES

Braun, S. (2001). WINDOWS. In S. Braun (Ed.), Encyclopedia of Vibration (pp. 1587–1595). Elsevier. https://doi.org/10.1006/rwvb.2001.0052

Ederwander. (2013, November 5). *MATLAB code for Harmonic Product Spectrum.* Retrieved from https://stackoverflow.com/questions/19765486/matlab-code-for-harmonic-product-spectrum

Fast Fourier transform—MATLAB fft. (n.d.). Retrieved May 20, 2021, from https://www.mathworks.com/help/matlab/ref/fft.html

Maklin, C. (2019, December 30). *Fast Fourier Transform.* Retrieved from https://towardsdatascience.com/fast-fourier-transform-937926e591cb#:~:text=As%20the%20name%20implies%2C%20the,)%20to%20O(NlogN)%20.

MediaFire. (n.d.). Mp3 Notes. Retrieved May 16, 2021, from https://www.mediafire.com/file/zd1mqtazulgv28a/mp3_Notes.rar/file

Middleton, G. (2003, December 18). *5.4 Pitch Detection Algorithms.* Retrieved from https://cnx.org/contents/i5AAkZCP@2/Pitch-Detection-Algorithms

Nicholson, R. (2010, November 19). *MATLAB - Missing fundamental from an FFT [closed].* Retrieved from https://stackoverflow.com/questions/4227420/matlab-missing-fundamental-from-an-fft

Nicholson, R. (2013, November 3). *How does one know how many times to downsample in Harmonic Product Spectrum?.* Retrieved from https://stackoverflow.com/questions/23369742/stft-fft-work-flow-order#:~:text=A%20single%20FFT%20is%20used,frequency%20spectrum%20of%20a%20signal.&text=The%20STFT%20is%20used%20when,event%20occurs%20in%20the%20signal.

Rauch, S. (2017. January 3). *What is the purpose of a window function?* Retrieved from https://dsp.stackexchange.com/questions/36649/what-is-the-purpose-of-a-window-function

Read audio file—MATLAB audioread. (n.d.). Retrieved May 15, 2021, from https://www.mathworks.com/help/matlab/ref/audioread.html

Suits, B. H. (1998). *Frequencies of Musical Notes*. Retrieved from https://pages.mtu.edu/~suits/notefreqs.html

Towell, G. (2020, December 28). Overtone & Harmonics (Physics): Definition, Differences & Frequencies. Retrieved from https://sciencing.com/overtone-harmonics-physics-definition-differences-frequencies-13722353.html

Braun, S. (2001). WINDOWS. In S. Braun (Ed.), *Encyclopedia of Vibration* (pp. 1587–1595). Elsevier. https://doi.org/10.1006/rwvb.2001.0052

Fast Fourier Transformation FFT. (n.d.). NTI Audio. Retrieved May 23, 2021, from https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft

Fast Fourier transform—MATLAB fft. (n.d.). Retrieved May 20, 2021, from https://www.mathworks.com/help/matlab/ref/fft.html

Read audio file—MATLAB audioread. (n.d.). Retrieved May 15, 2021, from https://www.mathworks.com/help/matlab/ref/audioread.html

Walker, L. (2020, May 6). *Fundamental and Harmonic Frequencies.* Retrieved from https://www.teachmeaudio.com/recording/sound-reproduction/fundamental-harmonic-frequencies

Weisstein, E. (n.d.).“Hanning Function.” Retrieved from https://mathworld.wolfram.com/HanningFunction.html

Witte, R. (2001). Spectrum and Network Measurements. Retrieved from https://www.globalspec.com/reference/77966/203279/4-10-hanning-window