EECE 301 Project
DFT-Based Guitar Tuner

Due Date

11:59 PM on Monday 11/25/2024.

- **Note: given that you have 4 weeks to work on this, I will not accept any late submissions!**

ASSIGNMENT

- **Perform the steps outlined in the "Procedure" section.**
- **Submit via BS a report (as a PDF) on your approach and your results (see details below)**
- **Submit via email your code (do NOT include your code in your report!)**

REPORT DETAILS & SUBMISSION

Electronically submit a written report (as a PDF) by the due date/time that discusses the results of your work to address the tasks described in the Procedure below.

There will be a TurnItIn assignment link created in Brightspace for the purpose of submitting your report. (Do not include your code in this submission – see below)

Turnitin will check your submission for plagiarism against others in the class as well as online and book sources. You should read the following link to make sure you understand what is considered to be plagiarism: Link to Academic Honesty Policy.

Plagiarism of *Code*: You are expected to write your own code! You may not get code from other students… even getting help from other students on how to write the code is considered plagiarism of code! You may not get code from online sources (other than the examples I have posted for the course); obviously this does not include using the help facility of MATLAB or websites that explain how to use certain commands.

******If you need help with your code, please direct your questions to Prof. Fowler!**
*****Do not seek help from the TA or UCAs… they have not been briefed on how to do this project!**

The format of the report should be as follows:

- **Include an Introduction section that provides an overview of the report**
- **Include a separate section for each of the "Tasks" of the procedure given below. These sections should be titled to match the titles of the Tasks given below. These sections should discuss the steps performed and results obtained during the corresponding part of the procedure.**
  - ***Use Concepts & Theory from class to support and/or explain what you observed.***
  - **Make sure you discuss HOW your processing works, WHY it works, and discuss HOW your results support the claim that it works.**
- **Include a Conclusions section that summarizes your results.**
  - **Use concepts and methods from the course to summarize your results and describe how/why they match (or do not match) expectations.**
  - ***Use class theory to support and/or explain what you observed*.**

  - **Don't tell me that you "saw how good MATLAB is" or any other trite thing like that…**
    - **Summarize the results you saw and the insights your results provide about the tasks at hand.**
  - **A common error is for students to write a very minimal summary section that does not really say much. Avoid that!**
  - **Do NOT just give an overview of WHAT you did**
- **Write your report NOT as if you are writing to a professor but rather like you are writing to your co-workers and manager to explain the essence of what you did, what your results were, why they seem reasonable, and what they mean.**
  - **Don't give details on exactly what was done (in other words, avoid a step-by-step "travelogue" discussion such as "First we did this and then we did that"… that is just regurgitating the procedure that I gave you… discuss what the results show and what they mean… discuss them in terms of the theory we studied in class.**
- **Make your report look professional!!**
  - **Use the MS word equation editor to make professional looking equations**
    - **It is pretty easy to use – ask me for help if you need it!**
  - **Use equation numbers for any equation set off from the text (follow my example shown in the rest of this document)**
  - **Use figure numbers and captions and refer to figures by number rather than using something like "… the figure below…"**
    - **Embed the figures into the report where they are discussed – not at the end!**
    - **See the MATLAB tutorial I posted for a discussion of how to paste a MATLAB figure into a document.**
  - **Yes… spelling and grammar count! Clarity of writing also counts!**
    - **Write clearly so that your reader can understand your writing.**
    - **Be careful when using pronouns such as "it", "this", "that", etc. so you can be sure your reader knows precisely to what you are referring!**
      - **Instead of writing "It shows…." say something like "The DFT computed using this approach shows…"**
- **Do NOT include your MATLAB code in the report – see below for how to submit that.**

CODE DETAILS & SUBMISSION

- **Make sure your code is *well-commented* so I can easily determine what you are doing and why.**
- **Make sure your submitted m-file adheres to the "proper style" outlined in the posted MATLAB tutorial material – see "Note_Set_M05 - Scripts & Functions"**
- **Email your code with the subject *exactly* like this ("*exactly*" means the spacing, too!):**
  EECE301 Project Code
- **Send your code to mfowler@binghamton.edu**
  - **In the body of the email put only your name**
  - **Attach your m-files to the email**
    - **Do NOT put them into a zip folder or any other equivalent form!!**

****

OBJECTIVE

Explore the use of the discrete Fourier transform (DFT) for the implementation of a guitar tuner. This exploration should include proper interpretation of the computed results and how the DFT results are related to the DTFT of the signal.

BACKGROUND

The objective of this project is to develop a MATLAB-based simplified version of a guitar tuner. A guitar tuner accepts a signal from a guitar (either directly through a cable or through a built-in microphone). If that signal consists of a single string being plucked, then the tuner will indicate which standard note that string is closest to and what direction the string must be tuned to be at the correct frequency. The foundation of this processing is the ability to measure (really, to estimate) the fundamental frequency of signal. This project ignores all the user-interface aspects of telling the user what direction to tune and what note is closest.

To understand the performance of a guitar tuner it is important to know that the accuracy of tuning is measured in a unit called "cents". In music, the interval of an octave means two notes whose fundamental frequencies form a ratio of 2. In Western music there are 12 notes (also called semitones) in an octave and the spacing between each semitone is divided into 100 cents – so there are 1200 cents in an octave. This relationship says that two frequencies f_2 and f_1 are C cents apart when the following is satisfied:

$$\frac{f_2}{f_1} = 2^{C/1200} \tag{1}$$

Note that C is positive if $f_2 > f_1$ and is negative if $f_2 < f_1$. If f_1 is the target frequency of a tuner and f_2 is an estimate of f_1, then the C value that satisfies equation (1) gives the error in the units cents. An accuracy of 5 cents for guitar tuners is considered just OK and many claim to be more accurate than that! But for our work here we'll aim for 5 cent accuracy.

There are many aspects of a tuner that we are NOT considering here. For example, we are not considering the analog electronics needed to get the signal into the form appropriate to provide it to the ADC (this includes any anti-aliasing filters). We are also not considering any digital processing that will detect *when* a note is plucked and then automatically select the section of the signal for processing. Nor are we considering that a tuner is constantly updating its frequency estimate as time goes on – we are assuming that we get one chunk of signal data and we make one estimate from it. So, this is a very stripped-down version of a tuner – but it includes some of the key aspects of what you might find at the heart of a real tuner.

Comment on MATLAB functions: All MATLAB functions that are created that take inputs should accept the inputs as "passed input variables" variables and NOT as prompted input variables. Similarly, output variables should be "passed output variables". In other words, if a function called my_stuff.m takes in input variables called In_1 and In_2 and output variables Out_1 and Out_2 they should be passed as follows:

```
>> [Out_1,Out_2] = my_stuff(In_1,In_2);
```

For all of this project we will assume a sampling rate of Fs = 4,410 Hz (and that is NOT a typo… it is NOT the more standard 44.1kHz value often used for audio).

PROCEDURE:

Part I. DFT of Single Sinusoid and the Use of the Hamming Window

Create a MATLAB function (not a script!) called DFT_1_Sine.m that does the processing described below. It should take in the following input variables: f1 in Hz, N in samples, Nzp in samples. It should do the following:

1. Create a row vector called x having N samples taken at a rate of Fs = 4410 Hz of a sinusoid with frequency of f1 Hz, amplitude of 1 and phase of 0.
2. Compute the DFT of this sinusoid in x using zero-padding out to the value Nzp and plot its magnitude in dB versus frequency in Hz. This plot should be put in the top of two subplots.
3. Now create the "Hamming window" of length N using the Hamming command as follows: w = hamming(N);
    a. Note that this creates a column vector whereas your x vector is a row vector
    b. So convert w into a row vector (there are several ways you can do this!)
4. In a separate figure plot your Hamming window versus a time index vector n
5. Now create the "windowed" signal xw by multiplying element-wise x times w
6. Compute the DFT of this windowed sinusoid in xw using zero-padding out to the value Nzp and plot its magnitude in dB versus frequency in Hz. This plot should be put in the bottom of two subplots, under the plot of the DFT of the non-windowed sinusoid.

Now explore the results of this function for a few values of f1 and N (always choosing Nzp to give an appropriately fine grid for the case you are considering). Show a few different representative results in your report; at least one should show the full axis range from the plot command but the rest should be zoomed in to better show the result (this zoom can be done by hand and need not be part of your function).

Use your knowledge of the relationship between CTFT, DTFT and DFT to explain what you would expect to see when you compute the DFT of a block of samples taken from a CT sinusoid. Are you seeing what you expect? Are your results valid? How do your DFT plot results change as you increase the number of samples?

Compare and contrast the DFT result for the non-windowed sinusoid versus the windowed version. What effect does the window have on the smearing effect? In particular, what effect does it have on the "mainlobe" and the "sidelobes" of the DFT result?

Part II. DFT of Two Sinusoids

Create a MATLAB function (not a script!) called DFT_2_Sine.m that does the processing described below. It should take in the following input variables: f1 and f2 in Hz, *a* in volts, N in samples, Nzp in

samples.

This function should do the same as DFT_1_Sine.m except now it should have two sinuosids: one with frequency of f1 and amplitude of 1 and one with frequency of f2 and amplitude of *a*.

Explore the results of this function for a few values of f1,f2, *a* and N (always choosing Nzp to give an appropriately fine grid for the case you are considering). Note that we can always choose $a < 1$. Show a few different representative results in your report; at least one should show the full axis range from the plot command but the rest should be zoomed in to better show the result (this zoom can be done by hand and need not be part of your function).

Does the windowing make it easier or harder to "see" two sinusoids that are close together in frequency? Does the windowing make it easier or harder to "see" two sinusoids that are fairly well-separated in frequency but one has a very small amplitude relative to the other?

Part III. DFT of Individual Real Guitar Notes

The above two sections provide insight that is helpful for the remaining sections.

There are three .WAV files that have been posted that contain recordings of guitar notes: the open low E string, the open A string, and the open D string. These recordings were made with a sampling rate of 4410 samples/second. Download these three wav files and make sure MATLAB's current directory points to where you have stored these files; see the MATLAB tutorial that I posted on Blackboard for details on how to change MATLAB's current directory .

Write a MATLAB script called DFT_Real_Guitar.m that does the following:

1. Use the command audioread to load each of these files into MATLAB and call each variable a different name. For example:
    - x_E = audioread('E_String.wav');
    - Do similar commands for x_A and x_D
2. In "Figure 1"… Plot each string's time signal versus the correct time vector; put these three plots on three vertically stacked sets of axes by using the subplot command.
    - Note that the following command creates a figure window for "Figure 1" (change the number to get other figure numbers)
    - >> figure(1)

3. Extract the part of x_E that lies between t = 4 sec and t = 5 sec and store the result back in x_E (your x_E should now have 4,411 samples because it has a duration of 1 second at 4,410 samples per second.
    - Do similar things for x_A and x_D

4. In "Figure 2" plot each of these 1 second duration signal against a time vector starting at t = 4 sec and ending at 5 seconds; put these three plots on three vertically stacked sets of axes by using the subplot command.

5. Now, compute the DFT of these segments: for each string, do this with and without using the Hamming window; if needed, use appropriate amount of zero-padding.

6. For each string, use the figure command to create a new figure window and plot the dB versions of the two DFT results on two different sets of axes using subplot… put one plot above the other. Here are some other things you should do for these plots
    - Use the "axis" command to show only the range of frequencies from 0 Hz to 1000 Hz and to show an appropriate range of dB values to clearly show the structure of the DFT results. Run the axis command for the current subplot BEFORE running the NEXT subplot.

Run your script and then answer the following:

Do the plots of the time signal look somewhat periodic? (it will help to zoom in on various sub-intervals to see their behavior). We used Fourier Series to represent a periodic signal… and we used Generalized Fourier Transform to understand what the CTFT of a periodic signal should look like.

From musical physics we know what fundamental frequency each musical note has:

Table 1: Note Frequencies for the Three Bottom Strings of a Guitar

Note on Guitar Open String	**Fundamental Frequency**
E (Low)	82.41 Hz
A	110 Hz
D	146.83

So… use your knowledge of Fourier Series, Generalized Fourier Transform, and the relationship between CTFT and DTFT to explain what you are seeing in your DFT plots.

Do you see the expected structure that your theory from above predicts? Mark & label on your figures the DFT spikes that your theory predicts. (The "Insert" option on the Figure menu bar provides tools to mark your plot's features: use "Text Arrow" and "TextBox".) These should be added by hand after the program DFT_Real_Guitar.m is run rather than

Comment on the usefulness of the Hamming window in this application. Was it helpful in seeing all of the guitar signal's Fourier-series-like structure? Why or why not?

Was the guitar I played perfectly tuned to the expected Fundamental Frequencies? (Extreme zooming will help!)

Are there any other DFT spikes that your theory doesn't predict? Are there some relationships between (at least some of) these spikes that you can identify? Roughly, what does theory say about the signal that these spikes represent? What do you think is the cause of them?

Part IV. DFT of Individual Synthetic Guitar Notes

For the final evaluation of a developed DSP algorithm it is ideal to have real data available but often it is desirable during development to use synthetic (or simulated) signal data. This allows us to easily create a variety of scenarios and to know precisely what the true signal parameters are for the signal we are processing. So for our case we'd like to have a function that simulates a 1-second section of a guitar note of any fundamental frequency we specify.

I experimentally determined the approximate amplitudes for an Amplitude-Phase Fourier Series model for a typical guitar string to be

$$A_0 = 0$$
$$A_1 = 0.0700$$
$$A_2 = 0.025$$
$$A_3 = 0.013$$
$$A_4 = 0.007$$
$$A_5 = 0.003$$
$$A_6 = 0.002$$
$$A_7 = 0.0004$$
$$A_8 = 0.0003$$
$$A_9 = 0.0002$$
$$A_{10} = 0.0001$$

The amplitudes for terms for k>10 were deemed to be insignificant.

Write a MATLAB function called DFT_Synth_Guitar.m that accepts as an input the desired note fundamental frequency and generates a 1-second snippet of synthesized guitar signal at that note, as described below.

1. Since there will be 10 terms in your model (the DC term is 0 as shown above), create 10 random phases that each lie between 0 and 2π using this command:
   - phase = 2*pi*rand(1,10);

2. Use those random phases and the given amplitude coefficients to compute the amplitude-phase form of the Fourier series to generate a 1-second snippet of a guitar signal. The result should be placed in a vector called x (lowercase).

3. Compute the DFT of your synthetic guitar signal using the Hamming window and appropriate zero-padding and put the result in a vector called X (uppercase).

4. Plot the DFT result in the same manner as was done in Section III.

5. The function should provide x and X as output variables that can be passed out of the function when the function is called like this:

- [x,X] = DFT_Synth_Guitar(fo);

Try your function for various values of fundamental frequency; specifically choose those frequencies for a correctly tuned E string, A string, and D string, but try other values – including those that are not specific guitar string frequencies. Verify that your DFT results show that your function provides a reasonable simulation of a guitar string signal.

For the implementation used here for computing the simulated signal, what is the largest value you can choose for the fundamental frequency? Why?

Part V. DFT-Based Guitar Tuner

In this section you will create a MATLAB function that will process a synthetic guitar signal (from your program created in Section IV) and operate somewhat like a guitar tuner in that it will provide an estimate of the fundamental frequency of the note being played. For simplicity we will limit ourselves to a fictional "three-string guitar" that has only the bottom three strings with notes as shown in Table 1.

The function will be named DFT_Tuner.m and will be called as follows:

fo_est = DFT_Tuner(x,b_E,b_A,b_D)

where:

Inputs:

x = A vector holding a 1-second chunk of guitar signal sampled at Fs = 4410 Hz
b_E = A vector holding FIR filter coefficients for detecting the E note (described below)
b_A = A vector holding FIR filter coefficients for detecting the A note (described below)
b_D = A vector holding FIR filter coefficients for detecting the D note (described below)

Outputs:

fo_est = A scalar holding the estimated fundamental frequency in Hz of the input signal

Although it is possible to simply compute the DFT of the signal and look for the highest peak and declare its frequency to be the frequency of the note played, there can be some downsides to that brute-force approach. Thus, we will first use three filters designed to help detect if the input signal is a note within a certain range of the three desired notes to which we want to tune (E, A, D). The results from these three filters will allow us to confine our search of the DFT results to get better results.

V-A: Detection Filter Design

The goal of this sub-section is to design three FIR bandpass filters so that a note played with a fundamental frequency near one of the three note frequencies (82.41 Hz, 110Hz, 146.83 Hz) will be passed by only one of the three filters. The widths of the passbands should be chosen so that a note with fundamental frequency in the range of 68 Hz to 165 Hz will fall in the passband of the filter

corresponding to the closest correctly tuned frequency; e.g., 90 Hz should fall in the passband of the "E Filter". The passbands should not overlap nor should there be gaps between the passbands. The stop bands should provide 60dB of attenuation and the passbands should have 1 dB of ripple. Each filter should have no more than 1000 FIR coefficients. Stopband edges should be adjusted accordingly.

Write a MATLAB function called Tuner_FIRs.m that takes no input but provides the three sets of filter coefficients as outputs, called b_E, b_A, and b_D. This function will use the Parks-McClellan procedure to design the filters and compute the FIR filter coefficient vectors that are then provided as outputs of the function.

This function should not be part of the DFT_Tuner.m function; it will be run once and then its results will be passed into DFT_Tuner.m as input variables.

V-B: Detection Filter Processing

This section describes the first part of the processing to be done by the DFT_Tuner.m function.

Once the filters are designed by the Tuner_FIRs.m file they can be used as part of your DFT_Tuner.m function as follows.

Use the MATLAB "filter" command to apply each of your filters to the 1-second guitar snippet in the input vector x. Call the respective outputs y_E, y_A, and y_D. Because of the design of the filters, you should get a large output from the filter whose passband matches the fundamental frequency of the input signal x. So, for example, if the input has a fundamental frequency near 82.41 Hz we should see a large signal out of the b_E filter and should see smaller signals out of the other two filters. In your report explain this idea using concepts from class and results for your filters.

As a simple means to detect which note the signal's fundamental frequency is closest to we will compute the total power out of each filter by summing the squares of the output and then dividing by the total number of elements summed. (Comment in your report as to why this can be viewed as a power).

Now we can use the largest of these three powers as an indicator of which of the three "bands" the signal's fundamental frequency lies in: the "E-Band", the "A-Band", or the "D-Band".

So the result of this stage of processing is: you should have an "indicator" variable that shows in which of the three "bands" the fundamental frequency lies.

V-C: DFT Processing

This section describes the second part of the processing to be done by the DFT_Tuner.m function.

Compute the DFT of the input signal using the Hamming window and an appropriate amount of zero-padding so that you can achieve frequency accuracy of "5 cents".

Use the results from the detection stage processing to extract the portion of the DFT result that "covers"

the detected band. There are quotes around "covers" because you might want to extend this portion to be a little bit beyond the edges of the detected band – just in case the fundamental frequency is close to the edge of the detected band.

To find your estimated value for the fundamental frequency, find the frequency that corresponds to the largest peak in the magnitude of the extracted DFT result. This is your fo_est that you will provide as an output variable for your DFT_Tuner.m function.

V-D: Detecting Out-of-Range Notes

In the above description we assumed that the signal will have a fundamental no lower than about 68 Hz and no higher than 165 Hz. If for some reason a string was tuned way too low or way too high then this situation would be violated. Now devise a way to detect when this occurs and provide an additional output that indicates if the fundamental is below or above the stated range – there is no need to estimate the frequency in this case… just give an indication of which side of the range it lies on. Since you have no estimated frequency in this case, you should assign the NaN value to the fo_est variable.

At this point you should have a working DFT_Tuner.m function that will take in the specified input variables and will output an estimate of the signal's fundamental frequency in Hz. In addition, your function should create a single figure that has two subplots: the top one should show the full DFT (in dB) over all the positive frequencies and the bottom one should show the DFT (in dB) only over the range of frequencies of the extracted section used while searching for the peak.

V-E: Testing

You should perform a variety of tests that explore the proper operation of the tuner over a range of fundamentals. Your goal is to demonstrate the overall proper operation as well as the proper operation of the various parts of the processing. Use concepts from the course to explain why your results support a claim of proper operation. Any unusual results should be discussed and explained.

Is the use of the Hamming window crucial for this application?

Think carefully about this section… crafting good test cases and procedures is VERY important!