6. Audio File Matching.

(a). → When $\vec{X_1} = [1\ 1\ \cdots\ 1]^T$, $\vec{X_2} = [1\ 1\ \cdots\ 1]^T$, with length n.

So $\vec{X_1}^T \vec{X_2} = [1\ 1\ \cdots\ 1] \cdot [1\ 1\ \cdots\ 1]^T = n \cdot 1^2 = n$.

→ When $\vec{X_1} = [1\ 1\ \cdots\ 1]^T$, $\vec{X_2} = [1\ -1\ 1\ -1\ \cdots\ 1\ -1]^T$ with an even length,

So $\vec{X_1}^T \vec{X_2} = [1\ 1\ \cdots\ 1] \cdot [1\ -1\ \cdots\ 1\ -1]^T = 1 + (-1) + \cdots + 1 + (-1) = 0$.

→ A larger dot product implies that the vectors are more similar.

This is because if the vectors are less similar, the dot product would cancel out more, which would lead to a smaller dot product, and vice versa.

(b) My approach is to pull out 3 consecutive digits from $\vec{x}$ each time, and then dot multiply each 3-bit string/vector with $\vec{Y}$.

Yes, I can write this as a matrix vector multiplication:

$$A = \begin{bmatrix} 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix}, \vec{X} = \vec{X} = \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$$

Finishing my approach above, the dot product of A and $\vec{x}$, $A\vec{x}$, would be a 6×1 vector, and the row with largest dot product is the index.
In other words, if row i has the largest dot product, then i is the answer, representing $[x_i \; x_{i+1} \; x_{i+2}]^T$ as closest to $\vec{Y}$.
Now, to figure out what's closest to $\vec{Y} = [1\,1\,1]^T$,

$$\Rightarrow A\vec{x} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 3 \\ 1 \\ 1 \end{bmatrix}$$

Thus, row 4 has the largest dot product, 3, so $i = 4$ is what we're looking for. (which is indeed $[1\,1\,1]^T$.)

(d)

- Run the following cell. Do your results here make sense given your answers to previous parts of the problem? What is the function vector_compare doing?

The results here make sense since the closer the vectors are, the larger the result (a normalized version of dot products), which is coherent with my previous answers.

In essence, the function vector_compare first takes the dot product of two vectors we wish to compare, and then divides the results by the magnitude of the two vectors, allowing us to negate the effect of their length. Thus, the result would give a “fairer” comparison.

- Run the following code that runs vector_compare on every subsequence in the song- it will probably take at least 5 minutes. How do you interpret this plot to find where the clip is in the song?

We would look at the magnitude (absolute value) of the resulting plot. The larger the absolute value on a given time, the closer the target signal is to the given signal beginning at that moment. (I got some help from the GSIs and their answers on Piazza, and I was convinced that the positive/negative signs of the value doesn’t matter, and only the absolute values matter.)

(e)

- The code below uses song_compare to print the index of given_signal where target_signal begins. Can you interpret how the code finds index? Verify that the code is correct by playing the song at that index using the play_clip function.

Yes, I can. The code functions by enumerating all absolute values of the variable song_compare, which has given us the plot in part (d). The code then prints out the index where the largest absolute value occurs. As I said above, we could find the clip in the given signal by by finding out the largest absolute value when we run the function vector_compare and run_comparison on each slice of the given signal with the target signal.

I can verify that the code is correct as I compared the target signal and the final answer given by play_clip; they are exactly the same.