Project 1: LPC Lossy Compression of Speech Signals

Thomas An (A11257870)

Prefessor: Ken Kreutz-Delgado
ECE174
11/01/2016

Objective/Purpose/Introduction

The purpose of quantization of audio signals and compressing them is to be able to transfer data at faster rates. Should everything be kept at original size, data transfer would be extremely inefficient. Thus, in this project we were to compress a .wav file that we obtained on the internet and reconstruct it with similar audio quality to the original. We compress using quantization; however, in most cases direct quantization would render an audio file unusable because of the severe loss in quality. In this assignment, we are to read in an audio file and calculate the least-squares-estimate of the coefficients of the model and residuals, as well as the difference in mean squared-error between the quantized speech signal and the reconstructed speech signal from residuals. We compare the two because we should be getting a smaller mean squared-error from the quantized residuals versus the directly quantized signal.

Procedure

Step 1: I went online and found a website with a numerous amount of .wav files, and I listened to each one and picked one that wasn't too long nor had any gaps in sound in the middle of the file. I stored the file into the MATLAB directly, and proceeded to load it into the MATLAB code using 'y = wavread('infinity.wav');'. I played the .wav file using the command 'sound(y)' to make sure it was loaded correctly.

Step 2: In step two is where a lot of the work starts to come in. Because we have to find ways to separate the data into blocks of 160 sample points, I decided to use MATLAB cell arrays. In my code, I ended up putting in the loop iterating from Y and storing into C. This way, we have all the sample points in Y stored into cell array C. In order to start the quantization of y, we first needed to calculate the mean and standard deviation of every single block in the cell array, as seen. We start with truncating all the outliers out to get a 'ythresh' vector, and the importance of this is to get rid of points which may interfere with data for better consistency. Only after that, can we start calculating the 'q' using the equation "q = (max(ythresh)-min(ythresh))/(L-1)". After we have the quantized Y value, we then calculate the mean squared-error by using the equation "MSE = (y-yq)'*(y-yq)/N" where y is the original y and yq is the quantized y vector. We then can later test for different MSE values of varying quantization bit rates as well as varying alphas, until we obtain a signal that sounds the best to us as well as with possibly the lowest MSE.

Step 3: In this step, we are measuring the filter coefficients (a(k)) for each separate block of 160 sample points. We do this by setting up a 'Aa = b' inverse linear problem. Luckily, the MATLAB function toeplitz can help us make the inverse matrix. I had to look up the API for toeplitz in order to figure out how the function actually worked. First, I initialized toepstore and fillstore in order to store the fillcoefficients as well as the toeplitz matrices. We toeplitz commands takes in a column and a row, but each column must have the last index of the previous block, as well as the next 159 sample data from the next block. In the row input for toeplitz, each row must have the last 10 elements of the block, but in reverse order. After storing these in the column and row we then call toeplitz upon these and then divide it by the current block in C (fillstore{itr,1} = toepmatrix1\current;) to then get the filter coefficients stored in a matrix. Because we must use the previous blocks, we can initialize the first block with all zeros and then start our block iterations from there. Then we move on to the next block, iterating row, column, and the current marker block.

Step 4: In step 4, we calculate the residual errors in each block with the equation

$$e(n) = y - \hat{y}(n) = y(n) - \sum_{k=1}^{\ell} a(k)y(n-k)$$

We must calculate the residuals block by block though. So, I created a for loop to iterate through every single block of b (or in this case, in my code my C cell array) as well as the toepstore and fillstore arrays. Step 4 in itself is pretty short and simple.

Step 5: In step 5, we were to create a FIR digital filter program to compute the residuals directly with the equation 'e(n) = y − y^(n) = y(n) − Σa(k)y(n − k)' where a(k) is our filter coefficients. In order to get the summation part of the equation, I had to iterate through every value in toepstore as well as multiply it through with the filter coefficient matrix, summing onto each result along the way. After going through the length of each one, I store that into a y_hat cell array, which will later just represent the whole summation. This y_hat can then be used to solve for the residuals. We then compare the MSE values calculated from step 4 and 5 and it should be close enough to zero.

Step 6: In step 6, we are now to write a AR digital filter program that reconstructs the audio directly from the equation y(n) = Σa(k)y(n − k) +e(n) where Σa(k)y(n − k) is y^(n). I have two constructed sets of residuals, so I decide to reconstruct y based off of each steps residuals. I initialize a y_reconstructed4 and y_reconstructed5 cell array to store the additions of each block. I then compared the MSE comparing the original y with the reconstructed y, which should be identically zero, which is it.

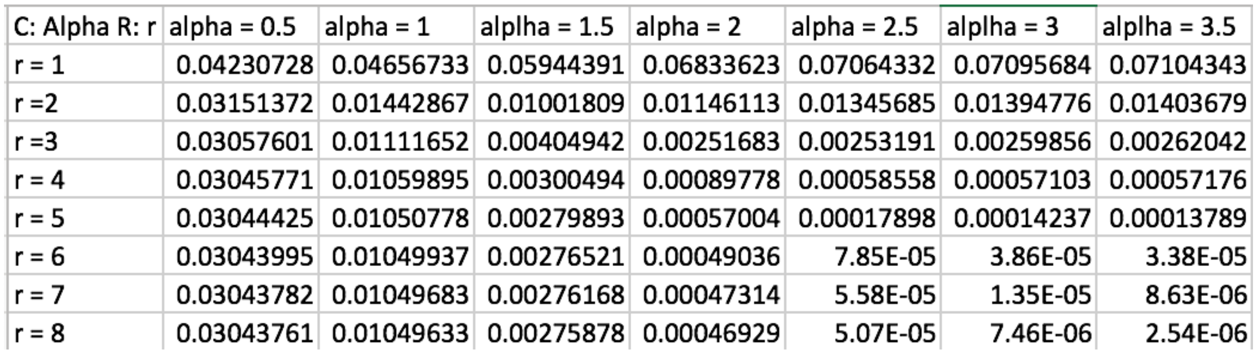
Step 7: In step7, I am now to write code to quantize the residuals using similar code as I did in step 2. However, because the choice of outliers here is very important, it is crucial to play around with the alpha values. In this step, I created a cell array to store all of my residuals, as well as two copies that I would later use to manipulate. I copy over the residuals that I calculated from step 4 over into the cell arrays. In order to see how alpha affects the outliers as well as the MSE, I put everything in a for loop, iterating through alphas, to see which alpha resulted in the best sounding sound file as well as the best MSE. Much of step 7 looks similar to step 2, and that's because the procedure is almost the same. After calculating the mean and standard deviations, I truncate the values and then quantize the residuals After that, I calculate the MSE values for varying alphas and store them into a MSE matrix. In this step, I decided that the sample with an alpha value of 3 was the best. After looking at the matrix, the MSE was identically zero and the sound of the file sounded close to the original file.

Step 8: In step 8, I calculated the MSEs similarly to how I calculated in step 7, however this time with varying quantization bit rates. I had to go back to steps 2 and 7 and implement a nested for loop, iterating through r as well as alpha. I decided to iterate r through the values of 1 and 8 and then stored each corresponding MSE value into a multidimensional matrix, which is later used to plot the values.

Step 9: In this step, I had to retrieve the filter coefficients that were obtained in step 3 and then quantize them just like I had done to the residuals as well as y. Again, I had to play around with the alpha values so I created a for loop to iterate through the alpha values and stored them into a MSE matrix. After quantizing the alpha values, I then quantized the residual values, very similarly to how they were quantized in step 7 except this time only with r values of 4, 8, and 9. We then repeat steps 7 and 8, reconstructing the y values and then finding the MSE values for every different alpha value as well quantization bit rate and then storing that into a matrix.

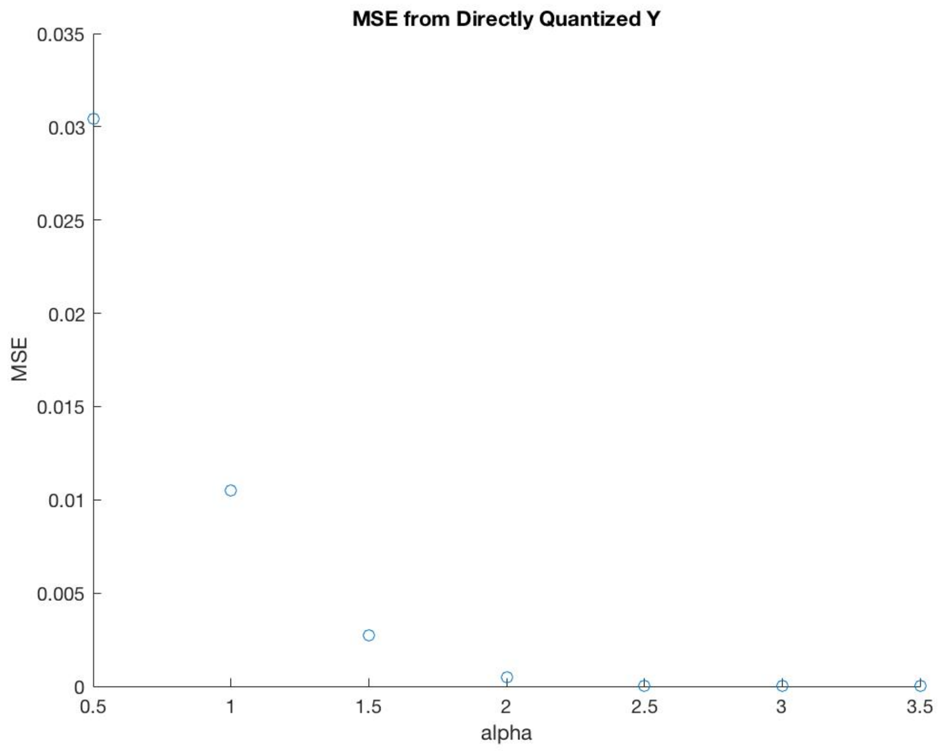
Results

Step 2: As discussed in step 2, I decided to go with a quantization bit rate of 8 as well as an alpha value of 3.5. After observing the graph as well as the data of varying alphas and r's, I decided that it yielded a low enough MSE value for there to be little different in interpretation in sound. In this report, alpha is the thresholding constant that I use to determine how many standard deviations away I want to truncate values. The higher the alpha, the wider the range of values I keep in. I proceeded to then test the sound comparing it to the original and it sounded close to the original.

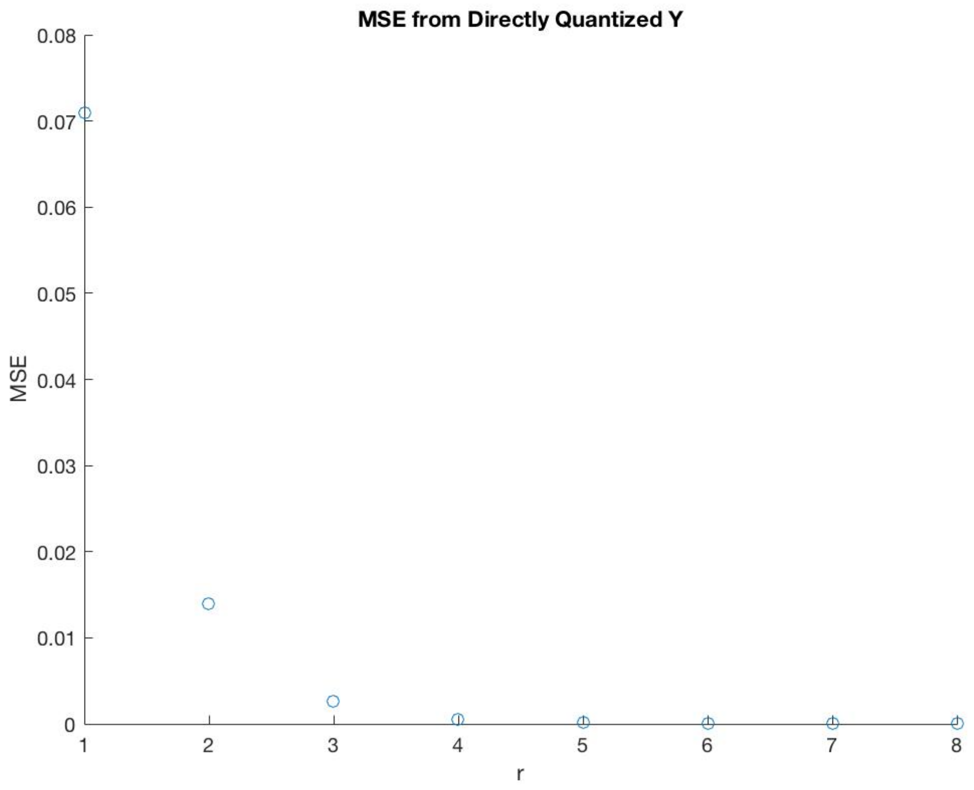


C: Alpha R: r	alpha = 0.5	alpha = 1	alplha = 1.5	alpha = 2	alpha = 2.5	alplha = 3	alplha = 3.5
r = 1	0.04230728	0.04656733	0.05944391	0.06833623	0.07064332	0.07095684	0.07104343
r =2	0.03151372	0.01442867	0.01001809	0.01146113	0.01345685	0.01394776	0.01403679
r =3	0.03057601	0.01111652	0.00404942	0.00251683	0.00253191	0.00259856	0.00262042
r = 4	0.03045771	0.01059895	0.00300494	0.00089778	0.00058558	0.00057103	0.00057176
r = 5	0.03044425	0.01050778	0.00279893	0.00057004	0.00017898	0.00014237	0.00013789
r = 6	0.03043995	0.01049937	0.00276521	0.00049036	7.85E-05	3.86E-05	3.38E-05
r = 7	0.03043782	0.01049683	0.00276168	0.00047314	5.58E-05	1.35E-05	8.63E-06
r = 8	0.03043761	0.01049633	0.00275878	0.00046929	5.07E-05	7.46E-06	2.54E-06

Chart of MSE values of directly quantized Y

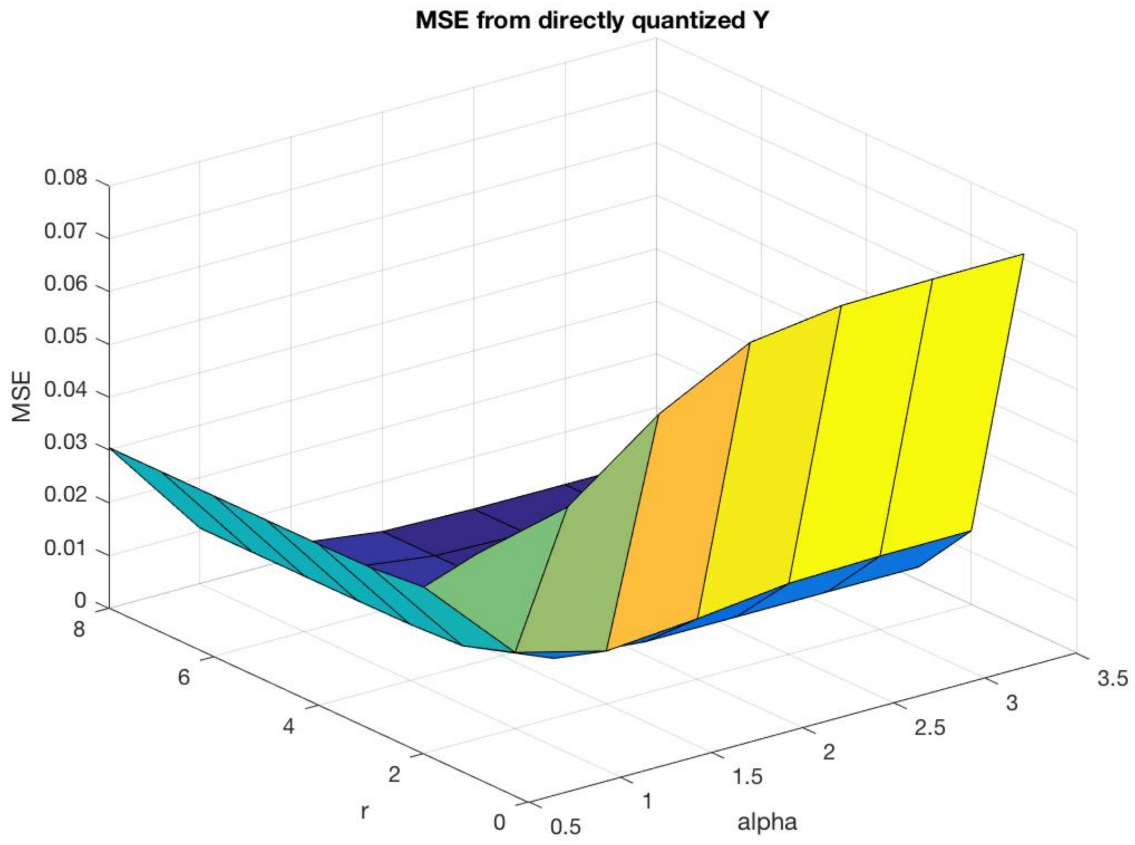


Graph of MSE values from Directly Quantized Y with R held constant



MSE from Directly Quantized Y with Constant Alpha

As we can see from the scatter chart of the directly quantized y, an alpha value of 3.5 yields a similar MSE value as when alpha is 3, but when listening to the sounds, the higher alpha value was still just a tad bit better. I chose the quantization bit rate of 8 because for this value, typically all of the MSE's were at their lowest value. Typically, as r increases as well as alpha, the MSE for the directly quantized values decreased as well. When listening to the sounds, generally, the lower the MSE, which essentially means the error between the files were less, the sounds were more close to the original.
For a more detailed graph, modeling every change in a 3d model, attached here is the graph of MSE with varying alpha and r. We can see the general trend of increasing r or alpha leads to a decrease in MSE.



Step 3: My filter coefficients were stored into a 185x1 cell array where each cell has 10 elements stored inside. The values of my filter coefficients vary greatly in range, where they can range anywhere from around 2 to around -2. Most of my filter coefficients stayed roughly between -1 and 1 though. The filter coefficients were obtained by using the toepliz(col,row) command, where column contains the last element of the previous block as well as all the elements of the current block save for the last one and row contains the last ten elements of the previous block in reverse order. In order for the toeplitz command to word, I initialized the previous block marker to all zeroes for the first iteration. After getting the toeplitz matrix, to get the filter coefficients I matrix divided it by the current block.

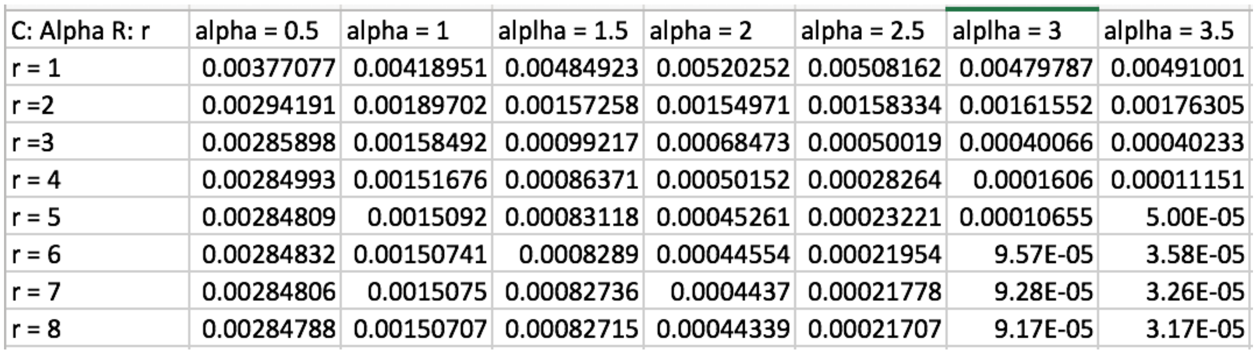
Step 4 and 5: In my results, I obtained that the dynamic range for the residuals was roughly 1.1888 whereas my y dynamic range is 1.9922. There is quite a bit of significance to dynamic range, especially when it has to do with quantizing the signal. The lower the dynamic range, the better. It means that the values aren't so spread apart. This is why we quantize the residuals, or would prefer to quantize the residuals, over the original signal. With a significantly lower dynamic range, there is less room for error during quantization. In step 5, when we calculate the residuals directly from the equation and compare the MSE with that of step 4, the result should be identically zero. This is because the method of calculating it is pretty much the same, just that the equation that we use:

$$e(n) = y - \hat{y}(n) = y(n) - \sum_{k=1}^{\ell} a(k)y(n-k)$$

just essentially compresses the toepmatrix part down to the summation. The summation part of the equation (a(k)y(n-k)) is equation to when I multiply my toeplitz matrix with my filter coefficient matrix. This, when we calculate the error between the two it should be zero: they are essentially the same function just in different forms.

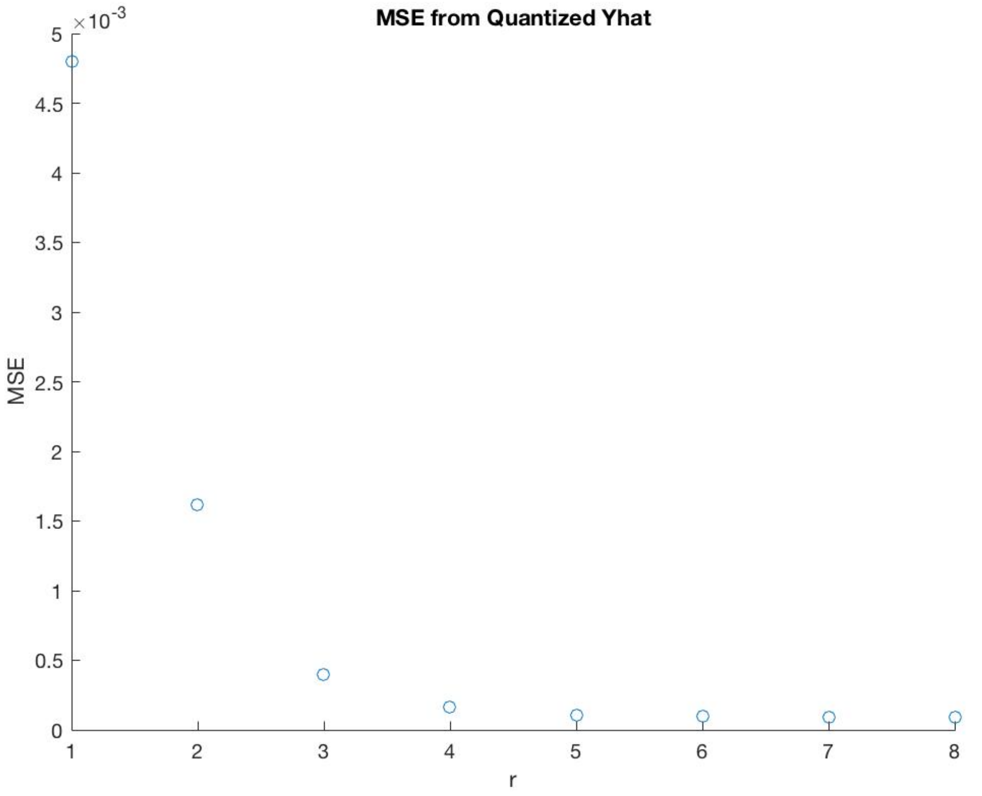
Step 6: In this step, I reconstructed two different audio signals, one with the residuals calculated in step 4 and the other using the residuals calculated in step 5. However, it shouldn't matter which one I used since the two values are virtually identical. From the step, I get a value of 3.0791x10^-33 for step 4 residuals and 9.7565x10^-37 for step5, which both are virtually zero. These samples should be identically zero because we are essentially remaking the original audio signal and then seeing how different it is from the original signal. There would be fault with our reconstruction should the MSE be large between the reconstructed signal and original signal.

Step 7: In this step, I also chose a value of r = 8, and an alpha value of 3. When going through the sample of the reconstructed sounds, I found that this signal produced a sound very similar to the original signal, as well as other high alpha/high r value signals.

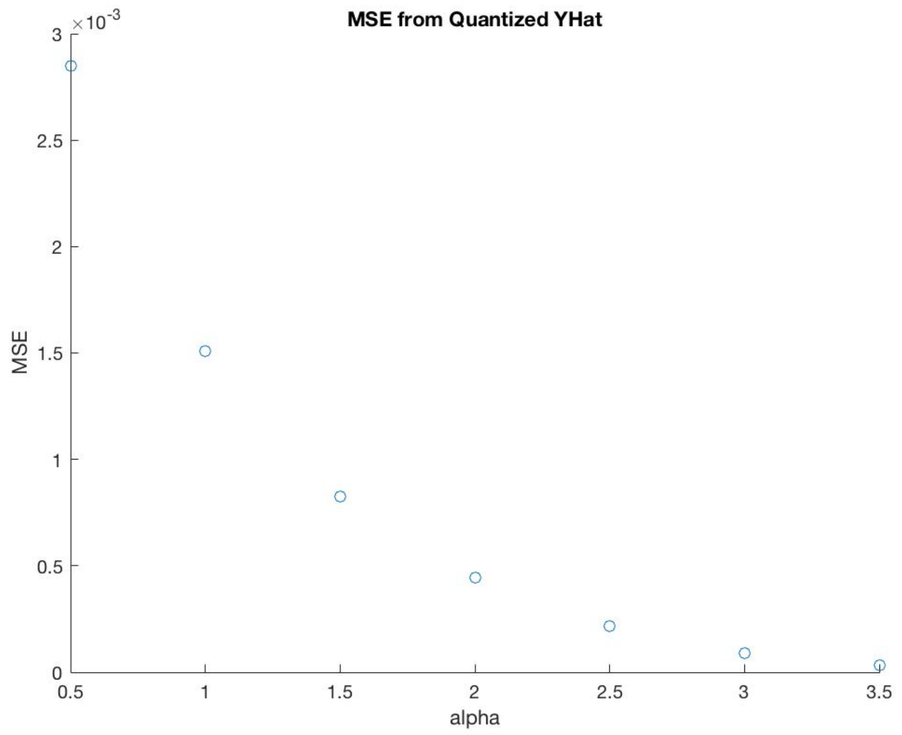


C: Alpha R: r	alpha = 0.5	alpha = 1	alplha = 1.5	alpha = 2	alpha = 2.5	alplha = 3	alplha = 3.5
r = 1	0.00377077	0.00418951	0.00484923	0.00520252	0.00508162	0.00479787	0.00491001
r =2	0.00294191	0.00189702	0.00157258	0.00154971	0.00158334	0.00161552	0.00176305
r =3	0.00285898	0.00158492	0.00099217	0.00068473	0.00050019	0.00040066	0.00040233
r = 4	0.00284993	0.00151676	0.00086371	0.00050152	0.00028264	0.0001606	0.00011151
r = 5	0.00284809	0.0015092	0.00083118	0.00045261	0.00023221	0.00010655	5.00E-05
r = 6	0.00284832	0.00150741	0.0008289	0.00044554	0.00021954	9.57E-05	3.58E-05
r = 7	0.00284806	0.0015075	0.00082736	0.0004437	0.00021778	9.28E-05	3.26E-05
r = 8	0.00284788	0.00150707	0.00082715	0.00044339	0.00021707	9.17E-05	3.17E-05

We can see in this chart MSE values of varying r and alpha values for the quantized residuals, the overall MSE values are lower as we deduced previously. Also, when we reach an alpha value of 3.5 and a r value of 8, the MSE is virtually zero with how low it is. When listening to the sound samples I found this r and alpha value led to the best sounding file. This makes sense however, because the MSE value for this signal is the lowest.

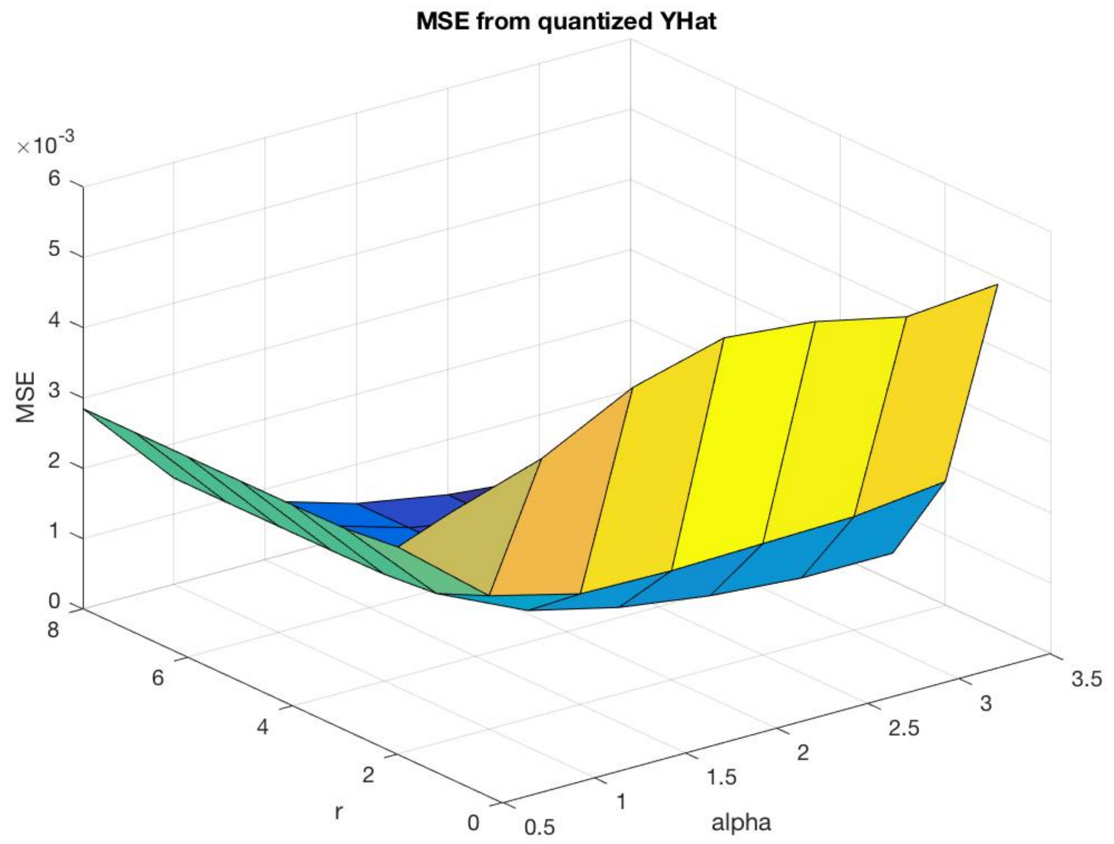


MSE from Quantized residuals with constant Alpha



MSE from Quantized residuals with constant R

We can see from the two graphs above, and also reinforced by the 3D model below, once the R and Alpha value reaches those low values the MSE is virtually zero and thus the reconstructed signal is relatively close to the original. As also observed from a previous step, and increasing R as well as increasing Alpha leads to a lower MSE.

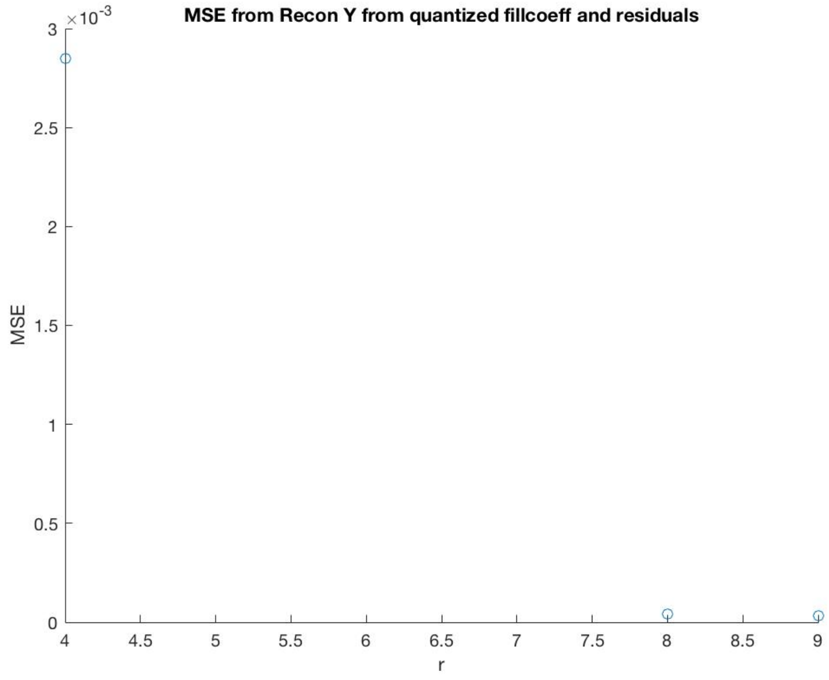


Step 8: When comparing the MSE values of the quantized residual versus when the signal was quantized directly, the MSE values for the quantized residuals were typically lower when R and Alpha were lower. Because there is less room for error with quantization with the residuals due to a lower dynamic range, we would typically want to quantize the residuals over directly quantizing the original signal. When listening to the sound of the reconstructed signals, the sound of reconstructed signal from the quantized residuals sounded better and that made sense since the MSE for that signal was lower, thus meaning it is closer to the original. I thought as the quantization rates and the alpha values got higher and higher for the quantized residuals, the sound signal sounded roughly the same. However, when compared to the quantized original signal, it definitely sounded less choppy.

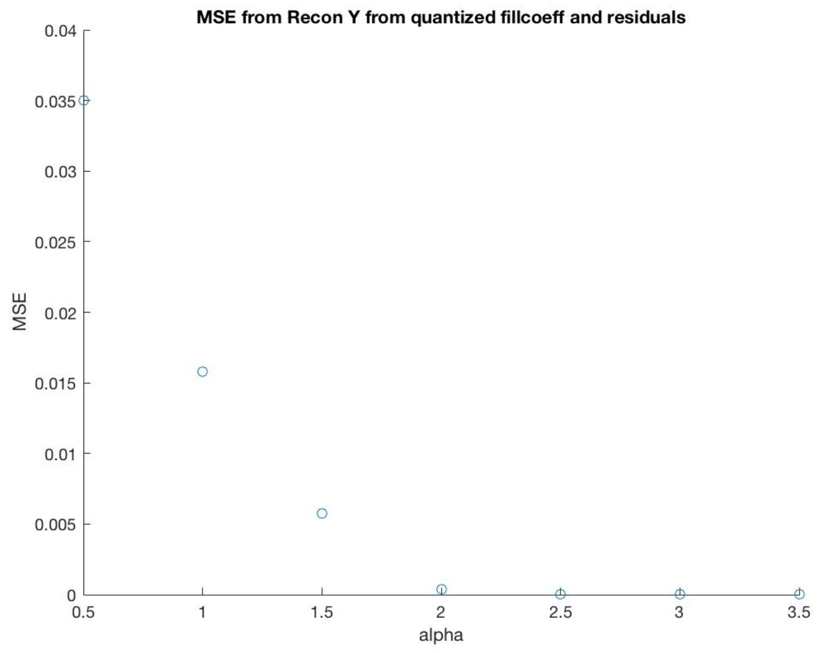
Step 9: In step 9, I quantized the filter coefficients and again got a 185x1 cell array with each cell storing 10 elements inside. I obtained the quantized filter coefficients very similarly to how I obtained quantized y and residuals. I went through every single element in the filter coefficient matrix and truncated each value accordingly with y, then I also quantized the residuals like I did in Step 7. From these two, I used the same equation as I did in step 5 to reconstruct the signal with both these quantized values then obtained the MSE for varying R and Alpha values.

C: Alpha R: r	alpha = 0.5	alpha = 1	alplha = 1.5	alpha = 2	alpha = 2.5	alplha = 3	alplha = 3.5
r = 4	0.03800416	0.02027382	0.01022381	0.00359383	0.00311128	0.00298632	0.00294579
r = 8	0.03819046	0.01757441	0.00664754	0.00077593	0.00022893	0.00010321	4.30E-05
r = 9	0.03820275	0.01750337	0.00660255	0.00076622	0.00021952	9.39E-05	3.39E-05

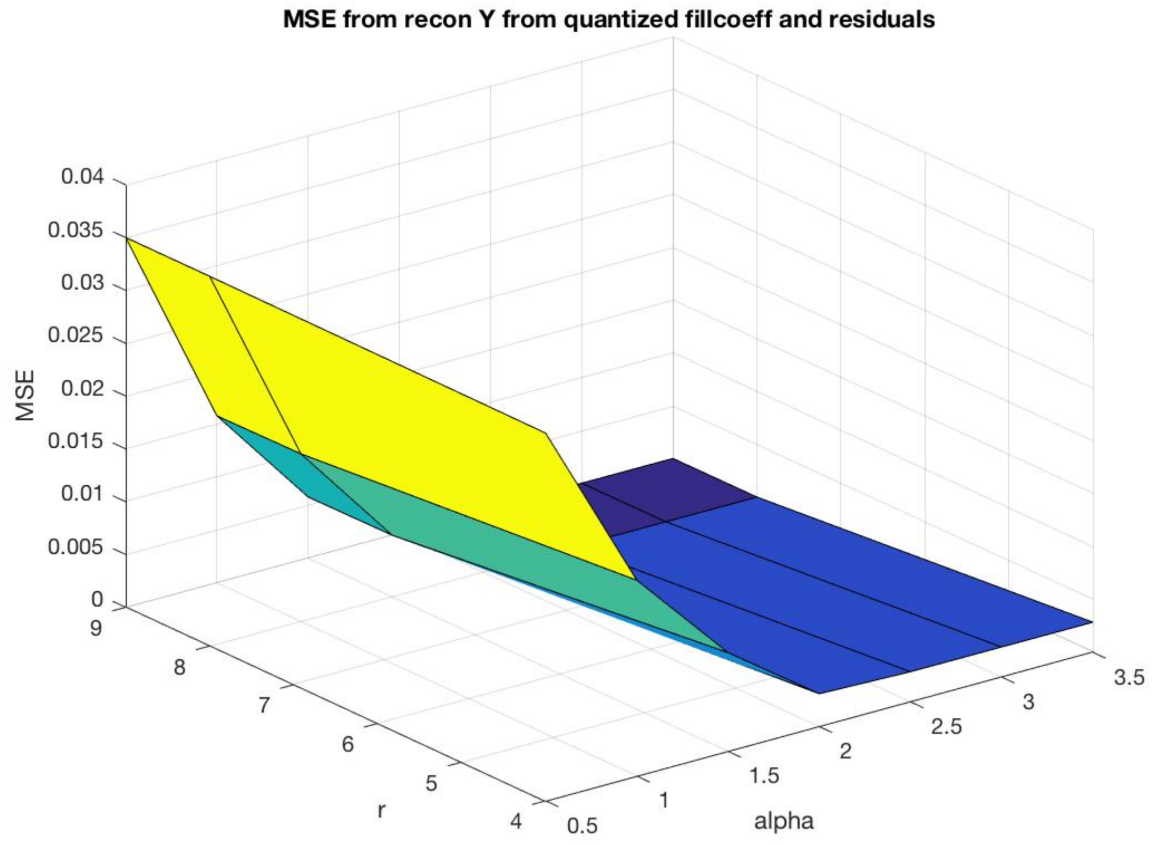
As we can see from the chart above, the alpha value of 3.5 and a R value of 9 led to the lowest MSE, and also just so happens to be the best sounding signal.
Both of the scatter plots below show how the MSE changes with varying R or Alpha values when either alpha or r is held constant. The final graph shows all of them tied together and we can still see a similar trend as we've seen in the previous graphs, but with increasing r and alpha, the MSE value is lower.



MSE from reconstructed Y with quantized FillCoeff and Residuals with Constant Alpha



MSE from reconstructed Y with quantized FillCoeff and Residuals with Constant R



Conclusion:

Overall in this coding assignment, we were to find a way to compress the audio signal without a severe loss in quality. We found that through quantizing the residuals, we could achieve this. The residuals have a lower dynamic range so quantizing those would lead to less error during the reconstruction of the signal, and thus giving a signal that is more similar to the original one in sound. Because of this, we are able to compress the signals, achieve faster transfer rates, without the severe drop in quality.