Audio Compression Using Least Squares

Thomas Kok
A07121303

October 28, 2010

Introduction

For organizations supporting the transaction of many thousands of digital audio signals at a time, the compression of those audio signals is crucial to reducing the cost of the network infrastructure. This paper will discuss the use of least squares optimization in the compression of audio data. The objective is to use least squares techniques to reduce the bit rate of an audio signal significantly while maintaining sufficient audio fidelity for the transmission of speech.

Method of Least Squares

The least squares compression technique begins with an equation representing the uncompressed audio signal $y(n)$ as the result of a filter with coefficients $a(k)$ added to the residual, or error, $e(n)$.

$$y(n) = \sum_{k=1}^{10} a(k)y(n-k) + e(n)$$

This equation can be rearranged to give an expression for $e(n)$ in terms of $y(n)$.

$$e(n) = y(n) - \sum_{k=1}^{10} a(k)y(n-k)$$

The compression algorithm is predictive, so it will function best if $y(n)$ is consistent in frequency and amplitude throughout the duration of the signal. This is an unreasonable assumption for speech, but if the speech is separated into small blocks, the model can still work well. These blocks must be small enough to be largely uniform internally (much less than one second) but large enough that a predictive model can form useful filter coefficients and avoid large residuals.

From a high level point of view, the least squares algorithm forms the filter coefficients $a(k)$ for each block by solving the linear inverse problem $y(n) = \sum_{k=1}^{10} a(k)y(n-k)$ for $a(k)$. There is no exact solution to this equation: the best possible result is the least squares solution, which minimizes $e(n)$. The residuals $e(n)$ are then calculated with the expression above. These residuals have a much smaller range of magnitudes than the original audio signal, which means they can be quantized at a lower bit rate while avoiding excessive distortion. The residuals and filter coefficients are then transmitted to the receiver, which implements the above expression for $y(n)$ to reconstruct the audio signal $\hat{y}(n)$.

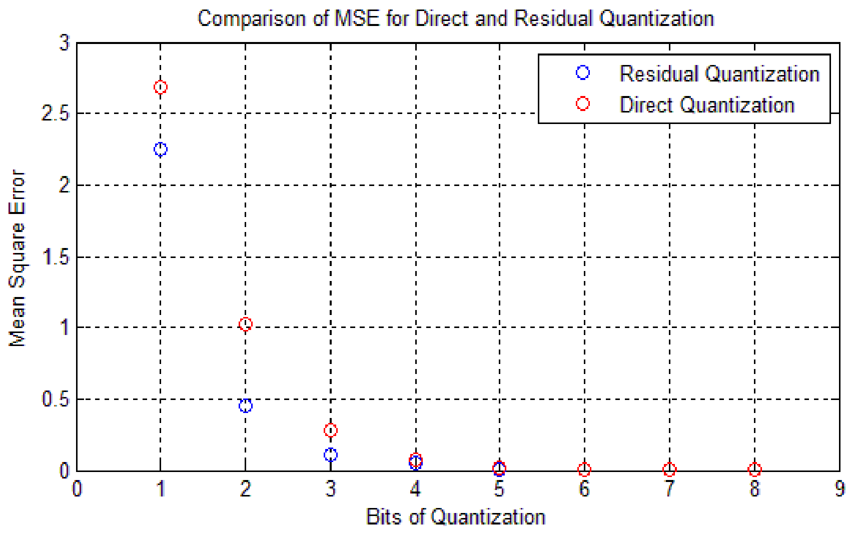
Implementation and Results

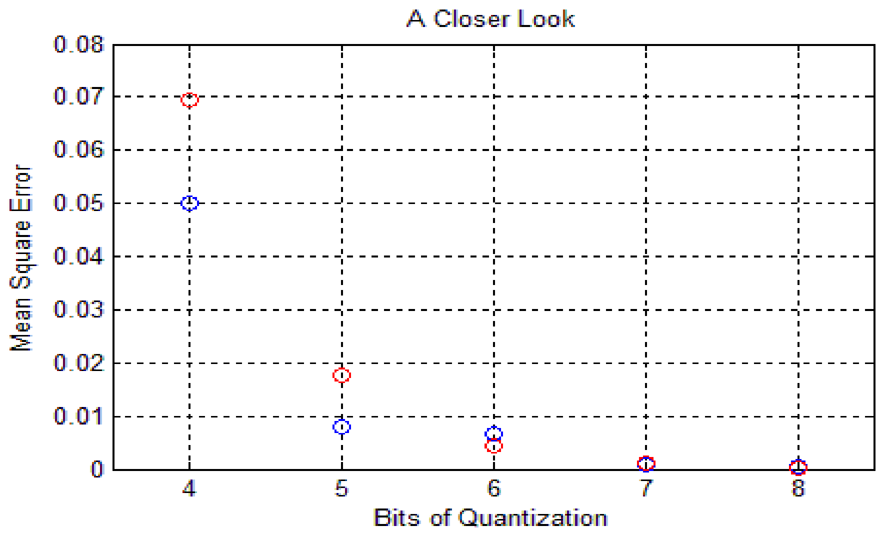
To prototype this algorithm and gauge its effectiveness, Matlab will be used to process a pre-existing uncompressed audio file. For convenience, the file is normalized so that its magnitude is at most one and any data points that do not fit into a block will be truncated from the signal. The sampling frequency for the file, F_s, is a typical 44.1kHz. A block size of 160 data points will be used (less than one tenth of a second).

In order to make a meaningful comparison of the least squares compression to other compression techniques, we will also compress a signal via direct quantization (this is computationally and conceptually simpler). For each block, the mean and standard deviation of the data is found. Data points exceeding α standard deviations from the mean are truncated to the threshold value, where α is a real scalar chosen by testing various values and minimizing the error of quantization. The blocks are then quantized into 2^r levels where r is the number of bits per data point. The mean square error between this quantized signal, $y_q(n)$, and $y(n)$ is easily calculated.

α plays an important role in the quantization of signals. The program tests a range of values for α, from one to five in steps of .25, for each quantization rate, and uses the value that gives the lowest error. Low quantization rates ($1 \leq r \leq 2$) had minimal error with lower values of α, but higher quantization rates work best with higher values of α, from three to five. The mean square error proves to be a fairly good indicator of the distortion heard in the quantized signal. Once the error is below .01 there is little to no audible difference in the sound as quantization rate is increased, though the error does still decrease. A quantization rate of $r = 6$ is sufficient to achieve full quality with direct quantization.

Proceeding to the least squares technique, MATLAB's matrix division (mldivide) operator is used to produce the least squares, minimal error values of $a(k)$ for each block. The calculation of the residuals follows trivially. After testing that the function for constructing $y(n)$ given $a(k)$ and $e(n)$ works correctly, the same quantization technique used for computation of $y_q(n)$ is applied to $e(n)$, producing $e_q(n)$. Again, a range of values for α were tested to find the error-minimizing values, with very similar results. To compare the fidelity of the signals produced by the least squares method to those produced by direct quantization, the errors produced by each method are plotted.





It is apparent that the least squares method produces substantially smaller errors at lower bit rates, while the two are effectively equal in error for higher bit rates. Since higher compression is desired, these results are favorable to the least squares method. However, it was previously noted that the error must be fairly small before the signal's quality is acceptable for transmission of speech. An error of .01 was taken to be, in effect, perfect transmission. The least squares method achieves this with only one bit less than direct quantization.

Listening to $\hat{y}(n)$ gives a different perspective. At low bit rates, $\hat{y}(n)$ sounds much less distorted than $y_q(n)$, but it has a loud chirp or click inserted into the speech. Even with $r = 3$ the quality is acceptable for speech if the click is excluded, half the bits required for direct quantization. To choose between the two techniques, it is also crucial to note that, in addition to $e_q(n), \hat{y}(n)$ requires $a(k)$. Transmitting $a(k)$ will add to the overall bit rate required for the least squares method. Unfortunately, attempts to quantize $a(k)$ have demonstrated that the standard quantization technique used for $y_q(n)$ and $e_q(n)$ is not suitable for producing $a_q(k)$. Doing so results in an unstable filter, which causes $\hat{y}(n)$ to grow without bound, rapidly exceeding the limits of double precision floating point numbers.

Using the data collected so far, we see that $F_s \times r_{direct} = 44100 \times 6 = 2.646 \times 10^5$ bits per second are required for transmission of directly quantized signals. For the least squares method, $F_s \times r_e + (10 \times \frac{F_s}{160})r_a = 44100 \times 3 + 2757 r_a$ bits per second are required. This means r_a can be as high as 48 and the least squares method will still have a lower bit rate requirement than direct quantization. To make the least squares technique viable, it will also be necessary to eliminate the loud click present in $\hat{y}(n)$. It is possible that a better quantization routine would solve both this problem and the issue of filter stability.

Conclusion

Prototyping of the least squares technique for audio compression indicates that it could greatly reduce the bit rate required to transmit acceptable-quality speech data. There is substantial work yet to be done in refining the algorithm, especially in the quantization technique. The potential cost savings to an organization like a telecommunications company are very large, assuming that the costs of computing filter coefficients, errors, and signal reconstruction are not so high that they offset the bandwidth savings. The least squares technique seems very likely to perform favorably against direct quantization.