Low Rank Approximations of Matrices using Rank Revealing QR Factorization

Byron Liu
Department of Computer Science
University of Colorado Boulder
Boulder, Colorado
byli4778@colorado.edu

John Mathews
Department of Applied Mathematics
University of Colorado Boulder
Boulder, Colorado
joma6354@colorado.edu

Will Vasilas
Department of Applied Mathematics
University of Colorado Boulder
Boulder, Colorado
wiva2833@colorado.edu

Abstract

In solving systems of linear equations, it is useful to employ matrices, but the large order of some linear systems makes their computation and storage resource-intensive. To combat this problem, one can compress a matrix so that less space is taken, but little information is lost. This is accomplished by constructing a QR factorization of the matrix A, where the columns of Q are orthonormal and R is upper-triangular. Under a permutation P of A, the diagonal entries of R are in decreasing order, and R can be truncated after these entries dip below a certain tolerance.

This compression can be applied to audio. If one can construct a matrix representation of an audio file, then it is possible to compress that audio file using the rank revealing factorization technique. It can also be possible to take a matrix, find its rank revealing QR factorization, and expand the R matrix to generate a matrix that compresses into the original matrix. In this way, one can decompress audio.

1. Introduction

In many instances, it is useful to be able to easily determine the rank of a matrix, especially if the matrix is rank-deficient. This is a difficult to impossible task by pure observation, but there are a number of factorizations that concentrate the ill-conditioning of such a matrix into a single diagonal matrix D, revealing the rank easily through observation of the D matrix. These rank-revealing factorizations (RRFs) all follow the same general form [Chandrasekaran and Ipsen, 1994].

In general, an RRF is defined as the following:

$$A = XDY^T$$

where A is a real matrix $(m \times n)$, and D is a diagonal, nonsingular matrix. The QR factorization is a rank-revealing factorization where $X = Q$, $D = diag(R)$, and $Y^T = D^{-1}R$.

In a rank deficient matrix, the Q matrix is $m \times r$ and the R matrix is $r \times n$, where r is the rank of the matrix. When the matrix is numerically rank deficient, the bottom $m - r$ rows of R have small entries along the diagonal. This definition of the factorization is flawed. Take the following example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Under the standard definition of the QR factorization, A breaks down into the following:

$$A = QR = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The rank-revealing definition of the factorization of this matrix is not defined for A — since the diagonal of R is zero, the inverse of D does not exist. This problem requires us to use pivoting to permute the matrix into a form that avoids singularity of the D matrix. The matrix is pivoted so that the norms of the columns are ordered from greatest to least. This results in the diagonal of the R matrix having entries ordered greatest to least, which avoids the singularity issue. The first r columns of R have the greatest norms (Gu & Eisenstat, 1996). For the factorization in this paper, A will be permuted so that the diagonal entries of R will be in decreasing order. This ordering produces a helpful property.

In R, there is an element r_{ii} such that r_{ii} is less than a given value ε. Because the diagonal entries of R are ordered, this inequality holds for every subsequent entry. The factorization can be written in the form below, where P is a permutation matrix and the Q and R matrices are written in block form. Note that R_{11} contains all of the diagonal elements greater than ε.

$$AP = [Q_1\, Q_2] \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

$$AP = Q_1[R_{11}\, R_{12}] + Q_2[0\, R_{22}]$$

We claim that the Q_2 term is negligible. This is because the 2-norm of R_{22} is bounded above by its Frobenius norm. The norm is bounded above by the expression $\frac{1}{\sqrt{2}}(n-r+1)\varepsilon$, where n is the number of rows of the R matrix, r is the number of rows in the R_{11} matrix, and ε is as defined above. This expression is $O(\varepsilon)$. If we take the difference between AP and $Q_1[R_{11}R_{12}]$, we get an expression for the error.

$$AP - Q_1[R_{11}\, R_{12}] = Q_2[0\, R_{22}]$$

$$\|AP - Q_1[R_{11}\, R_{12}]\|_f = \|Q_2[0\, R_{22}]\|_f$$

The Frobenius norm of the two matrices multiplied together on the right-hand side can be expressed as follows, with q_i denoting the columns of Q_2 and r_i^T denoting the rows of R_{22}.

$$\|Q_2[0\, R_{22}]\|_f = \sum_i \|q_i\|_2^2 \|r_i\|_2^2 + \sum_{i \neq j} < q_i, q_j >< r_i, r_j >$$

Since Q_2 has orthogonal columns, the term $< q_i, q_j >= 0$ for $i \neq j$. This means that the second sum is zero. The columns of Q_2 are also unitary, so $\|q_i\|^2$ is equal to 1. With these simplifications, the bound for the error becomes more clear.

$$\|Q_2[0\, R_{22}]\|_f = \sum_i \|r_i\|^2 = \|[0\, R_{22}]\|_f^2 \leq \frac{1}{2}(n-r+1)^2\varepsilon^2$$

This means that the norm of the error term is bounded by $O(\varepsilon^2)$, so the Q_2 term can be neglected.

The size r of the matrix block R_{11} is the number of rows in the new truncated R matrix. It is also the number of columns in the truncated Q matrix, which means that there are r numerically independent columns in A, i.e. the order r is equal to the numerical rank of A. The r columns of Q_1 nearly span the image of A. In other words, they approximate an orthonormal basis of the range of A. The first r columns of AP are the r most linearly independent columns of A that are spanned by Q_1.

2. Numerical Experiments

Method

To begin to test our methodology of RRQR factorization, we first have to develop a procedure for generating random rank-deficient matrices where we can control the rank. The process of randomly constructing a rank-deficient matrix A involves utilizing our definition of an RRF, $A = XDY^T$. Because the ill-conditioning of the matrix A is condensed into D, if we generate a random matrix by manually creating a diagonal matrix D and randomly generating the X and Y matrices, the result is a pseudo-random matrix A where the rank of the matrix is known based on D.

To evaluate the accuracy of the compression, as well as the effects of a chosen ε on the accuracy, it is necessary to test a wide range of matrices over different values of ε. We take ε on different orders of magnitude (0.01, 0.1, 1, 10, 100, etc) and test fifty randomly generated matrices for each value. Then, the resulting approximated matrix is subtracted from the original to generate the error matrix. The average 2-norm of these error matrices is plotted against the value of ε on a log-log plot (Figure 1).

Results

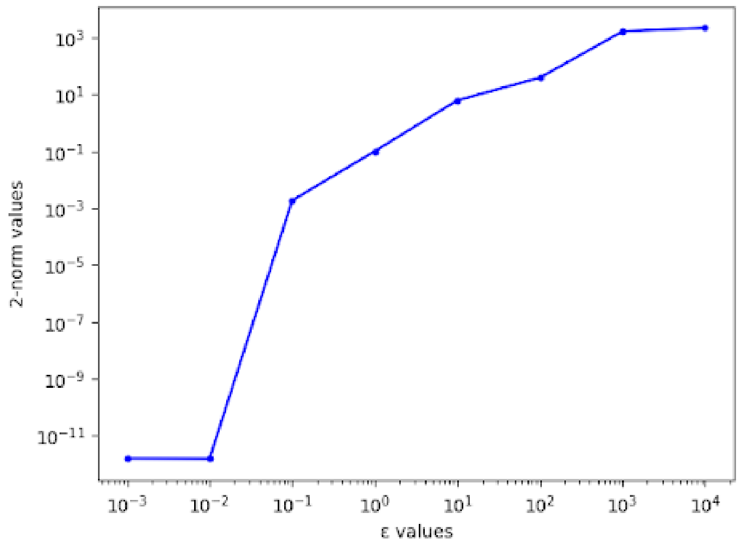


Figure 1: Log-log plot of ε vs average 2-norm

As is evident from Figure 1, smaller values of ε create more accurate approximations. This result is intuitive. Smaller values for ε will truncate less of the R matrix, which means that less data is discarded. In the plot, the relationship between ε and the norm appears roughly linear. For a log-log plot, this means that the correlation is a power relationship, the order of which is determined by the slope of the line. We estimate the visual slope to be approximately 1.5, which suggests superlinear convergence. This calculation neglects the extreme values of ε because of the behavior at the ends of the graph.

It is noteworthy that the slope of the plot drops off in the last interval. This is because an ε value of 104 is so high that the error cannot get any worse. As such, the norm of the error matrix stagnates at absurdly high ε values. The behavior at the left of the graph is also interesting. The error drops to machine precision very quickly. This suggests that very little of the data is being thrown out for these values, so no compression is occurring. For the randomly generated matrices we tested, setting ε to 0.1 both compresses the matrix and retains relatively high accuracy.

3. Extension

One interesting application of the RRQR decomposition is the compression of audio. Like a picture, an interval of sound can be expressed as a matrix. Unlike analog implementations, digital audio storage has the advantage of being more-or-less immune to fluctuations in electric circuits, which allows for storage without much loss in data, so long as the waveform can be sampled enough to capture its nuances.

One way to represent an audio signal is through a spectrogram matrix, the rows of which represent frequency ranges and the columns of which represent overlapping time intervals. The value of entry (i, j) is the amplitude of the Short-Time Fourier Transform (STFT) at frequency i and time interval j [Smith, 2010]. (While a spectrogram usually refers to a heat-map-like visualization of the amplitudes, the idea can be applied to a matrix instead.) A useful advantage of the spectrogram is the fact that the matrix is relatively square, as opposed to a time-domain signal, which has a number of rows equal to the number of samples used—usually

44,100 times the length of the audio signal in seconds—and one or two columns depending on whether the track is in mono or stereo. This dimensionality problem means that the representation is not well-suited for QR decomposition.

To create the spectrogram, we first read a `.wav` file with Python's scipy.io.wavfile to produce the time-domain signal and apply the Short-Time Fourier Transform `stft` function from `scipy.signal`. This function takes a time-domain signal, sample rate, and other parameters (such as overlap between time frames) and returns the complex-valued STFT matrix. We then take the magnitudes of the complex entries for the spectrogram and the angles for the phase matrix, at which point we apply the RRQR method to the spectrogram. (While it is theoretically possible to implement RRQR on the phase matrix, this could affect the data to the degree that it is uninterpretable when reconverting it into a waveform.)

After compression via RRQR, we reassemble the spectrogram by multiplying Q and R and pivoting the columns back into their original positions. We then combine the newly-compressed spectrogram with the phase matrix to produce a new Short-Time Fourier Transform and apply the Inverse STFT to create a new time-domain signal, which we normalize and write onto a new `.wav` file.

To apply this method, we use the `test_file.wav` recording in the Github repository linked on the first page. This 11-second track has minimal distortion in its uncompressed form, containing only kick drum, snare drum, shaker, bass, electric keyboard, and clean electric guitar, which makes it easy to aurally determine where compression occurs.

After generating the spectrogram of the recording and its pivoted QR factorization, we experiment with different levels of tolerance in the compression of R. Notably, as the tolerance parameter ε increases exponentially, the rank of the matrix decreases linearly, and with it, the level of audio compression also increases (Figure 2). For example, for $\varepsilon = 10$, Q was reduced to 269 columns from its original 513. For $\varepsilon = 100$, this number drops to 77.

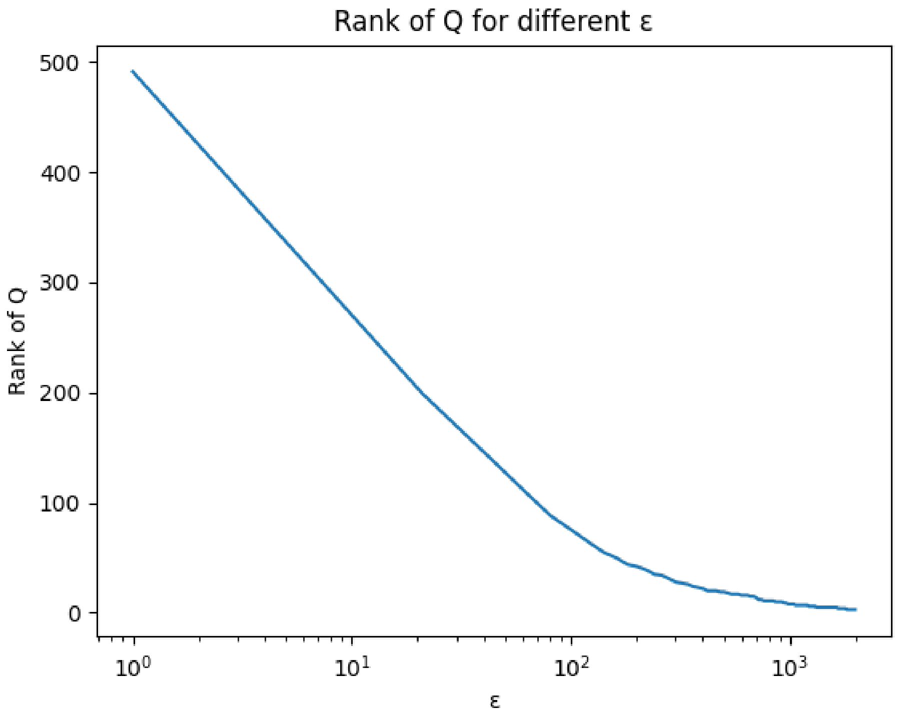


Figure 2: Semilogx plot of the rank of Q for different ε

Measuring the compression of the waveform is more difficult. For $\varepsilon = 10$, the compressed waveform `reconstruction_ε=10.wav` is hardly different to the original—we could say the ear-norm of the difference is negligible. However, when $\varepsilon = 100$, we notice some high-frequency fuzz in the background of the track `reconstruction_ε=100.wav`. Further increases in ε produce even worse sounds, such as in `reconstruction_ε=1000.wav`, where the actual frequencies are drowned out by the noise.

While the compression of a multi-intrumental track like `test_file.wav` added generally unpleasant noises to the recording, we hoped that compressing a recording of a single instrument might create interesting timbres that could be used creatively in a musical context. For example, in the 1950s, when musicians were experimenting with the electric guitar, some guitarists would poke holes in their amplifiers to create natural fuzz tones [Scott, 2018].

To test this idea, we recorded the intro to *Purple Rain* by Prince and applied RRQR with varying thresholds, ε, for data preservation. The original recording (Figure 3) contains only electric guitar with reverb, pitch modulation (chorus), and minimal distortion, but as ε increased, the data loss produced artificial distortion (Figure 4).

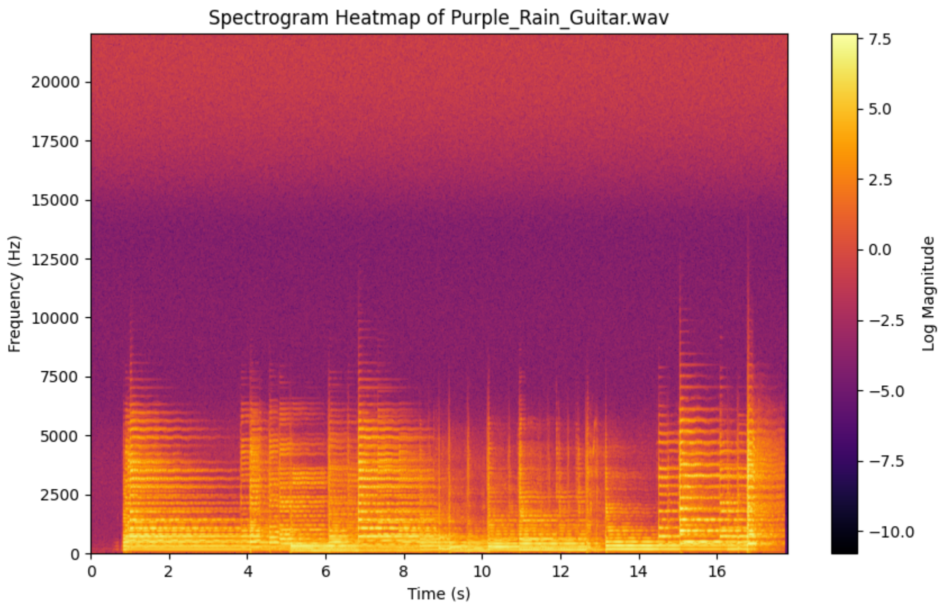
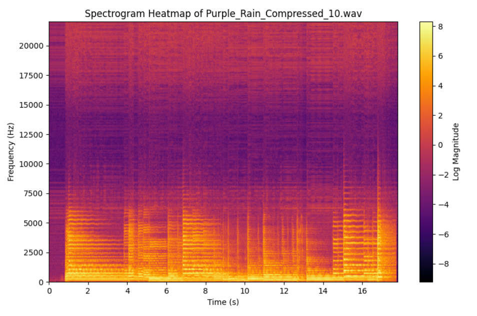


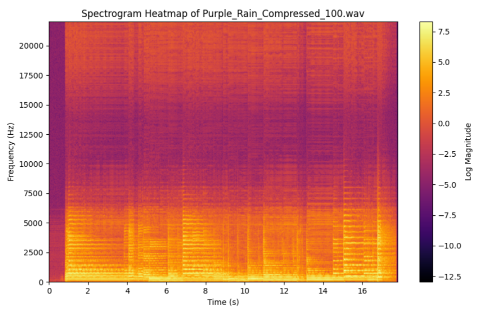
Figure 3: Spectrogram of *Purple Rain* with no data loss

Since the uncompressed recording rarely reaches frequencies above 12000 Hz, the higher frequency space appears empty with some low-volume background noise. However, this space is affected even in the least compressed recording (Figure 4 (a)) leading to light fuzz in the audio and horizontal lines in the visualization. The spectrograms also do an excellent job in representing the rhythm breakdown caused by RRQR compression, as represented by the smoothness of the plot in Figure 4 (d).

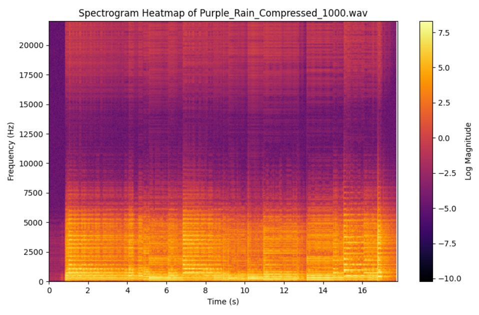
This experiment was inherently subjective, as different people have different musical preferences, but there were some interesting conclusions. The audio quality was noticeably worse for all levels of compression. In the $\varepsilon = 10$ case, the tone was relatively unchanged, with only some fuzz breaking out on louder notes. Similarly, for $\varepsilon = 100$, the actual guitar did not appear affected —- the compression manifested itself in a high-pitched hum in the background. However, the most compressed recording (Figure 4 (d)) was almost unrecognizable. The interesting case occurred with $\varepsilon = 1000$, where fuzz appeared not only as an additional hum, but also in the guitar tone itself, creating a compelling retro or lo-fi sound. Although our method was not designed to be used as a creative mechanism, this case shows that there may exist levels of compression that could produce viable musical timbres.



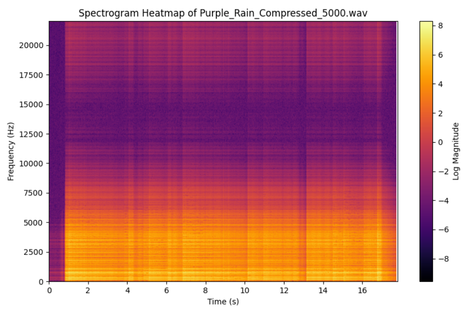
(a) $\varepsilon = 10$



(b) $\varepsilon = 100$



(c) $\varepsilon = 1000$



(d) $\varepsilon = 5000$

Figure 4: Spectrograms of *Purple Rain* with varying distortion levels

4. QR Expansion Methods

The goal of the expansion algorithm is to produce a matrix A^* that compresses to the same matrix that A does. As shown in the Introduction section, we can neglect the $Q_2[0\, R_{22}]$ term in the block form of the pivoted QR factorization if the matrices satisfy two properties. First, the columns of Q_2 must be orthonormal. This property allows for the elimination of the cross terms:

$$\sum_{i \neq j} \langle q_i \ , \ q_j \rangle \langle r_i \ , \ r_j \rangle$$

Second, the diagonal entries of R_{22} must have an upper bound of ϵ. Take $AP = QR$. In order to generate a matrix A_1 that compresses to the same matrix as A, we can take the compression of A and add a valid $Q_2[0\, R_{22}]$ term.

To create the R_{22} matrix, it is necessary to enforce the restriction on the diagonal. This is accomplished by setting the top left entry to a random value between $-\epsilon$ and ϵ (the tolerance used to compress the A matrix). Every subsequent entry will follow this pattern.

$$r_{n\,n} \in (-r_{n-1\,n-1}, r_{n-1\,n-1})$$

The lower triangular portion of the R_{22} matrix will be filled by zeroes. The proof for the neglect of the $Q_2[0\, R_{22}]$ places no restrictions on the upper triangular entries of the R_{22} matrix. They can be filled completely arbitrarily, but for the simplicity of the algorithm, they will conform to the same condition as the diagonal entries. Practically, this is achieved by setting the top row to be bounded by the corresponding entries in the bottom row of the $[R_{11}\, R_{22}]$ matrix (that being the entries directly above and to the left) and repeating this pattern until the upper triangle is filled. If the matrix A is $m \times n$, and the $[R_{11}\, R_{22}]$ matrix is $k \times n$, then the $[0\, R_{22}]$ matrix must be $(m-k) \times n$. This is because the uncompressed Q matrix is square with dimensions $m \times m$, which requires the R matrix to be $m \times n$.

The Q_2 matrix only has the requirement that its columns must be orthonormal. This requirement could be fulfilled by simply appending columns of the identity matrix onto the Q_1 matrix until it is square. However, this would not produce a valid Q matrix, since the columns of Q_1 and Q_2 would not form an orthonormal basis. Instead, we will create a valid Q matrix using a least squares approximation.

$$Q_1^T Q_1 x = Q_1^T b$$

Setting the vector b equal to a column of the identity matrix, the vector $Q_1 x$ becomes the closest vector in the column space of Q_1 to that column of the identity matrix. Since Q_1 has fewer columns than it has rows, the span of its columns is a strict subset of the space of all $m \times 1$ matrices. The columns of the identity matrix, on the other hand, do span the space of all $m \times 1$ matrices, which means that for some columns of I, the least squares approximation given by $Q_1^T Q_1 x = Q_1^T b$ is not equal to the vector b. Moreover, this inequality implies that $b - x$ is orthogonal to all columns of Q_1. This column can be normalized by dividing it by its 2-norm.

The evaluation of the least squares problem is trivial. First, we examine the term $Q_1^T Q_1$. Each entry of the resulting matrix is a dot product of the rows of Q_1^T and the columns of Q_1. The rows of Q_1^T are identical to the columns of Q_1. Since the columns of Q_1 are orthonormal, all entries off the diagonal evaluate to zero and all entries on the diagonal evaluate to 1. Thus, we derive the identity matrix. This means that $x = Q_1^T b$. This expression simplifies further still. Since we take b to be a column of the identity matrix, it selects one column of Q_1^T, or one row of Q_1. More specifically, the expression $Q_1^T e_j$ selects the j^{th} row of Q_1, which will be referred to as q_j^*. The expression for the orthogonal component of e_j becomes the following:

$$\hat{q} = e_j - Q_1 q_j^*$$

Normalizing this matrix and appending it to the Q_1 matrix completes the first step of an iteration that fills the missing columns. We apply this algorithm repeatedly until the matrix is square. The new columns define a Q_{22} matrix not necessarily identical to the original, but valid nonetheless.

The algorithm has issues. First, it is resource intensive. Each step of the iteration requires multiple manipulations of large matrices. Second, the iteration becomes less accurate the closer the algorithm comes to completion. When the matrix Q_1 is almost square, its columns nearly span the space of $m \times 1$ matrices, so the error in the least squares approximation is very small. The subtraction inherent in finding the orthogonal component then produces floating point errors. To combat this effect, one could run the least squares approximation once more on the orthogonal component $\hat{q}$ until the dot product of $\hat{q}$ with each of the columns of Q_1 is bounded by a certain tolerance. This solves the accuracy problem at the cost of more computer resources.

5. Conclusion

In the block form of the rank revealing QR factorization, the $Q_2[0\, R_{22}]$ term can be neglected due to the orthogonality of the Q matrix and the diagonal of R descending in absolute value. It is necessary to pivot the matrix A in order to fulfill this second condition. Numerical experiments revealed that the data loss in compression scaled with the tolerance parameter, ε.

The rank revealing QR factorization produces an audio compression algorithm that preserves the broad structure of the sound. From auditory observation, the compression sounds similar to audio compressed by other means. Certain levels of compression on certain structures of sound produce interesting effects that musicians could use artistically.

With further time, the expansion algorithm could be optimized and implemented on audio to produce novel sounds. The expansion for the R matrix in particular has few limitations, only requiring the diagonal to be bounded above. The upper triangular values of R_{22} can be filled completely arbitrarily, which allows for many different algorithms. Each algorithm would produce a different kind of sound. In the future, these different algorithms could be developed and tested to create a variety of auditory effects.

References

S. Chandrasekaran and I. C. F. Ipsen. On rank-revealing factorisations. *SIAM Journal on Matrix Analysis and Applications*, 15(2):592–622, 1994. doi: 10.1137/s0895479891223781.

C. Daniel, A. Kaufman, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the gram-schmidt qr factorization. *Mathematics of Computation*, 30(136):772–795, 1976. doi: 10.2307/2005398.

M. Gu and S. C. Eisenstat. Efficient algorithms for computing a strong Rank-Revealing QR Factorization. *SIAM Journal on Scientific Computing*, 17(4):848–869, 1996. doi: 10.1137/0917055.

N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Review*, 53(2):217–288, 2011. doi: 10.1137/090771806.

J. Scott. History of distortion part 1. *The JHS Show*, 2018.

J. O. Smith. *Mathematics of the Discrete Fourier Transform (DFT): With Audio Applications (2nd Ed.)*. BookSurge, 2010.