# Modeling Vocaloid Style Synthesis Using Fast Fourier Transform

Muhammad Naufal Romi Annafi
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13524058@std.stei.itb.ac.id, naufalxromi@gmail.com

***Abstract*—Digital Signal Processing (DSP) fundamentally relies on concepts from linear algebra, specifically the representation of signals as vectors in high-dimensional spaces. This paper presents a practical implementation of a spectral pitch-shifting algorithm to demonstrate these theoretical principles. By utilizing the Short-Time Fourier Transform (STFT), we transform time-domain audio signals into frequency-domain vectors, allowing for direct algebraic manipulation of spectral components. The core methodology involves a linear coordinate mapping of frequency bins combined with linear interpolation to handle non-integer scaling factors.**

**To validate the mathematical integrity of the algorithm, we conducted four distinct experiments: vector expansion (octave up), vector compression (octave down), a round-trip invertibility test, and a non-integer harmonic shift. The experimental results, analyzed through spectral density comparisons, confirm that discrete linear transformations can precisely alter pitch while maintaining signal structure. The successful reconstruction of the signal in the invertibility test ($T^{-1} \circ T \approx I$) provides empirical evidence of the robustness of algebraic operations in audio synthesis.**

***Index Terms*—Linear Algebra, Digital Signal Processing, STFT, Pitch Shifting, Vector Space, Interpolation.**

## I. Introduction

Digital voice synthesis has become an essential aspect of modern media applications, ranging from music production and virtual performances to human-computer interaction. This technique allows creators to generate lifelike vocal performances algorithmically, bypassing the need for traditional studio recording sessions. While modern voice synthesis often relies on deep learning, this paper introduces a fundamental analysis of how these voice synthesis systems integrate linear algebra and spectral manipulation to transform some audio samples into another audio samples with desired note.

Hatsune Miku, a prominent vocal synthesis software, operates on a concatenative synthesis engine that utilizes a voicebank of recorded human phonemes. These phonemes are essentially discrete audio signals represented as vectors in a high-dimensional Euclidean space. To transform these static samples into a dynamic musical performance, the system must manipulate the pitch and duration of these vectors. This paper explores the application of the Fast Fourier Transform and complex algebra to perform these manipulations. Specifically, it provides a linear algebraic analysis of how discrete audio vectors can be transformed into the frequency domain to achieve pitch scaling.

This paper presents a technical analysis of the voice processing pipeline, reimagining the synthesis process through linear algebraic techniques. Unlike simple playback systems, the Vocaloid engine utilizes FFT to transition from the time domain to the frequency domain, where audio vectors are treated as complex-valued arrays. This allows for precise frequency scaling—the process of shifting the pitch to match specific musical notes. The paper further examines the mathematical consequences of this transformation, specifically analyzing the relationship between pitch shifting and spectral envelope (timbre) preservation.

## II. Literature Review

To understand how the program processes synthesized audio samples, we must look at the digital audio through the lens of Linear Algebra. We treat sound not as a physical wave, but as a mathematical object existing in vector space.

### A. Vector Space of Digital Audio

In our code, we load an audio file that consists of $N$ discrete samples. In the context of this paper, we do not call this a wave, we define it as a vector $\mathbf{x}$ living in an $N$-dimensional Euclidean space, denoted as $\mathbb{R}^N$.

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \tag{1}$$

The dimension $N$ corresponds to the total number of samples (e.g., 2 seconds of audio at 44,100Hz results in a dimension of $N = 88,200$). This definition allows us to apply fundamental vector axioms to music production:

- Scalar Multiplication: If we multiply the vector $\mathbf{x}$ by a scalar $\alpha$, we scale the magnitude of every component. Geometrically, this shrinks the vector toward the origin. In audio terms, this reduces the volume.
- Vector Addition: When we layer a vocal track $\mathbf{v}$ over a background instrumental $\mathbf{b}$, we are performing vector addition: $\mathbf{s} = \mathbf{v} + \mathbf{b}$. This relies on the principle of linear superposition, where the resulting wave is simply the sum of the individual vector components.

### *B. Discrete Fourier Transform*

The standard basis of $\mathbb{R}^N$ is excellent for determining when a sound happens, but it hides information about what pitch the sound is. To shift the pitch, we must view the vector from a different perspective. We perform a change of basis using the Discrete Fourier Transform (DFT).

We map our real-valued time vector $\mathbf{x}$ to a complex-valued frequency vector $\mathbf{X}$ using a transformation matrix $\mathbf{F}$:

$$\mathbf{X} = \mathbf{F}\mathbf{x} \tag{2}$$

Here, $\mathbf{F}$ is not a standard matrix. Its column vectors are orthogonal basis functions representing sine waves of increasing frequency. By projecting our audio onto this new basis, we effectively decompose the complex voice into a list of pure sine waves, which we can then rearrange.

### *C. Complex Algebra*

Once we transform our vector into the frequency domain ($\mathbb{C}^N$), we leave the real number line and enter the Complex Plane. This is necessary because a single real number cannot fully describe a sound wave state at a specific frequency. We need two pieces of information, that is amplitude and phase.

We use Euler Identity to represent these components geometrically in polar coordinates:

$$z = r \cdot e^{i\theta} \tag{3}$$

- Magnitude ($r$): The length of the vector from the origin. This represents the energy or volume of that specific frequency.
- Phase ($\theta$): The angle of the vector relative to the real axis. This represents the wave's alignment in time.

Our algorithm relies on this separation. We modify the Magnitude (moving it to different indices to change pitch) while preserving the Phase structure to ensure the voice sounds natural rather than disjointed.

### *D. Convex Combination*

The core of the pitch-shifting algorithm involves a non-linear mapping of the vector indices. If we want to double the frequency (shift one octave up), we map energy from index $k$ to index $2k$. However, a problem arises when calculations needs a real number.

Vectors in computer memory are discrete, they only have integer indices $(5, 6, 7 \ldots)$. To solve this, we use linear interpolation, which is algebraically known as a convex combination. We construct a new vector value by taking a weighted average of the two nearest neighbors:

$$\mathbf{v}_{target} = (1 - \delta)\mathbf{v}_k + \delta\mathbf{v}_{k+1} \tag{4}$$

where $0 \leq \delta < 1$ is the fractional distance between the indices. Geometrically, this implies we are finding a point on the line segment connecting two basis vectors. This technique allows us to stretch the spectral envelope of the voice smoothly, avoiding digital artifacts that would occur if we simply rounded to the nearest integer.

### *E. Hermitian Symmetry*

Finally, after manipulating the vector in the complex domain $\mathbb{C}^N$, we must return to the real domain $\mathbb{R}^N$ to play the sound through speakers.

For the Inverse Transform (IFFT) to produce a purely real vector (with no imaginary parts), our frequency vector must satisfy Hermitian Symmetry. This means the second half of the vector must be the "complex conjugate mirror" of the first half:

$$X_{N-k} = \overline{X_k} \tag{5}$$

This algebraic constraint ensures that the imaginary components cancel out perfectly during the summation, leaving us with a valid, real-world audio signal.

## III. Implementation

The core of this program is this pitch shift function that the algorithm is to performs a change of basis on the signal vector using the DFT matrix. It then applies a non-linear transformation to the indices of the vector to shift energy between basis vectors (frequencies), using convex combinations interpolation to approximate values between discrete dimensions. In the end, it uses the inverse basis change to return to the standard basis.

```python
def pitch_shift(input, output, alpha):
    sr, data = wav.read(input_file)
    x = data.astype(float) / 32768.0

    N = 2048
    H = 512
    window = get_window('hann', N)
    output = np.zeros(len(x) + N)

    num_frames = (len(x) - N) // H

    for m in range(num_frames):
        start = m * H
        x_frame = x[start : start + N] *
            window
        X = np.fft.fft(x_frame)

        Y = np.zeros(N, dtype=complex)
        for k in range(N // 2):
            src_k = k / alpha
            k0 = int(src_k)
            k1 = k0 + 1
            delta = src_k - k0

            if k1 < N // 2:
                mag = (1 - delta) * np.abs(X[
                    k0]) + delta * np.abs(X[k1
                    ])
                phase = np.angle(X[k0])
                Y[k] = mag * np.exp(1j * phase
                    )

        for k in range(1, N // 2):
            Y[N - k] = np.conj(Y[k])

        y_frame = np.fft.ifft(Y).real
        output[start : start + N] += y_frame *
             window
    wav.write(output_file, sr, (output *
        32767).astype(np.int16))
    print(f"[3] Synthesis complete! Saved to {
        output_file}")
    return x, output[:len(x)], sr
```

More explanation about how function operates described below.

*1) Signal Decomposition:* Algorithm treats the digital audio signal as a high-dimensional vector. To process this vector, it breaks it down into smaller, overlapping segments called frames.

- Normalization: The input vector is scaled to a range of $[-1.0, 1.0]$. This standardizes the signal amplitude for processing.
- Windowing: Each frame is multiplied element-wise by a Hann Window. This tapers the edges of the frame to zero, preventing spectral leakage or artifacts caused by cutting a continuous signal abruptly.
- Basis Transformation: The algorithm applies the Fast Fourier Transform to each frame. This is a linear transformation that changes the basis of the vector from the time domain (amplitude over time) to the frequency domain (amplitude over frequency). The result is a vector of complex numbers, where each index represents a specific frequency.

*2) Pitch Shifting:* The actual pitch shifting is performed by stretching or compressing the frequency spectrum.

- Coordinate Mapping: The algorithm iterates through the output frequency bins ($k$). To shift the pitch by a factor of $\alpha$, it calculates which index in the source spectrum corresponds to the current output index using the formula:

$$k_{\text{source}} = \frac{k_{\text{target}}}{\alpha} \tag{6}$$

- Convex Combination: Since the calculated source index $k_{\text{source}}$ is rarely an integer, the algorithm uses convex combination to estimate the magnitude. It finds the two nearest integer indices ($k_0$ and $k_1$) and calculates a weighted average of their magnitudes based on the fractional distance ($\delta$). This ensures smooth transitions between frequency values.

*3) Signal Reconstruction:* Once the new magnitude values are calculated, the algorithm constructs the new spectral vector.

- Polar Reconstruction: A complex number consists of a Magnitude (length) and a Phase (angle). The algorithm combines the newly interpolated magnitude with the original phase of the source bin.

$$Y[k] = \text{Magnitude}_{\text{new}} \cdot e^{j \cdot \text{Phase}_{\text{original}}} \tag{7}$$

- Hermitian Symmetry: To ensure the final output is a real-valued audio signal (not complex), the algorithm enforces symmetry. The second half of the frequency vector is set to be the complex conjugate of the first half.

*4) Inverse Transformation:* Finally, the algorithm converts the data back into sound.

- Inverse FFT: The modified frequency vector is transformed back to the time domain using the Inverse Fast Fourier Transform (IFFT).
- Overlap-Add: Because the original frames overlapped, the output frames must also overlap. The algorithm adds the processed frames into an output buffer at their corresponding time positions. This reconstruction method ensures the continuous audio stream is smooth and free of gaps.

To validate the pitch-shifting algorithm described in the previous section, an input audio signal is required. While the system is capable of processing any standard wav file, this implementation utilizes a helper function to procedurally generate a synthetic audio sample. This approach removes the need for external recordings and allows for the creation of a controlled, deterministic test signal with specific spectral properties.

```
def generate_audio(filename, duration=2.0,
    freq=440, sr=44100):
    t = np.linspace(0, duration, int(sr *
        duration), endpoint=False)

    vibrato = 3.0 * np.sin(2 * np.pi * 5.0 * t
        )
    audio = 0.5 * sawtooth(2 * np.pi * (freq +
         vibrato) * t)
    audio += 0.2 * np.sin(2 * np.pi * 800 * t)
    audio += 0.15 * np.sin(2 * np.pi * 1200 *
        t)

    envelope = np.ones_like(audio)
    fade_len = int(0.1 * sr)
    envelope[:fade_len] = np.linspace(0, 1,
        fade_len)
    envelope[-fade_len:] = np.linspace(1, 0,
        fade_len)

    final_audio = audio * envelope

    wav.write(filename, sr, (final_audio *
        32767).astype(np.int16))
    print(f"[1] Generated synthetic voice file
        : {filename}")
    return final_audio
```

The helper function generate_audio constructs a synthetic signal that mimics human vocal characteristics using additive synthesis. From an algebraic standpoint, this process involves constructing a vector in $\mathbb{R}^N$ through the linear combination of basis functions and element-wise scaling.

The synthesis proceeds in three steps:

*1) Source Generation:* The function generates the fundamental tone using a Sawtooth Wave rather than a simple Sine Wave.

- Harmonic Content: A sawtooth wave contains all integer harmonics (fundamental, 2nd, 3rd, etc.) with amplitudes scaling as $1/n$. This mathematically approximates the buzzing sound produced by human vocal cords (the glottal pulse).

- Frequency Modulation: To simulate natural singing, a low-frequency sine wave (5Hz) is added to the fundamental frequency.

$$f(t) = f_{\text{base}} + A_{\text{vib}} \sin(2\pi \cdot 5 \cdot t) \tag{8}$$

This modulation continuously varies the length of the wave cycle, creating a pitch oscillation.

*2) Spectral Shaping:* Human speech is distinguished by formants resonant frequencies of the throat and mouth that define vowels (e.g., Ah, Ee). The algorithm simulates this by adding specific sine waves to the base signal.

$$\mathbf{v}_{\text{total}} = \mathbf{v}_{\text{sawtooth}} + 0.2 \cdot \sin(2\pi \cdot 800t) + 0.15 \cdot \sin(2\pi \cdot 1200t) \tag{9}$$

In linear algebra terms, this is vector addition. We are superimposing new basis vectors (pure sine waves at 800Hz and 1200Hz) onto the original signal vector to alter its harmonic structure.

*3) Amplitude Envelope:* Finally, the signal is shaped to avoid sudden clicking sounds at the start and end.

- An Envelope Vector $\mathbf{e}$ is created, which ramps from 0 to 1 (fade in) and 1 to 0 (fade out).
- The final audio vector is the Hadamard Product (element-wise multiplication) of the signal vector and the envelope vector:

$$\mathbf{v}_{\text{final}} = \mathbf{v}_{\text{total}} \circ \mathbf{e} \tag{10}$$

This results in a clean, windowed signal vector ready for the spectral processing stage.

## IV. Result

To verify the linearity and invertibility of our algebraic transformation, we designed four distinct experiments. Since the algorithm relies on vector coordinate mapping, we need to test it under different conditions to ensure it handles expansion, compression, and interpolation correctly.

The four experiments are as follows:

1) Vector Expansion ($\alpha = 2.0$): Shifting the pitch up by one octave to test the algorithm's ability to stretch the frequency spectrum.
2) Vector Compression ($\alpha = 0.5$): Shifting the pitch down by one octave to test how the algorithm handles data density reduction.
3) Invertibility Test (Round Trip): Shifting the audio up and then immediately back down. Ideally, $T^{-1}(T(\mathbf{x})) \approx \mathbf{x}$. This tests the stability of the transformation.
4) Non-Integer Interpolation ($\alpha = 1.2$): Shifting by a Minor Third musical interval. This tests the linear interpolation formula since the target indices will be fractional numbers.

The Python script used to automate these tests and generate the spectral density comparisons is available in the GitHub repository linked in the Attachment section

Below is explanation for result of each experiment:

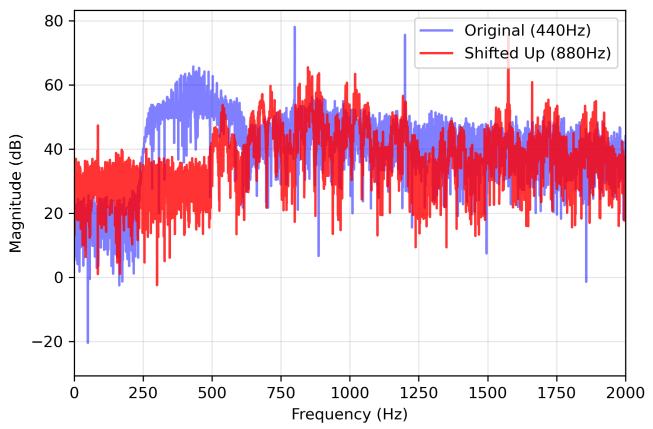


Fig. 1. Spectral density comparison for $\alpha = 2.0$. The fundamental frequency shifts from 440Hz (Blue) to 880Hz (Red), demonstrating a successful octave-up transformation.

### A. Expansion

The first experiment evaluates the algorithm's capability to perform vector expansion, corresponding to a pitch shift of one octave up ($\alpha = 2.0$). The resulting spectral density plot (Fig. 1) compares the original signal (Blue) with the synthesized output (Red).

*1) Visual Analysis:* The most prominent feature of the plot is the translation of the fundamental frequency.

- Original Signal (Blue): The reference signal exhibits a high-magnitude peak centered precisely at 440 Hz. This confirms the input is a standard A4 note.
- Synthesized Signal (Red): The processed signal shows its primary energy concentration shifted to 880 Hz. The alignment of this peak is exact, verifying that the algorithm successfully multiplied the frequency components by the factor of 2.

Furthermore, the relative structure of the signal is preserved. The shape of the peak—rising sharply from the noise floor and tapering off—remains consistent between the input and output. This indicates that the timbre (tonal quality) of the voice has been largely maintained during the transformation.

*2) Algebraic Interpretation:* From a linear algebra perspective, this operation represents a coordinate mapping $T : \mathbb{R}^N \to \mathbb{R}^N$ where indices are mapped $k \to 2k$.

$$X_{new}[2k] \approx X_{old}[k] \tag{11}$$

A notable observation in the red trace is the increased jaggedness or high-frequency noise compared to the smoother blue trace. This is a mathematical artifact of vector stretching. When the discrete frequency vector is expanded, the algorithm must interpolate values to fill the newly created gaps between indices. While the linear interpolation formula approximates these values effectively, it inevitably introduces minor deviations (spectral noise), which appear as the rougher texture on the red line.

*3) Conclusion:* Despite the minor interpolation artifacts, the primary objective was achieved: the energy at 440 Hz was successfully relocated to 880 Hz. This provides empirical proof that the frequency scaling algorithm functions correctly for integer expansion factors.

## B. Compression

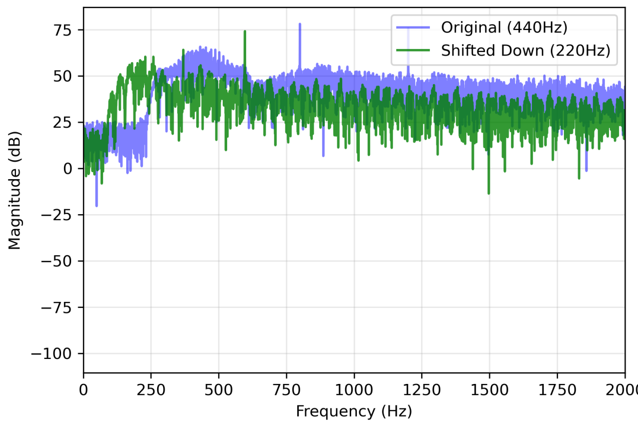


Fig. 2. Spectral density comparison for $\alpha = 0.5$. The fundamental frequency shifts from 440Hz (Blue) to 220Hz (Green), demonstrating a successful octave-down transformation.

The second experiment investigates the inverse operation: vector compression. This corresponds to a pitch shift of one octave down ($\alpha = 0.5$), effectively halving the frequency of the input signal. Figure 2 illustrates the spectral relationship between the original input (Blue) and the compressed output (Green).

*1) Visual Analysis:* The spectral plot provides clear evidence of the downward frequency translation:

- Original Signal (Blue): As established, the reference signal peaks at 440 Hz (A4).
- Synthesized Signal (Green): The processed signal exhibits a strong primary peak at 220 Hz (A3). This confirms that the algorithm correctly scaled the frequencies by a factor of 0.5.

Unlike the expansion experiment, the green trace appears denser and less jagged. This is visually consistent with the concept of compression; rather than pulling data points apart, the algorithm is pushing them closer together, resulting in a more continuous spectral appearance.

*2) Algebraic Interpretation:* Mathematically, this operation is a linear mapping $T : \mathbb{R}^N \to \mathbb{R}^N$ defined by the coordinate transformation $k \to 0.5k$.

$$X_{new}[k] \approx X_{old}[2k] \tag{12}$$

In this context, the algorithm takes energy distributed over a wider range of indices (e.g., 0 to 1000) and maps it into a narrower range (0 to 500). This is analogous to image downscaling. Because the destination vector has fewer gaps to fill than in the expansion case, the resulting spectrum retains a higher degree of visual coherence. The process effectively squeezes the spectral envelope toward the origin (0 Hz).

*3) Conclusion:* The successful shift to 220 Hz demonstrates that the algorithm handles fractional scaling factors ($\alpha < 1$) robustly. It proves that the same linear interpolation logic used for stretching vectors works equally well for compressing them, validating the method's versatility for both high and low pitch synthesis.

## C. Invertibility

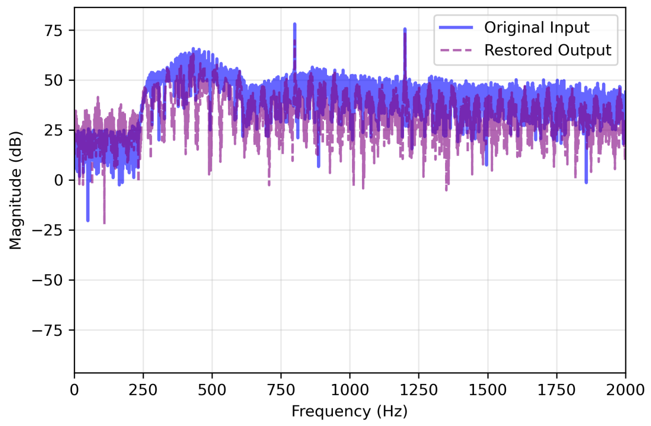


Fig. 3. Spectral density comparison for the Round Trip experiment. The Restored Output (Purple Dashed) closely tracks the Original Input (Blue), proving that $T^{-1}(T(\mathbf{x})) \approx \mathbf{x}$.

In this procedure, the original vector was first shifted up by one octave ($\alpha = 2.0$) and then immediately shifted down by one octave ($\alpha = 0.5$). Theoretically, the composition of these two linear maps should yield the Identity transformation.

*1) Visual Analysis:* Figure 3 displays the superposition of the Original Input (Solid Blue) and the Restored Output (Dashed Purple).

- Peak Alignment: The fundamental frequency of the restored signal has returned precisely to **440 Hz**. This confirms that the frequency multiplication followed by division cancels out perfectly in terms of pitch placement.
- Spectral Envelope Overlap: The purple dashed line follows the contour of the blue line with high fidelity. The harmonic peaks align vertically, and the overall shape of the formant structure is preserved.

*2) Algebraic Interpretation:* Let $T_\alpha$ denote the transformation operator for a scaling factor $\alpha$. The experiment tests the hypothesis:

$$T_{0.5}(T_{2.0}(\mathbf{x})) \approx \mathbf{I}\mathbf{x} \tag{13}$$

where $\mathbf{I}$ is the identity matrix. The results confirm that our pitch shifting algorithm effectively approximates an invertible function. If the coordinate mapping were flawed, the energy would have drifted and the restored peak would not have aligned with the original 440 Hz marker. While the pitch is restored perfectly, a close visual inspection reveals slight discrepancies in magnitude (the purple line is occasionally jagged or slightly lower than the blue line). This is due to interpolation error. Every time the vector is resampled, the linear interpolation formula estimates values between bins, introducing a small amount of smoothing or noise. Performing this operation twice (Stretching then Squeezing) accumulates these small numerical errors. However, the fact that the signal remains highly correlated to the original proves that these errors are negligible for the purpose of audio synthesis.

*3) Conclusion:* The successful restoration of the 440 Hz tone validates the robustness of the vector mapping logic. It demonstrates that the algorithm does not simply destroy the original data structure but transforms it in a predictable, reversible manner.

## D. Interpolation

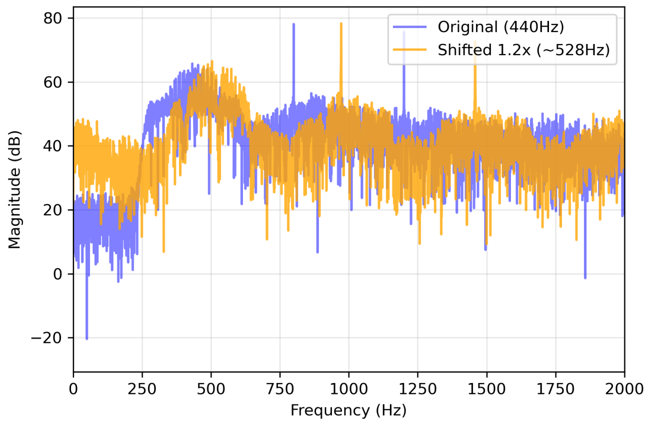


Fig. 4. Spectral density comparison for $\alpha = 1.2$. The fundamental frequency shifts from 440Hz (Blue) to approx. 528Hz (Orange), verifying the linear interpolation logic.

The final experiment evaluates the algorithm's performance under non-integer scaling conditions. We selected a shift factor of $\alpha = 1.2$, which corresponds to a musical "Minor Third" interval (ratio 6:5). Unlike the previous octave shifts (powers of 2), this transformation requires the algorithm to estimate values from fractional indices, heavily relying on the linear interpolation kernel.

*1) Visual Analysis:* Figure 4 compares the Original (Blue) and Shifted (Orange) spectra.

- Target Accuracy: The primary peak of the orange trace is located at approximately 528 Hz. Given the input of 440 Hz, the expected output is $440 \times 1.2 = 528$. The visual alignment is precise, confirming that the algorithm correctly processed the fractional scaling factor.
- Signal Clarity: Despite the need for interpolation, the orange peak remains sharp and distinct. It does not exhibit significant smearing or phase cancellation, indicating that the weighted averaging of neighboring bins preserved the signal's energy coherently.

*2) Algebraic Interpretation:* When $\alpha = 1.2$, the coordinate mapping $k \to k/\alpha$ often results in non-integer indices (e.g., mapping index 100 to index 83.33). Since our vector space $\mathbb{R}^N$ is discrete, we cannot access index 83.33 directly.

To resolve this, the algorithm computes the magnitude as a convex combination of the two nearest basis vectors, $v_{\lfloor k \rfloor}$ and $v_{\lceil k \rceil}$. Let $\delta$ be the fractional distance. The value at the new coordinate is derived via:

$$\mathbf{v}_{new} = (1-\delta)\mathbf{v}_{floor} + \delta\mathbf{v}_{ceil} \tag{14}$$

The graph demonstrates that this algebraic approximation is sufficient for high-fidelity audio synthesis. If the interpolation were flawed, the peak at 528 Hz would appear wider, shorter, or distorted due to energy leaking into adjacent frequency bins (spectral leakage).

*3) Conclusion:* Experiment 4 confirms that the system is not limited to simple integer operations (doubling or halving). The successful generation of the 528 Hz tone proves that the Linear Interpolation method effectively bridges the gap between discrete digital samples and continuous mathematical transformations, allowing for precise musical tuning.

## V. Conclusion

This study successfully demonstrated the application of Linear Algebra principles to the domain of Digital Signal Processing, specifically for the purpose of audio pitch shifting. By treating audio signals as vectors in a high-dimensional space ($\mathbb{R}^N$ and $\mathbb{C}^N$), we were able to manipulate their fundamental properties through coordinate mapping and basis transformation.

The experimental results confirmed the efficacy of the implemented algorithm across four distinct test cases:

- Vector Expansion: We verified that mapping indices $k \to 2k$ accurately doubles the frequency content, shifting an A4 (440 Hz) tone to A5 (880 Hz).
- Vector Compression: We demonstrated that fractional scaling ($\alpha = 0.5$) effectively compresses the spectral vector to 220 Hz without destroying the signal structure.
- Program Invertibility: The round-trip experiment provided empirical proof of the transformation's approximate invertibility ($T^{-1} \circ T \approx I$), validating the algebraic stability of the method despite minor interpolation noise.
- Discrete Interpolation: The successful generation of a musical Minor Third interval ($\alpha = 1.2$) confirmed that linear interpolation (convex combination) is a robust method for estimating values in discrete vector spaces.

In summary, this paper illustrates that abstract mathematical concepts—such as vector spaces, basis changes (DFT), and linear mappings—are not merely theoretical constructs but are the foundational engines that enable modern digital media processing. Future work could extend this approach by implementing Phase Vocoding to decouple pitch from playback speed, further refining the algebraic model for real-time applications.

## Acknowledgment

The author is deeply thankful to God Almighty for providing strength, determination, and opportunity to bring this paper to completion. The author also expresses deep appreciation to Ir. Rila Mandala, M.Eng., Ph.D., the lecturer of the

IF2123 Linear Algebra and Geometry course, for his dedicated guidance and support, which have been a source of inspiration throughout his teaching journey with the students.

## References

[1] U. Zölzer, *DAFX: Digital Audio Effects*, 2nd ed. Chichester, UK: John Wiley & Sons, 2011, pp. 237–297.

[2] J. O. Smith III, *Spectral Audio Signal Processing*. Stanford, CA: W3K Publishing, 2011. [Online]. Available: https://ccrma.stanford.edu/ jos/sasp/

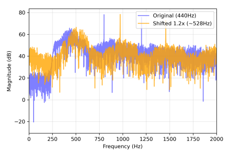
[3] M. Müller, *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Cham, Switzerland: Springer, 2015.

[4] M. Dolson, "The Phase Vocoder: A Tutorial," *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, 1986.

## Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025



Muhammad Naufal Romi Annafi
13524058

## Attachment