Zi Hang Yin

Prof: Philippe Depalle

MUMT 605

Date: Dec. 23, 2020

Phase Locked Vocoder

A phase vocoder is an audio processing technique used for time expansion/compression and pitch shifting. At the same time, a high fidelity of the sound is preserved. A phase vocoder, as per its name, requires the algorithm to have control over the synthesized signal's phases. However, with an old strategy of working on time stretching or contraction, an artifact called "phasiness" often occurs because of problems with vertical phase coherence. In order to improve the vertical phase coherence of a synthesized signal, a phase-locking technique is proposed by Miller Puckett. At the end of my paper, I propose a quantitative measurement that could be used to evaluate the vertical phase incoherence.

Part 1 - Phase Vocoder

First, I introduce a way to rescale the signal without considering the phase. Given a time-domain signal, we extract short fragments of length M that start from each starting point s_i, where $s_i = iH$, $i \in \mathbb{N}$, and H is the analysis hop size defined as $H = s_i - s_{i-1}$. Each of the time fragments is called a frame.

After each of the time segments are analyzed, they need to be synthesized in a new way. For the synthesis of the modified signal, a window function is applied to each of the frames in order to smooth the edges. Otherwise, unsmooth edges would cause sudden voltage increases and drops that result in a clipping sound of the signal. Usually, a Hanning window is chosen. The idea is to overlap each of them with a different hop size, which is the synthesis hop size that will be denoted by J, obtained by subtracting the starting time of a frame from that of the frame preceding it.

As a result, the value of the stretch factor, denoted by α, is calculated by dividing the synthesis hop size J by the analysis hop size H, meaning that the rescaled sound would be played α times slower. An audio rescaled with an α smaller than one is played $1/\alpha$ times faster. For example, an audio contracted by an α value of 0.5 means that the contracted audio is played 2 times faster.

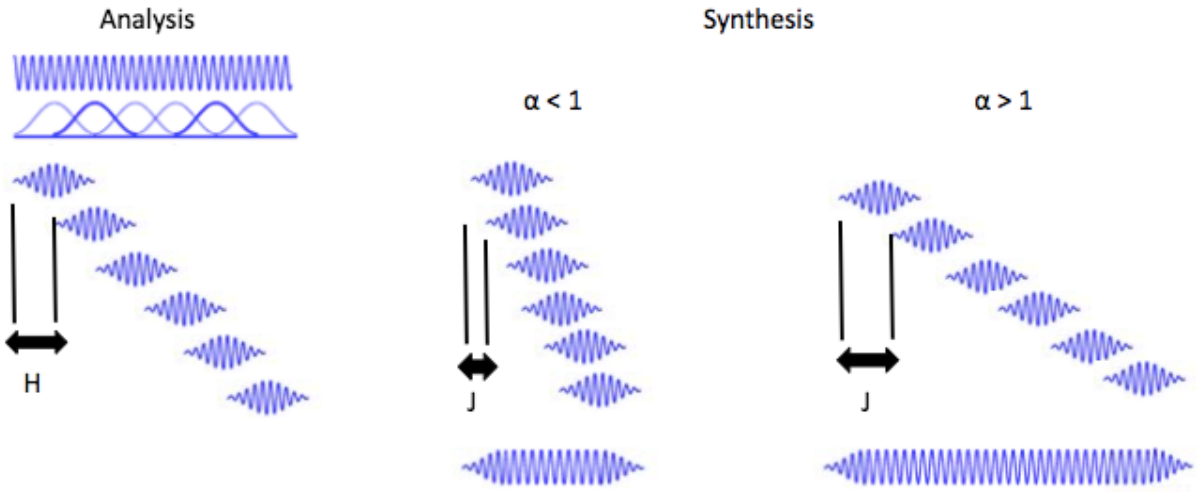

Figure 1. Time Stretch.

However, this simple version of vocoder technique does not take the phase into consideration. While the original signal has naturally coherent phases within frames, the stretched audio signal no longer preserves the phase coherence horizontally. This method would generate a very rough sound signal. A phase vocoder, as per its name, resolves the problem with the phase, and hence brings the final result's phases horizontally coherent.

The vocoder previously mentioned has both the amplitudes and the phases unchanged in each frequency component of each of its channels. In a phase vocoder, only the amplitude values are kept, the phase values will be changed. The phase updates would be different from one frequency component to another. Therefore, it is important to control the phase in the frequency domain. Fourier transform is hence needed, allowing us to select the frequency component for its phase updating.

Since each Fourier transform is done on one short time frame only, we use short-time Fourier transform (STFT) that generates a series of complex-number signals for each frame. Short-time Fourier transform, as per its name, is a technique where we need to divide the time signal into numerous short time segments with equal lengths. A window function with the same length as the short frame is multiplied to each of the frames. The result is subsequently passed to a system where a discrete Fourier transform (DFT) is performed. Ultimately, we obtain a signal with a discrete and finite frequency domain. A more detailed explanation of DFT could be found in [4] and a more detailed explanation of STFT could be found in [1].

In practice, rather than using a DFT-based method, we use the fast Fourier transform (FFT) of the windowed frame of M samples. Readers could refer to [2] for a more detailed

explanation of FFT. In this paper, I will call the output of this method simply by windowed FFT. Conventionally, each time frame has a length of 1024 samples, and each analysis corresponds to a FFT size of 1024, which means there are 1024 frequency bins.

Each frame is analyzed with an analysis hop size of H. The STFT generates a series of complex-number signals for each frame. Each complex-number value allows us to calculate its corresponding amplitude and the phase values.

After obtaining the amplitude and phase values for each frequency component, we keep the amplitude and change the phase value to another value allowing a phase coherence. The phase updating equation is:

$$ArgY[u_i, k] = ArgY[u_{i-1}, k] + d\Phi. \quad (1)$$

where $ArgY[u_i, k]$ are the phases of the current frames' bins, $ArgY[u_{i-1}, k]$ are the phases of the previous frames' bins, and the $d\Phi$ are the phase changes.

Afterwards, we convert the combination of the original amplitudes and updated phases back to a series of updated complex numbers, obtaining a new frequency spectrum signal for one frame. I will simply call it modified FFT. Inverse Fourier transform is subsequently applied to each of the modified FFT, obtaining a series of frames with updated phases. Finally, we multiply a Hanning window to each frame and overlap-add them according to the synthesis Hop size J.

The phase change value could be calculated in different ways. Without using the method proposed by Puckett, we calculate the phase change by evaluating the instantaneous frequency of a channel in a frame. The equation (1) becomes:

$$ArgY[u_i, k] = ArgY[u_{i-1}, k] + (u_i - u_{i-1})\omega[k]. \quad (2)$$

where $u_i - u_{i-1}$ is the synthesis hop size, and $\omega[k]$ is the instantaneous frequency defined as:

$$\omega[k] = (ArgX[t, k] - ArgX[s, k] + 2\pi p)/H. \quad (3)$$

where $ArgX[t, k]$ are the phases of bins in the analysis frames corresponding to the current synthesis frames, $ArgX[s, k]$ are the phases of bins in the analysis frames corresponding to the previous synthesis frames, H is the analysis Hop size, and the p is an integer value to be determined.

Puckett pointed out in [5] that this method is inefficient and inaccurate because the instantaneous frequency could be multivalued, which causes the phase change to not necessarily be well-defined.

With this limitation, Puckett proposed in [5] to obtain the phase increase values directly from the original signal rather than calculating the angular frequency, suggesting that:

$$t_i - s_i = u_i - u_{i-1}. \quad (4)$$

which could be visualized in the (Fig. 2):

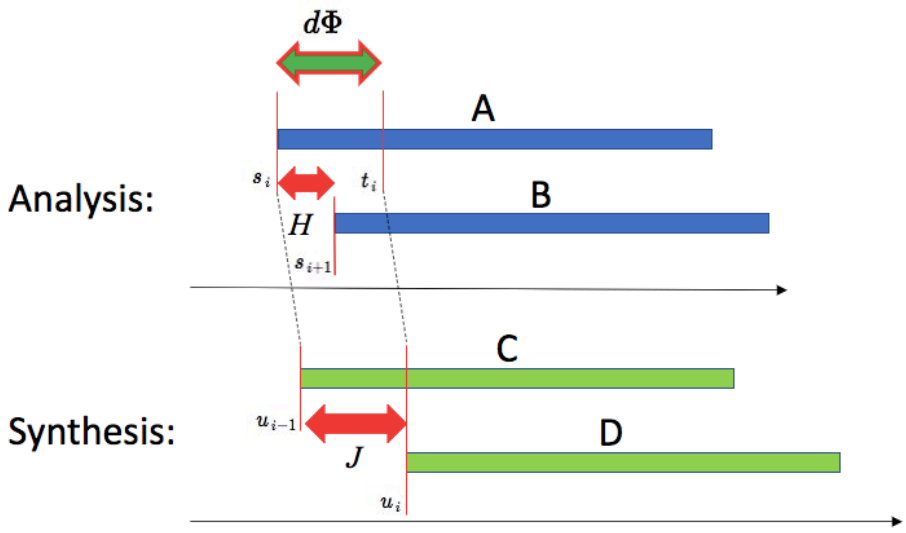

Figure 2. Obtaining the phase update from the original signal.

The frame D in the synthesis uses the amplitude of the frame B. Its phase at u_i is calculated by adding the phase change $d\Phi$ to the phase at point u_{i-1}. The phase change $d\Phi$ is calculated by subtracting the phase of point s_i from the phase of point t_i.

With the equation (4), the equation (1) becomes:

$$ArgY[u_i, k] = ArgY[u_{i-1}, k] + ArgX[t_i, k] - ArgX[s_i, k]. \quad (5)$$

As shown in [5], with mathematical deduction, the equation to calculate the phase-updated windowed FFT is obtained as:

$$Y[u_i, k] = [t_i, k]\left(\frac{Y[u_{i-1}, k]}{X[s_i, k]}\right)\left|\frac{Y[u_{i-1}, k]}{X[s_i, k]}\right|^{-1}. \quad (6)$$

where $ArgY[u_i, k]$ are the phases of the current frames' bins, $ArgY[u_{i-1}, k]$ are the phases of the previous frames' bins, $ArgX[t_i, k]$ are the phases of bins in the analysis frames corresponding to the current synthesis frames, $ArgX[s_i, k]$ are the phases of bins in the analysis frames corresponding to the previous synthesis frames.

Part 2 - Phase-locked Vocoder

With the phase update method proposed by Puckett, the synthesized signal has phases that are horizontally coherent. Hence, there is no more roughness in the sound synthesized.

However, the sound is still imperfect. For a sound signal with the pitch changing over time, in other words, with frequency components traveling from one bin to another over time, the stretched sound would sound somehow reverberant compared to the original sound. This reverberant sound comes from the interferences of different frequency components caused by the phase adjustment, where there were no such interferences in a natural sound signal.

The reason is that natural sound signals not only have phase coherence horizontally, but also vertically. Horizontal phase coherence means that the phase coherence exists from frame to frame. Vertical phase coherence means that the phase coherence exists from channel to channel. While analyzing a natural signal with a windowed FFT, we often found that, in each frame, each bin corresponding to the center of a peak was roughly out of phase with each of the two bins next to it. Therefore, the modified FFTs used to compute the stretched signal needs to keep this property.

To deduce the phasiness, maximizing the vertical phase coherence, Puckett proposed a phase-locking technique.

Without using this phase-locking technique proposed by Puckett, there was a much less efficient way to deduce the phasiness, suggesting using only channels of peaks in a frequency spectrum. However, there are two problems with the peak detections for natural sound signal:

1. You might end up getting a false peak and miss the right peaks.
2. More heuristics are needed to provide continuity from window to window because peaks migrate from one channel to another most of the time, unless the signal has a very stable frequency(i.e., a signal artificially synthesized by adding 10 cosine signals with fixed frequencies).

The phase-locked technique proposed by Puckett in [5] is surprisingly simple. It is a modification of the equation (1), replacing $Y[u_{i-1}, k]$ with $Z[u_{i-1}, k]$:

$$Y[u_i, k] = [t_i, k]\left(\frac{Z[u_{i-1}, k]}{X[s_i, k]}\right)\left|\frac{Z[u_{i-1}, k]}{X[s_i, k]}\right|^{-1}. \quad (7)$$

where the $Z[u_{i-1}, k]$ could be denoted as:

$$Z[u_{i-1}, k] = Y[u_{i-1}, k] - Y[u_{i-1}, k-1] - Y[u_{i-1}, k+1]. \quad (8)$$

where $Y[u_{i-1}, k-1]$ is value of the bin on the left side of $Y[u_i, k+1]$, and $Y[u_{i+1}, k+1]$ is the value of the bin on the right side of $Y[u_i, k+1]$.

In this method, from the complex value of a channel in the previous frame, we subtracted the complex values of the two channels on its both sides in that same frame, trying to make every pairs of adjacent bins out of phase. The reason is that the argument of a complex number is π different to the argument of the that complex number multiplied by -1.

To visualize this calculation, we could consider the $Z[u_{i-1}]$ as the result of a convolution over the frequency spectrum of $Y[u_{i-1}]$. The convolution window would have a size of three with weights –1, 1, and –1 accordingly. However, since it is a convolution, an input of 1024 bins generates an output with 1022 bins. To make sure that there are 1024 channels for every $Z[u_{i-1}]$ values, I added two bins on both ends of the output. The value of those two bins $Y[u_{i-1},0]$ and $Y[u_{i-1},1023]$ are the values of their adjacent bins $Y[u_{i-1},1]$ and $Y[u_{i-1},1022]$ multiplied by -1.

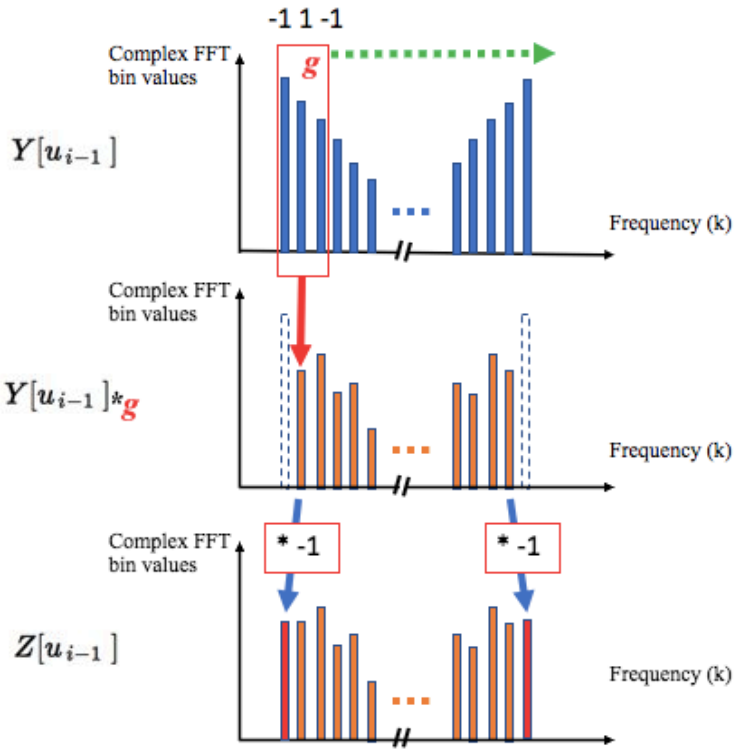

Figure 3. The calculation of Z from a convolution.

In fact, we only need these phase differences around the peaks. As it is so difficult to find the peak, especially with the peak not detectable and moving from one channel to another, we just do it for every three bins.

Part 3 - Result

Even though the phase locking technique is not perfect, the fidelity of the original signal is improved.

The result could be evaluated acoustically. The reverberant sound is reduced after applying the phase-locking technique. In the Appendix, there are several sound examples allowing comparison. Several Matlab scripts are also available for testing with any input sound.

To be able to compare the acoustic effect of the phase-unlocked vocoder and phase-locked vocoder, it is recommended to use dry sound recordings to start. A wet original sound would be difficult to allow comparison by ear. However, from each signal, we could find the location of one peak of a frame's FFT. Here is the result of an implementation on Matlab, giving phase and amplitudes values of a peak's surrounding bins in each of the three samples' one frame:

Phase	Amplitude
-2.8121	0.2830
-2.0331	0.1647
0.9888	5.7417
-2.1184	11.4725
0.8565	8.2548
3.0928	4.2475
-0.8115	2.4782

Matlab Script: fPeakPart0.m

original

Phase	Amplitude
-0.5832	0.5057
1.0787	2.5758
1.4827	25.8682
-2.0360	35.4323
-2.9174	11.6576
0.6380	2.7889
0.2429	3.0361

Matlab Script: fPeakPart1.m

Phase-unlocked

Phase	Amplitude
0.8317	0.5057
-1.8693	2.5758
1.3056	25.8682
-1.7905	35.4323
1.5333	11.6576
2.5244	2.7889
-0.6607	3.0361

Matlab Script: fPeakPart2.m

Phase-locked

Figure 4. Matlab output showing the phases and amplitudes of bins around the peaks [3].

We can see that the bin corresponding to the peak in the original signal is practically out of phase with the two bins on its both sides. Using phase-unlocked vocoder, the coherence disappeared. However, when the phase-locking technique is applied, bins become roughly out of phase once again.

However, the fact that one single peak in one single frame have bins roughly in phase does not convince us that the phase-locking method is efficient because it might be due to chance. It is also inconvenient to look at every peak in every frames. Therefore, I propose a quantitative evaluation of the phase incoherence for a sound signal, called Index-5-20. Here are the four steps to calculate the Index-5-20:

1. We first measure the amount of the phase one bin that needs to be shifted to be exactly out of phase with the next bin. I call this value Index-0 in the paper; Index-0 evaluates the vertical incoherence of two adjacent channels. All Index-0s have values between 0 and π, since it must be a positive number. If the phase that needs to be shifted for a value larger than π to one direction, it could be shifted to the other direction with a smaller value that is within the interval of $[0,\pi]$. An Index-0 of 0 means that bins are exactly out of phase,

and the signal's phase is the most coherent vertically. The larger the Index is, the less coherent vertically the two bins' phases are.

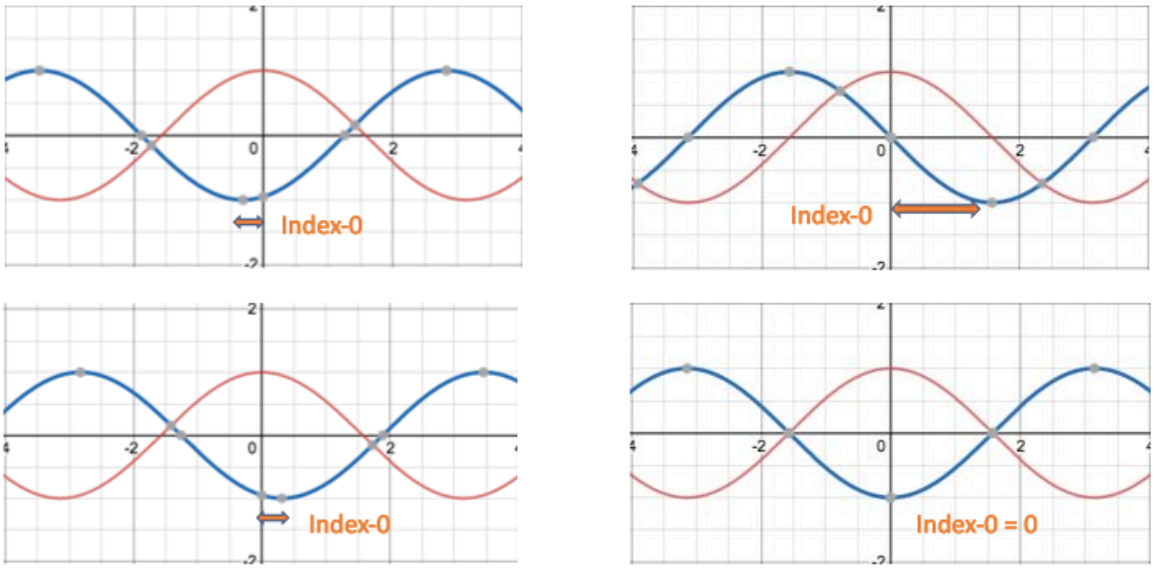

Figure 5. The Index-0 as an amount of phase that needs to be shifted.

2. For a peak, there are 3 bins. The Index-0 would need be computed twice. We calculate the average of the two Index-0s to obtain a new value. I call this value Index-1, meaning that it evaluates the vertical incoherence for one specific peak.
3. However, since Index-1 evaluates one bin only, we need a larger sample size to make it more reliable. Here, I propose an Index-5, which is calculated by taking the mean of the five highest peaks' Index-1 in a frame.
4. Index-5 evaluates one frame only. To further enlarge the sample size, I propose an Index-5-20, which is calculated by taking the mean of 20 consecutive frames' Index-5s.

With the 4 steps defined above, you could calculate the Index-5-20 of any segment of signal. Since we simply calculate the average in the three last steps, the Index-5-20 has values between 0 and π as well.

As a result, I computed the Index-5-20 of the 100-119th frames of three samples: the first from the windowed FFTs of the original signal, the second from the modified FFTs of the signal without phase locking, and the third from the modified FFTs of the signal with phase locking. Values are shown as below:

The signal analyzed	Index-5-20 Values	Matlab script used for the calculation
Original Signal	0.2750 (0.088 π)	indexCheck0.m

Phase-unlocked	1.6518 (0.526 π)	indexCheck1.m
Phase-locked	0.4097 (0.130 π)	indexCheck2.m

The Index-5-20 for the original signal is very small, close to zero, meaning that it is practically coherent.

For the phase-unlocked one, since we do not have control over the phase, the Index-5-20 is high, which is not coherent. It is noticeable that the value is about $\pi/2$ which is right in the middle of the interval. There should hence be a normal distribution of the phase value.

Finally, with the phase locking, we can bring the index back to 0.4097, which is about 0.130 π. This is the best result possible. It reflects that the phase locking is very effective, even if the phase locking technique is not perfect.

This quantitative evaluation I propose could be further used to evaluate the effect of other techniques of the phase vocoder. If a new technique could be better than the phase-locking technique presented by Puckett, the Index-5-20 calculated for the same frames of the same signal would have a value smaller than 0.4097.

In my definition of the index, I used the values 5 and 20, which could be more flexible (i.e., it could be changed accordingly). It is encouraged to use you own parameter, for example, Index-x-y. You could try computing the Index-5-20 I defined with the scripts, which can be accessed at the link provided in the Appendix, by using your own audio and your own preferred starting point.

Part 4 - Applications

The phase vocoder has a stable structure, and its applications are stable as well. Many studies have done to improve it.

The application of the phase vocoder could be used for time stretching/contraction. For example, it could be used to play an audio at a different speed.

The pitch shifting uses resampling techniques. The audio is resampled and then brought back to the original sampling rate, and then passed to a phase vocoder. If you sample an audio with half the sampling rate and then set the sample rate back to the original sampling rate, the audio is two times faster and the pitch becomes one octave higher. Then, you should apply the phase vocoder technique that changes the length back to the original length while keeping the pitch one octave higher.

A more extended way of working on the pitch shifting technique is to add a fixed-interval melody accompaniment. The Matlab script pitchShift.m in the Appendix generates a sound output that is the input signal with a parallel fourth melody accompaniment.

One limitation of this technique is that the interval of pitch shifting is fixed, while in music, intervals are usually not fixed. For example, the parallel third changes from major third to minor third while notes' pitches change. Therefore, a more advanced implementation is needed. A floating-interval accompaniment could also be achieved by adding pitch detection techniques to it and setting the pitch shifting factor. This technique could be expanded to a technique of automatic harmonization.

Appendix

Matlab Scripts:

File Names	Descriptions
simpleVocoder.m	Time stretching without phase consideration, very rough output signal.
part1.m	Time stretching with $\alpha = 1.52$, without phase locking, generating a reverberant output signal.
part2.m	Time stretching with $\alpha = 1.52$, with phase locking, generating a reverberant output signal.
fPeakPart0.m	For the FFT of a frame in the original signal, the script outputs the phases and amplitudes of bins in the surrounding of a peak to the console.
fPeakPart1.m	For the FFT of a frame used to synthesize the stretched signal without phase locking, the script outputs the phases and amplitudes of bins in the surrounding of a peak to the console.
fPeakPart2.m	For the FFT of a frame used to synthesize the stretched signal with phase locking, the script outputs the phases and amplitudes of bins in the surrounding of a peak to the console.
index5.m	The function used to find the Index-5 of a frame. The function is called by the 3 following scripts.
indexCheck0.m	For the FFTs of the 100-119th frames in the original signal, the script outputs the Index-5-20 to the console.
indexCheck1.m	For the FFTs of the 100-119th frames used to synthesize the stretched signal without the phase locking, the script outputs the Index-5-20 to the console.
indexCheck2.m	For the FFTs of the 100-119th frames used to synthesize the stretched signal with the phase locking, the script outputs the Index-5-20 to the console.
pitchShift.m	Using an input sound signal, the script output a pitch shifted sound and a polyphonic version of the input melody with parallel fourth accompaniment.

Audio Excerpts:

File Names	Descriptions
check.wav	A dry speech sound file.
simpleVocoderCheck.wav	The output of script simpleVocoder.m using check.wav as an input.
unlockedCheck.wav	The output of script part1.m using check.wav as an input.
lockedCheck.wav	The output of script part2.m using check.wav as an input.
clarinet.wav	A clarinet solo sound file.
unlockedClarinet.wav	The output of script part1.m using clarinet.wav as an input.
lockedClarinet.wav	The output of script part2.m using clarinet.wav as an input.
bobDylan.wav	A dry sound file, singing a melody of Bob Dylan.
unlockedBobDylan.wav	The output of script part1.m using bobDylan.wav as an input.
lockedBobDylan.wav	The output of script part2.m using bobDylan.wav as an input.
pitchShiftedSolo.wav	The first output of script pitchShift.m using bobDylan.wav as an input.
parallelFourth.wav	The second output of script pitchShift.m using bobDylan.wav as an input.

Matlab Scripts and Audio Excerpt available on https://github.com/fumer555/MUMT605.

References

[1] Borisagar, Komal R., Rohit M. Thanki, and Bhavin S. Sedani. "Fourier Transform, Short-Time Fourier Transform, and Wavelet Transform." *Speech Enhancement Techniques for Digital Hearing Aids*, 2018, 63–74. https://doi.org/10.1007/978-3-319-96821-6_4.

[2] Nussbaumer, Henri J. "The Fast Fourier Transform." *Fast Fourier Transform and Convolution Algorithms Springer Series in Information Sciences*, 1982, 80–111. https://doi.org/10.1007/978-3-642-81897-4_4.

[3] MATLAB. Natick, MA: The MathWorks, Inc., 2002[DVD-ROM].

[4] Olson, Tim. "The Discrete Fourier Transform." *Applied Fourier Analysis*, 2017, 75–120. https://doi.org/10.1007/978-1-4939-7393-4_3.

[5] Puckette, M. "Phase-Locked Vocoder." *Proceedings of 1995 Workshop on Applications of Signal Processing to Audio and Accoustics*, 1995. https://doi.org/10.1109/aspaa.1995.482995.

List of Related References

Laroche, J., and M. Dolson. "Improved Phase Vocoder Time-Scale Modification of Audio." *IEEE Transactions on Speech and Audio Processing* 7, no. 3 (1999): 323–32. https://doi.org/10.1109/89.759041.

Laroche, J., and M. Dolson. "Phase-Vocoder: about This Phasiness Business." *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*, 1997. https://doi.org/10.1109/aspaa.1997.625603.