Pitch Scaling of Music Signals

Théo Verhelst
Université Libre de Bruxelles

Abstract

Pitch scaling is the task of modifying the frequency of a signal while keeping its playback speed intact. Pitch scaling is an important feature of many digital music tools, and needs to be done in real-time with the highest available quality. In this report, we describe succinctly state-of-the-art techniques of pitch scaling, and we explain in detail one that is particularly suited to music signals. We also describe a Clojure implementation of this technique. This technique uses phase vocoder for time-scaling and 3rd order spline interpolation for resampling.

Introduction

Pitch scaling is a process that changes the frequency of a signal without modifying its speed. The main difficulty of pitch scaling is to make the synthesised signal as natural as the original, so that it sounds like as if it was recorded or created on this new pitch. More precisely, the timbre and the speed of the signal need to be preserved.

One application of pitch scaling is to scale the pitch of an instrument recording in a music production software, in order to tune it to the other instruments, or for any other musical purpose. But the main use of pitch scaling is probably in DJ software: one can change the pitch of a track, and therefore its key, in order to make the transition to the next track easier and smoother.

The most straightforward way to change the pitch of an audio signal is to resample the signal and playing it back at its original rate (Driedger and Müller, 2015). But both pitch and speed are modified at the same time. Thus, we need a more sophisticated technique in order to preserve the playback speed constant.

In contrast to pitch scaling, time-scale modification (TSM) is a process that modifies the speed of a signal without modifying its pitch and its timbre. TSM has been subject to many more studies than pitch scaling, but we base our work on these studies, since it is possible to show that both processes are mathematically equivalent. Indeed, in order to change the pitch of a signal, one can use a well-known TSM method, and then resample the signal.

Time-scale modification techniques

TSM techniques can be grouped in two main categories: *time-domain TSM* and *frequency-domain TSM*.

Time-domain TSM extracts portions of the input signal at a frequency defined by the scaling factor, and places them in the output signal. This technique is particularly suited to monophonic, harmonic signal, as it preserves almost perfectly the timbre of the signal. The basic time-domain algorithm, *overlap-and-add* (or *OLA*), suffers from phase jumps artifacts, but there are many variations of this algorithm that avoid this effect. But all of them are only able to correct phase jumps on the most prominent frequency, i.e. the fundamental frequency of the signal. Phase jumps can still occur in less important frequencies, i.e. the harmonics, which is clearly audible. OLA-base algorithms are thus not suited to polyphonic signals, since they contains more harmonics than monophonic ones. Furthermore, non-harmonic signals, such as drums or percussive instruments, have non-periodic patterns. These patterns are known as *transients*. A typical time-domain TSM technique leads to transient doubling or skipping (depending on the scaling factor), since these techniques periodically repeat or discard some small parts of the signal. This can be avoided by taking a very short frame size.

Frequency-domain TSM is based on the short-time Fourier transform (STFT). It splits the signal in small chunks, and computes the Fourier transform of each of these chunks, in order to get a discrete frequency-domain representation of the signal. Often, the technique also uses the *phase vocoder* in order to refine the frequencies estimates, and are thus named *phase-vocoder time-scale modification*, or *PV-TSM*, or simply phase vocoder. The idea is to preserve phase continuity across all frequencies, and not only on the most prominent frequency as in time-domain TSM, by exploiting the frequency-domain representation of the sound. PV-TSM behaves well on polyphonic signals, but are subject to vertical phase incoherence, i.e. the relationship between the phases of different frequencies at a point of time is not preserved, leading to audible artifacts, known as *phasiness*, or *loss of presence*.

General procedure

Pitch scaling process is split in two steps: 1) apply a TSM procedure 2) resample the signal. Let α be the scaling factor. We first apply a TSM procedure with parameter α, so that the playback speed is modified, while the pitch is left unmodified. Then, we resample the signal by a factor $1/\alpha$, so that the playback speed of the signal is the same as the original, but the pitch is multiplied by α.

For the TSM part, we use a simple phase vocoder with phase propagation. For the resampling part, we use an interpolator based on 3rd order splines. These are explained below.

Time-scale modification

Basics of time-scale modification

Notation: Let

$$[a:b] := \{a, a+1, \dots, b-1, b\} \quad \forall a, b \in \mathbb{Z} : a < b$$

and

$$[a:b[:= \{a, a+1, \dots, b-1\} \quad \forall a, b \in \mathbb{Z} : a < b$$

First, we have to define the basic concepts involved in time-scaling. Let the function $x : \mathbb{Z} \to [-1, 1]$ be signal to time-scale. In practice, the analysed audio signal has a finite duration of $L \in \mathbb{N}$ samples. Thus we define $x(n) = 0 \, \forall n \in \mathbb{Z} \setminus [0 : L[$ for the sake of simplicity. We want to construct the output signal $y : \mathbb{Z} \to [-1, 1]$ that have the same frequency-domain properties as x, but being stretched in time by the factor α.

Almost all TSM techniques are based on the following procedure: first, x is divided in *analysis frames* x_m, $m \in \mathbb{Z}$ having each a length of N samples, and these analysis frames are spaced by an *analysis hop size* H_a:

$$x_m(n) = \begin{cases} x(mH_a + n) & \text{if } n \in [0:N[\\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

Then, we could want to put these frames in the output signal by spacing them by the synthesis hop size H_s, achieving the correct time-scale modification. But this would cause very audible phase discontinuities, since the end of a frame would no longer match with the beginning of the next frame. Thus, we need to modify the analysis frames x_m into *synthesis frames* y_m before adding them to the output signal $y(n)$:

$$y(n) = \sum_{m \in \mathbb{Z}} y_m(n - mH_s) \qquad (2)$$

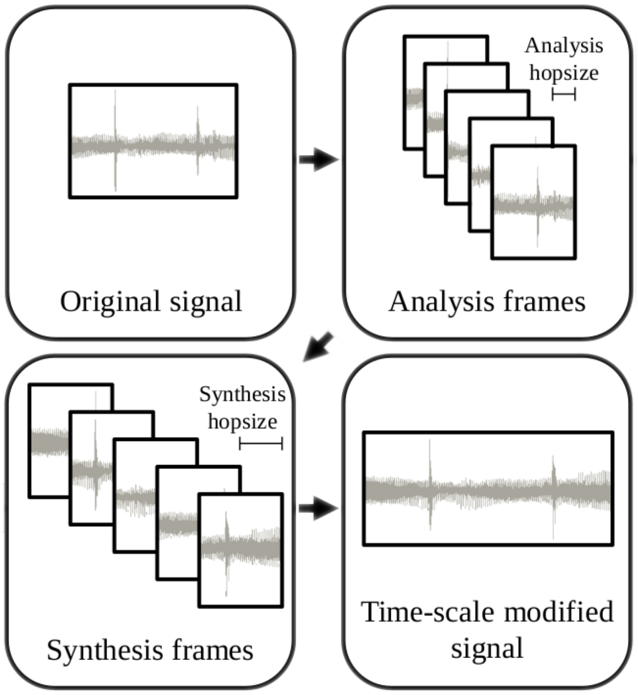

Figure 1: General procedure of time-scale modification.

H_s is usually set to $N/2$ or $N/4$, in order to have a constant overlap between the synthesis frames. And since we know that $\alpha = \frac{H_s}{H_a}$, we then have $H_a = \frac{H_s}{\alpha}$.

The method used to transform x_m into y_m is critical, as it determines the quality of the result. In addition to phase discontinuities, it must also compensate gain fluctuations: if no care is taken, two overlapping frames may have a higher gain in the overlapping part than in the rest of the signal. A common procedure to compensate gain fluctuations is to multiply each modified frame by a windowing function $w : \mathbb{N} \to [0, 1]$ before adding them to the output signal y. The Hann window is widely used for this purpose, and is defined as

$$w(n) = \begin{cases} 0.5(1 - cos(\frac{2\pi n}{N-1})) \text{ if } n \in [0:N[\\ 0 \text{ otherwise} \end{cases} \qquad (3)$$

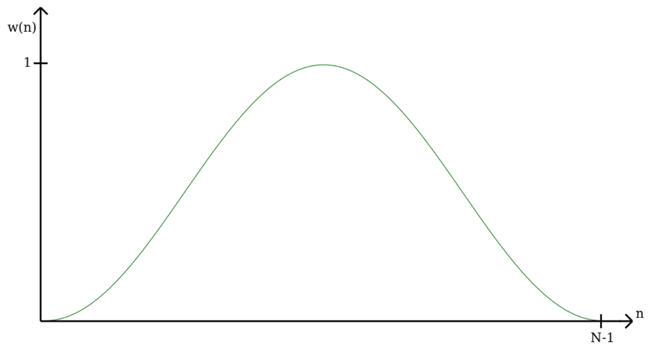

Figure 2: The Hann window function.

Phase-vocoder time-scale modification

The phase vocoder is a common frequency-domain technique used to derive the *synthesis frames* y_m from the analysis frames x_m. The idea is to compute the spectrum of the frame with the Fourier transform, and change the phases of this spectrum so that the signal re-synthesised from this modified spectrum has no phase jump with the following and previous frames.

Short-time Fourier transform The spectrum of a signal is the frequency-domain representation of a time-domain signal, and is composed by the amplitudes and phases of the Fourier series decomposition of the signal. The Fourier transform of discrete-time signals is primarily defined for signals of length N:

$$X(k) = \sum_{n=0}^{N-1} x(n) \exp(-2\pi ikn/N) \tag{4}$$

where $k \in [0 : N[$ denotes the frequency index. This yields k complex numbers, where $|X(k)|$ is the magnitude of the kth frequency, and $\arg(X(k)) \in [0, 1[$ is the phase of the kth frequency.

As required by the time-scaling algorithm, we will apply the Fourier transform on small portions of the signal. We could want to do apply directly the transform on analysis frames x_m:

$$X(m, k) = \sum_{n=0}^{N-1} x_m(n) \exp(-2\pi ikn/N)$$

But this can lead to unexpected high amplitudes in the high frequencies: the Fourier transform works as if the signal was periodic with a period N, i.e. as if we where computing the Fourier series of a signal

$$\tilde{x} : \mathbb{Z} \to \mathbb{R} : n \mapsto x_m(n \mod N) \tag{5}$$

which can have high amplitude in high frequencies around $n = aN \ \forall a \in \mathbb{Z}$. A solution is to apply a windowing function to the analysis frame, so that the signal is always zero at the frame boundaries:

$$X(m, k) = \sum_{n=0}^{N-1} x_m(n) w(n) \exp(-2\pi ikn/N) \tag{6}$$

$X(m, k)$ is the coefficient of the *short-time Fourier transform* of the signal x at time m and at frequency k. It is also named a *time-frequency bin*. w is a windowing function, usually the Hann window as defined in (3)

Note that the frequency index k corresponds to the physical frequency

$$F_{\text{coef}}(k) = \frac{F_s k}{N} \tag{7}$$

and the frame index m corresponds to the physical time

$$T_{\text{coef}}(m) = \frac{H_a m}{F_s} \tag{8}$$

Phase adjustment Our goal is to compute the synthesis frequency bins $Y(m, k)$ from the analysis frequency bins $X(m, k)$, so that the signal synthesised with the inverse Fourier transform from these synthesis frequency bins has no phase incoherence from one frame to the next. Since we are only interested in phase adjustment, we can already set

$$|Y(m, k)| = |X(m, k)| \tag{9}$$

The phase adjustment involves the observation that the short-time Fourier transform yields coefficients for a finite quantity of frequencies, and that may be insufficient to characterise exactly the underlying sinusoids involved in the Fourier series decomposition of the signal. However, it is possible to refine the frequency of the kth frequency index by using the phases of the current and next frame. This refined frequency is called the *instantaneous frequency* and is denoted by $F^{\text{IF}}_{\text{coeff}}(m, k)$.

Notation: Let

$$\phi_m := \arg(X(m, k))$$

In order to find the instantaneous frequency, we first can observe that for any frequency bin $X(m, k)$, we can compute its *unwrapped phase advance*, that is, the phase increment that should occur from the time $T_{\text{coef}}(m)$ to the time $T_{\text{coef}}(m + 1)$, according to the frequency estimate $F_{\text{coef}}(k)$.

$$\phi^{\text{inc}} = F_{\text{coef}}(k)\Delta t \tag{10}$$

where Δt is analysis hop time given is seconds:

$$\Delta t = H_a/F_s \tag{11}$$

Since we know the phase ϕ_m, we can compute the predicted phase at the time $T_{\text{coef}}(m)$:

$$\phi_m^{\text{pred}} = \phi_m + \phi^{\text{inc}} \tag{12}$$

But because of the lack of precision of the phase vocoder, this predicted phase may not be equal to ϕ_{m+1} when mapped in the range $[0, 1[$. We can then compute the difference with the effective phase ϕ_{m+1}, and this difference is called the *phase error* (or *heterodyned phase increment*):

$$\phi^{\text{err}} = \Psi(\phi_{m+1} - \phi_m^{\text{pred}}) \tag{13}$$

where the function Ψ is the *principal argument function* that maps the phase difference to the range $[-0.5, 0.5]$. Here is a possible implementation of Ψ:

$$\Psi : \mathbb{R} \to [-0.5, 0.5] : \phi \mapsto \phi - \lceil \phi - 0.5 \rceil \tag{14}$$

From this phase error, we can compute the *instantaneous frequency*: this is a refinement of $F_{\text{coef}}(k)$, in an attempt to determine the frequency of the underlying sinusoid by taking the phase error into account. This sinusoid should have a phase of ϕ_m at the time $T_{\text{coef}}(m)$ and a phase ϕ_{m+1} at the time $T_{\text{coef}}(m + 1)$.

$$\begin{aligned} F_{\text{coeff}}^{\text{IF}}(m, k) &= \frac{\phi^{\text{inc}} + \phi_m^{\text{err}}}{\Delta t} && (15) \\ &= F_{\text{coef}}(k) + \frac{\phi_m^{\text{err}}}{\Delta t} && (16) \\ &= F_{\text{coef}}(k) + \frac{\Psi(\phi_{m+1} - \phi_m - \phi^{\text{inc}})}{\Delta t} && (17) \end{aligned}$$

Result and limitations

Further improvements

Resampling

The goal of resampling is to reconstruct a continuous-time signal from the given discrete-time samples, and then sample this signal again with another sampling rate. More formally, we want to construct the continuous-time signal

$$\hat{x} : \mathbb{R} \to \mathbb{R} \tag{18}$$

such that

$$x(n) = \hat{x}(Tn) \; \forall n \in \mathbb{Z} \tag{19}$$

where T is the sampling period (the inverse of the sampling rate). Then, we sample a new signal y at a sampling period T':

$$y(n) = \hat{x}(T'n) \; \forall n \in \mathbb{Z}$$

In order to construct the continuous-time signal, we need an interpolator. There exists various interpolators, such as truncated sinc, linear-interpolator, b-spline interpolator, Lagrange interpolator. We chose 3rd order spline interpolator, for its simple implementation, although it does not give the best results for musical signal interpolation.

For 3rd order spline interpolation, we search the factors of a 3rd-degree polynomial for each pair of consecutive signal values $(x(n),\, x(n + 1))$, such that this polynomial passes through these values. For the sake of the notation, let

$$y_0 = x(n-1);\; y_1 = x(n);\; y_2 = x(n+1);\; y_3 = x(n+2)$$

We search a function

$$f : [0, 1] \to \mathbb{R} : t \mapsto \sum_{i=0}^{3} \alpha_i t^i = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 \tag{20}$$

In order to determine the value of the factors α_i, we need to set four constraints on the function f:

$$\begin{cases} f(0) = y_1 & (21) \\ f(1) = y_2 & (22) \\ f'(0) = \dfrac{y_2 - y_0}{2} & (23) \\ f'(1) = \dfrac{y_3 - y_1}{2} & (24) \end{cases}$$

(21) and (22) are natural constraints of spline interpolator: we want that the interpolating function passes through the supplied values $(x(n),\, x(n + 1))$. But in order to determine the factors α_i, we need two more constraints. One solution is to use Hermitian splines, that is, the derivative of f at $t \in \{0, 1\}$ is equal to the derivative of a straight line between the previous and the next point ((23) and (24)). By expressing the equations as a matrix equation, we can find that

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ -3 & 3 & -1 & -0.5 \\ 2 & -2 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_2 - y_0 \\ y_3 - y_1 \end{bmatrix} \tag{25}$$

For each pair of samples, we just have to compute the four factors α_i with this equation, and then evaluate the function f as in (20), at a value of t corresponding to the new sampling rate.

Clojure implementation

The Clojure programming language

Overtone

Implementation of pitch scaling

Results

Conclusion

References

Driedger, J. and Müller, M. (2015). A review of time-scale modification of music signals. In Valimaki, V., editor, *Proceedings of the International Conference on Digital Audio Effects (DAFx)*.