Spring 2018

Time and Frequency Independent Manipulation of Audio in Real Time

Christian Albert Yost
Bard College

Time and Frequency Independent Manipulation of Audio in Real Time

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Christian Yost

Annandale-on-Hudson, New York
May, 2018

Abstract

Analog audio implies time-frequency dependence. With digitally sampled audio, this time-frequency dependence can be broken and either variable can be manipulated independently of the other, in real time. This paper will mostly focus on the frequency domain algorithm called the *Phase Vocoder* which breaks this time-frequency dependence. We will start by looking at Fourier Theory and the effect of discrete sampling. Then we will look at the Phase Vocoder's theory of operation, as well as improvements made by Puckette, Laroche, and Dolson, to name a few. Through all of this, simple examples will be presented in order to gain intuition into the principles at hand. Towards the end, a time domain approach for time-frequency independence called *Granular Synthesis* will be explored. We will compare it to the Phase Vocoder, and see how our understanding of one changes how we think and make decisions for the other. Finally we will propose some ideas for further improvement to real-time time-frequency independent manipulation of audio.

Contents

Dedication

This project is dedicated to Joel Embiid, Ben Simmons, Dario Saric, Robert Covington, JJ Redick, Markelle Fultz, TJ McConnell, Marco Belinelli, Ersan Ilyasova, Richaun Holmes, Justin Anderson, and Amir Johnson. Trust the Process.

Acknowledgments

I would like to thank Matthew Deady and Matthew Sargent for all of their help and generosity; Paul Hembree for some inside experience and personal advice; and Cort Lippe for coming through with guidance in tricky territory at the right time. Lastly I would like to thank my parents.

1
Introduction

Understanding how information is represented in signals is always the first step in successful DSP [7]. The original goal of this project was to gain a nuanced understanding of the Phase Vocoder. Along the path created with that aim, many stops are made to better understand the processes that go into such an algorithm. We will explore properties of the FFT in order to try and gain a more complete understanding. Contained in this project is information about the Fourier Transform and its inverse as it applies to Digital Signal Processing, from the point of view of an undergraduate who has background in mathematics and music. The aim of this project is simply as stated above: to understand how information is represented in signals. To achieve this, properties, theorems, and applications of the Fast Fourier Transform are investigated and often demonstrated with simple examples in order to instill the roots of intuition. Our signals of interest are musical signals, and therefore we place the additional constraint that any signal processing done must result in a signal that is pleasing to the ear. This project is meant as a guide for somebody who enjoys manipulation of digital signals. Some experience with Max/Msp/PD and understanding of simple math concepts is assumed.

To start we will look at an overview of Fourier Theory and specifically how it applies to digital signal processing. Included are properties of the transform, a conceptual view of Fourier

Theory as a change of basis to a function space, and some tricks used to improve the performance of the Discrete Fourier Transform, as well as a look at a simple DFT calculation program.

Then we will explore how these ideas allow for the time modification of signals. The Phase Vocoder is an algorithm first proposed in the 1960's by Flanagan and will be investigated intensely. The core observation was to construct signals based on the phase derivative. We will note some peculiarities when creating the Phase Vocoder, as well as some improvements proposed by the creator of Max, Miller Puckette.

After we have a solid understanding of the Phase Vocoder as well as some idea of how to remedy certain ailments, we look at the work of Laroche and Dolson, who have put the largest stamp on our approach to the Phase Vocoder since it was invented. These include Identity Phase Locking and Scaled Phase Locking. Some notes and approaches to implementing Identity Phase Locking in real time are made. Observations made by other engineers in reference to Laroche and Dolson will be mentioned as well. The most notable of these ideas is how the nonlinearity of human hearing affects Identity and Scaled Phase Locking.

Finally we will look at other, in between time and frequency domain approaches to modifying the duration of a signal independent of its frequency. This include the ideas of Dennis Gabor, Granular Synthesis, and the Grain Vocoder. I then propose a Phase Vocoder which recognizes the problem of multiple harmonic sources occupying the same spectral space. Using Cepstrum Analysis to separate individual harmonic sources, the Saphe Covoder hopes to remedy the problems caused by spectral crowding.

I hope to bridge the gap between the DSP engineers who pioneered what we are about to get into, and the signal processing hobbyists who use these ideas everyday, and would like to dig a

bit deeper into what is going on, in order to better understand how information is represented in signals for more successful DSP.

2
Fourier Theory and the Fourier Transform

Fouriers theory states the following: any periodic signal can be represented as a sum of sinusoids. A Fourier Transform looks like:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt$$

where we input a function of time, and return a function of frequency. Similarly, the Inverse Fourier Transform takes a function of frequency and returns a function of time.

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}d\omega$$

These equations tell us how to go from a representation in time to a representation in frequency, and back, without loss of information. It is important to notice here that both integrals have endpoints that stretch from negative to positive infinity. These formulas use parentheses to denote continuous time and frequency. This is a Digital Signal Processing (DSP) project, and such endpoints, domain properties, and in fact integration itself, are of little use to us. Therefore we will be focusing on the following: the Discrete Fourier Transform (DFT). For an analysis time interval T,

$$\text{where } t_n = n \cdot \frac{T}{N}, \text{ and } \omega_k = k \cdot \frac{2\pi}{T}$$

$$X[\omega_k] = \sum_{n=0}^{N-1} x[t_n]e^{-i\omega_k t_n} = \mathcal{F}[x[t_n]]$$

and the Inverse Discrete Fourier Transform (IDFT)

$$x[t_n] = \sum_{k=0}^{N-1} X[\omega_k] e^{i\omega_k t_n} = \mathcal{F}^{-1}[X[\omega_k]]$$

It is important to notice how these latter transforms have changed from the former. Let us first start with time. In the DFT, our time variable is placed in brackets. This denotes discrete time, rather than continuous time. Why discrete time? For the remainder of this project we will be using *digital* processing techniques, where a function is *sampled* once over a specified time interval, T. Our frequency variable ω_k also appears in brackets because it is also not continuous. Since we are no longer dealing with continuous time and frequency functions, our integrals have now become finite sums over N discrete points, each point taken every $\frac{1}{f_s}$ seconds where f_s is our sampling frequency with units samples per second.

The output of the forward DFT, $X[\omega_k]$, is frequency domain data. The DFT takes N time domain points as input, and produces $\frac{N}{2}$ complex frequency domain points called *Fourier Coefficients*. These Fourier coefficients look like $a + bi$, where a is the real component and describes how much $cos(\omega_k)$ is in the signal, or how similar the signal and $cos(\omega_k)$ are; and b is the imaginary component and describes how much $sin(\omega_k)$ is in the signal, or how similar the signal and $sin(\omega_k)$ are. We will get into what we mean by "how similar" later. Fourier coefficients that look like this are in *cartesian* form. There is one other form for Fourier coefficients, *polar* form.

We arrive at the polar representation of Fourier coefficients using Euler's identity

$$A \cdot e^{i\theta} = Acos(\theta) + Aisin(\theta)$$

where A is the amplitude and θ is the phase. We obtain the amplitude and phase information for a particular frequency ω_k from the cartesian form where

$$A = Mag(X[\omega_k]) = |X[\omega_k]| = \sqrt{a^2 + b^2} \text{ , and}$$

$$\theta = arg(a + bi) = tan^{-1}\big(\frac{b}{a}\big)$$

numbers that look like these are called *complex exponentials*, and we see them in our equations for the DFT. This form of Fourier coefficients is perhaps easier to understand than the former, especially for audio applications, perhaps because it conveys something closer to how we already talk about sound. For example, if for a Fourier coefficient $X[\omega_k]$, its amplitude is large, then we hear the frequency ω_k more in whatever chunk of sound we are analyzing. Fourier coefficients in polar form describe a *complex sinusoid*. A complex sinusoid is a 3 dimensional plot where we look at the complex plane versus time. Here we can see how the cartesian and polar forms of Fourier coefficients are describing the same thing. For $X[\omega_k] = a + bi$, the real a, or cosine, component describes how the horizontal dimension of the complex sinusoid changes with time, while the imaginary b, or sine, component describes how the vertical dimension of the complex sinusoid changes with time. The behavior of a complex sinusoid is commonly describes as a corkscrew motion, which we can see in figure 2.0.1.

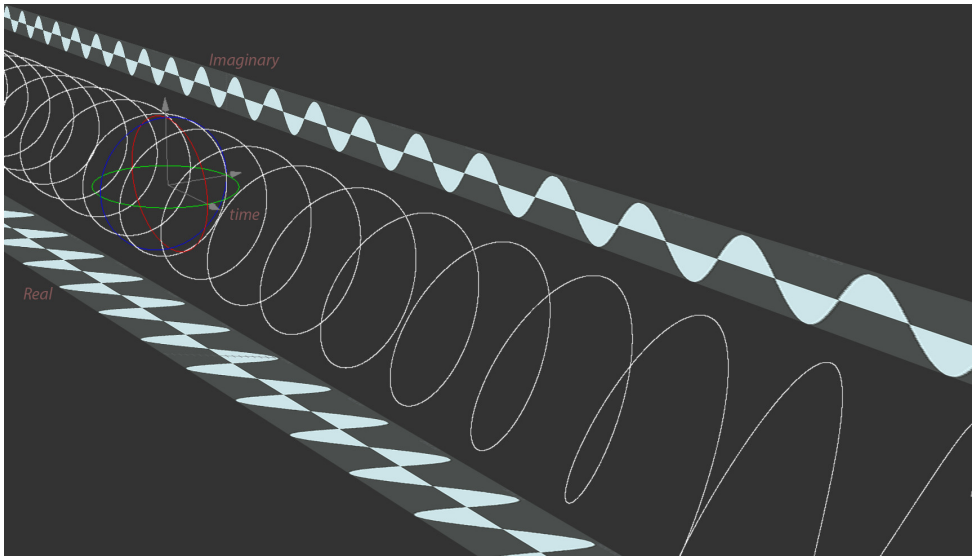


Figure 2.0.1. A complex sinusoid with its individual components graphed vs. time

So if such a motion is described by each Fourier coefficient $X[\omega_k]$, and there are Fourier coefficients for $\frac{N}{2}$ unique frequencies per frame, we can think of our frequency domain representation as data for a bank of N oscillators, a sine and cosine wave for each frequency ω_k, that when summed together give us the analysis time domain signal.

The DFT produces $\frac{N}{2}$ complex frequency domain points. So out of all frequencies $\omega \in \mathbb{R}$, which frequencies does the DFT produce data for? For a sampling period $T = \frac{N}{f_s}$, where f_s is the sampling frequency, the DFT produces data for all frequencies ω_k where

$$\omega_k = \frac{k \cdot 2\pi}{T} \text{ radians per second}$$

where $k = 0, 1, 2, ..., \frac{N}{2}$. Here we can see that the frequencies the DFT produces data for are the first $\frac{N}{2}$ harmonics of the frequency $\frac{2\pi}{T}$ radians per second. $\frac{2\pi}{T}$ radians per second is the fundamental frequency of the DFT, and has the value

$$\omega_1 = \frac{2\pi}{T} = \frac{2\pi}{\frac{N}{f_s}} = \frac{2\pi f_s}{N} \cdot \frac{1 \text{ cycle}}{2\pi \text{ radians}} = \frac{f_s}{N} \; Hz$$

The max frequency, the $k = \left(\frac{N}{2}\right)^{th}$ harmonic, that the DFT produces data for is

$$\omega_{\frac{N}{2}} = \frac{\frac{N}{2}2\pi}{T} = \frac{N}{2} \cdot \omega_1 = \frac{N}{2} \cdot \frac{f_s}{N} = \frac{f_s}{2} \; Hz$$

For the case that $k = 0$, $X[\omega_{k=0}]$ tell us the average value, or the DC offset of the signal over the sampling interval T. Thus we see that an N point DFT produces data for frequencies 0 to $\frac{f_s}{2}$, linearly, in increments of $\frac{f_s}{N} Hz$. The value $\frac{f_s}{2} Hz$ is called the *Nyquist Frequency* or the *Nyquist Rate*. Why does the DFT stop at $\frac{f_s}{2} Hz$? It's not that the DFT filters out frequencies about the Nyquist Frequency, but rather that as frequencies get higher than $\frac{f_s}{2}$, because of sampling, they start to look to the DFT like other frequencies. What sampling means is that we only get one look at the signal every $\frac{1}{f_s}$ seconds, and from the collection of looks at the signal, or samples, we can try and guess what is happening in the original signal. Therefore we can see that for any sinusoid which completes a cycle in under 2 samples, when we patch together the looks we got of it, the result will look vastly different than the original signal, since we will have missed all of the original signal's behavior in between sampling instances, which for high frequencies is most of the behavior.

Consider the complex exponential form of our analysis frequencies,

$$e^{i\omega_k t_n} = e^{i\frac{k2\pi}{T} \cdot \frac{nT}{N}} = e^{i\frac{2\pi nk}{N}} = (e^{i\frac{2\pi n}{N}})^k$$

We see that our frequencies are the k^{th} powers of the N^{th} roots of unity around the unit circle. Thus we see that for any frequency $\omega_k = \frac{2\pi}{N}k$ radians per second, it can have the following values at times $t_n = \frac{n*T}{N}$. The following figures illustrate the "snapshot" behavior of sampling.

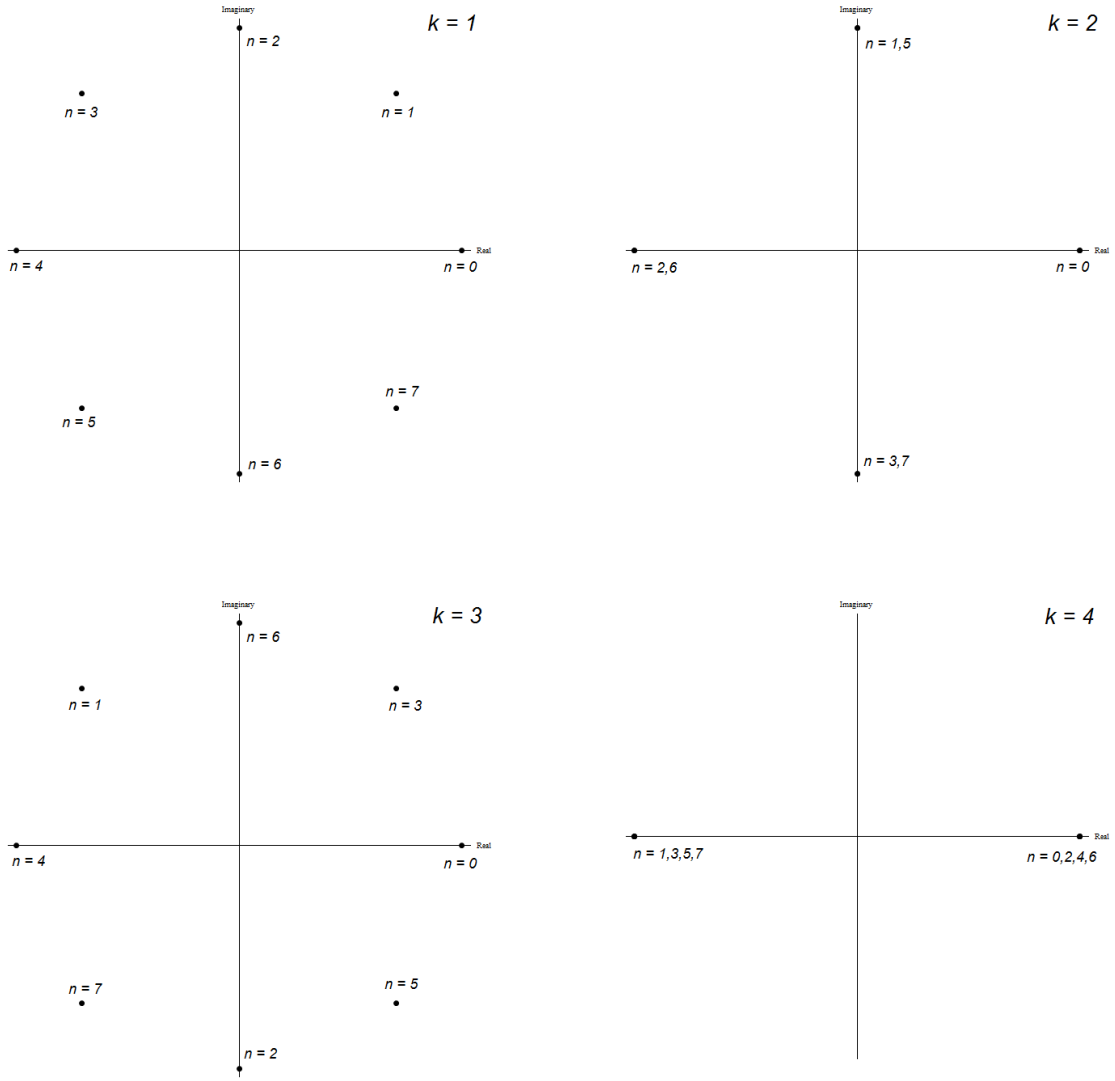
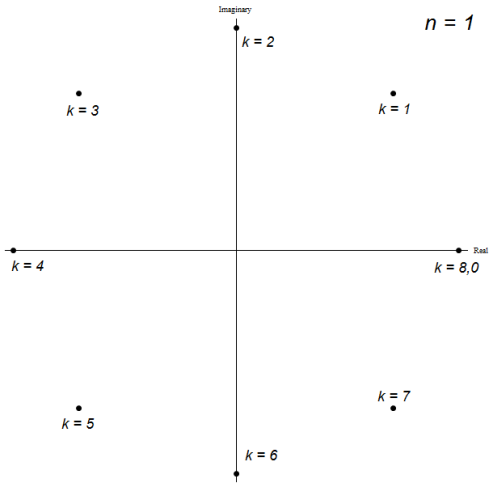
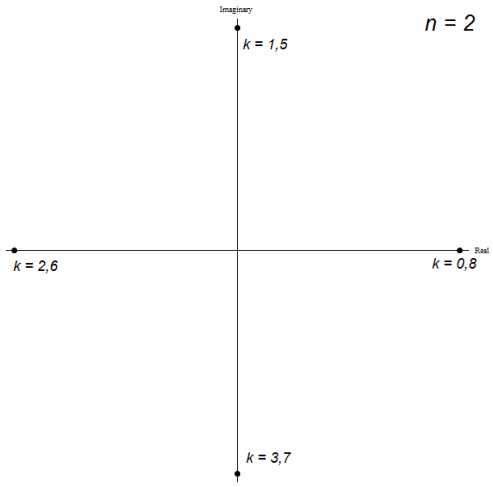


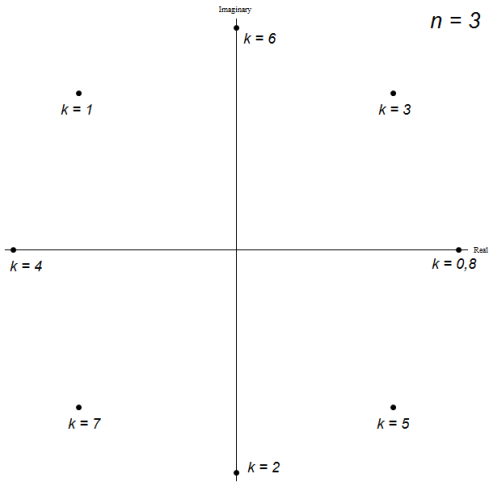
Figure 2.0.2. Possible values during analysis time period T for a frequency ω_k

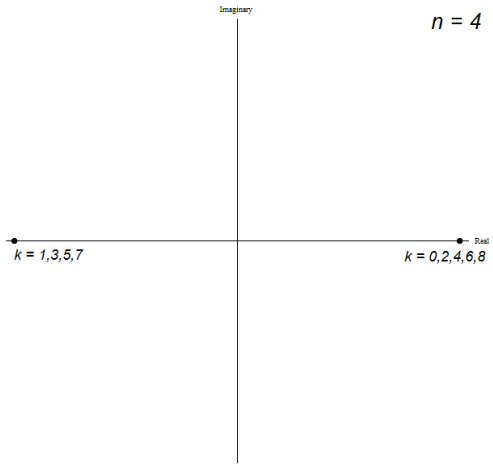
Now let's instead fixed the time point t_n and see what different frequencies are "allowed".

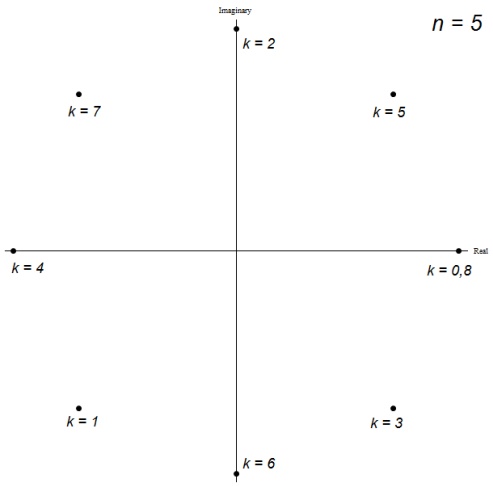
This latter set of figures gives us a good look at the sampling process, and how that translates to only a finite set of analysis frequencies. We also see that data for $\frac{N}{2} < k \leq N$ has been included. However, earlier we stated, and intuited, that only data for the frequencies corresponding to $k = 0, 1, ..., \frac{N}{2}$ was produced by the DFT. Notice that for any frequency ω_{N-p}, where $0 \leq p < \frac{N}{2}$,

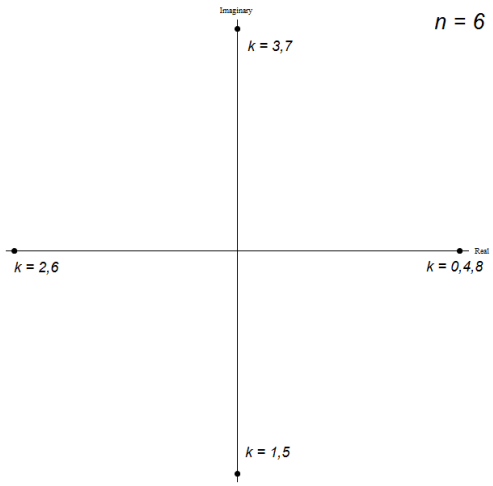

n = 1
Imaginary
Real
k = 2
k = 3
k = 1
k = 4
k = 8,0
k = 7
k = 5
k = 6


n = 2
Imaginary
Real
k = 1,5
k = 2,6
k = 0,8
k = 3,7


n = 3
Imaginary
Real
k = 6
k = 1
k = 3
k = 4
k = 0,8
k = 7
k = 5
k = 2


n = 4
Imaginary
Real
k = 1,3,5,7
k = 0,2,4,6,8


n = 5
Imaginary
Real
k = 2
k = 7
k = 5
k = 4
k = 0,8
k = 1
k = 3
k = 6


n = 6
Imaginary
Real
k = 3,7
k = 2,6
k = 0,4,8
k = 1,5

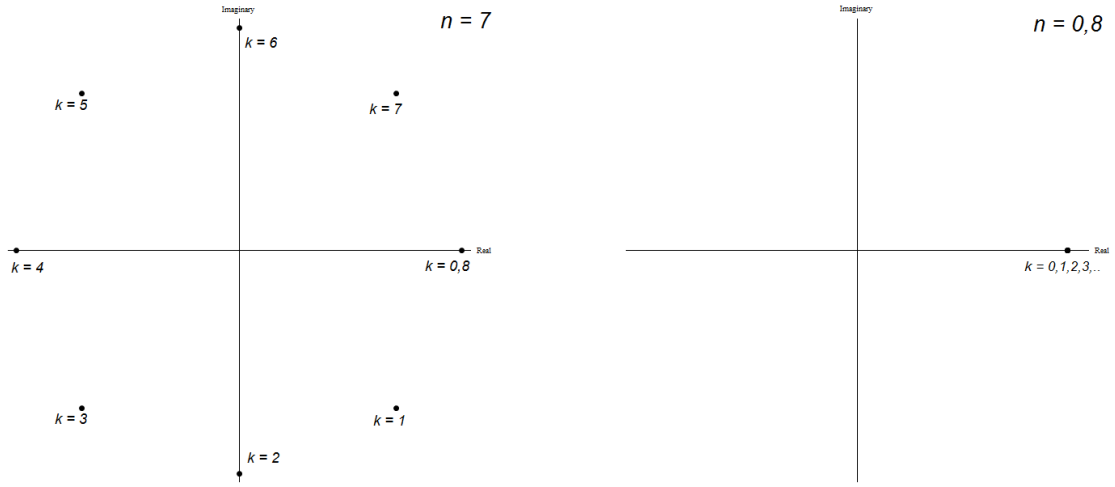


Figure 2.0.3. Possible frequencies ω_k at a certain time point t_n

in other words $k > \frac{N}{2}$, we see that

$$\omega_{N-p} = \frac{2\pi}{T} \cdot (N-p) = \frac{2\pi N f_s}{N} - \frac{2\pi f_s p}{N} = 2\pi f_s - \frac{2\pi f_s p}{N} = -\frac{2\pi f_s p}{N} = -\omega_p$$

Here we see that the frequencies corresponding to $\frac{N}{2} < k < N$ are just the negatives of frequencies below the Nyquist Frequency. We will focus on this mirrored information later, and why we need it. Furthermore, for any frequency ω_{N+q}, where $0 < q \leq \frac{N}{2}$, we see that

$$\omega_{N+q} = \frac{2\pi}{T} \cdot (N+q) = \frac{2\pi N f_s}{N} + \frac{2\pi f_s q}{N} = 2\pi f_s + \frac{2\pi f_s q}{N} = \frac{2\pi f_s q}{N} = \omega_q$$

In this way, frequencies above the sampling frequency simply look like frequencies below the Nyquist Frequency, as we can see in the following figure. When a frequency is moved to another frequency band in the spectrum, it is called *aliasing*. In the following figure, we can see how the frequency 44219 Hz is aliased to 119Hz when the sampling rate is 44100 samples per second, however if we raise the sampling rate to 96000 samples per second, we see that the frequency is no longer aliased.

As we can see in the above figure, when a signal contains frequencies higher than the Nyquist Frequency, they are reconstructed in the time domain erroneously. This is one limitation of the DFT. In order to avoid this, signals are *band limited* before frequency domain analysis. Band limiting is the process of filtering out frequencies above the Nyquist Frequency in order to avoid

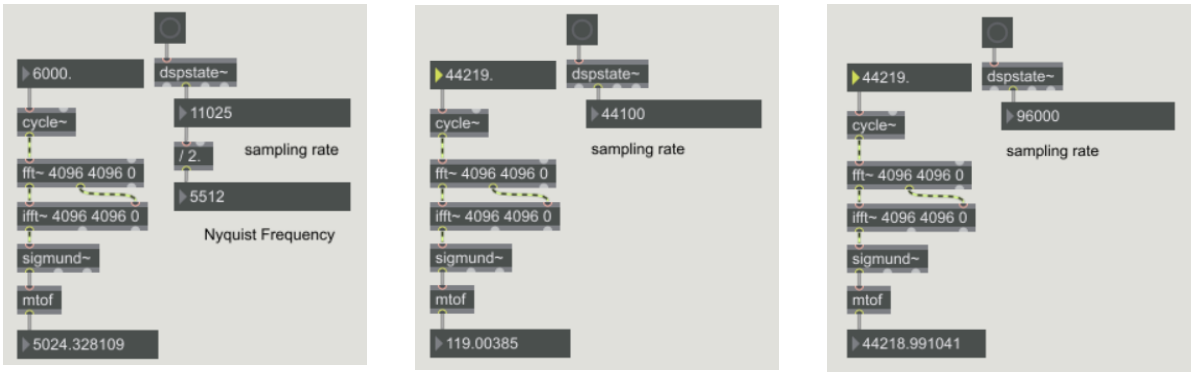


Figure 2.0.4. Frequency aliasing based on sampling rate

reconstructing the time domain signal incorrectly. This is usually done by applying a filter with a square shaped frequency response and unity gain for frequencies 0 - $\frac{f_s}{2}Hz$, and zero everywhere else.

We normally think of frequency as a function of time: by inputting a time and output the value of a sinusoid at that time. However, there is a slight tweak to this model that will become useful conceptually later on. Instead of thinking of frequencies as functions of time, we want to sometimes think of frequencies as functions of phase. We are able to go between the two domains seamlessly since both time and phase are linear. In between inputting time and outputting a value, we convert the time to where the sinusoid is in its cycle, or in other words its phase. This is done by multiplying the time by however many radians the frequency completes over a specified period of time. Thinking about sinusoids in this way is common in DSP, for example when we think of updating the phase for a digital driven oscillator, as pointed out by Puckette in [6]. Thus we can also think of the above figures which fix the frequency k and show the values at time n of instead showing the values at phase ϕ.

As we stated earlier, the DFT returns $\frac{N}{2}$ complex frequency domain points for frequencies $0 - \frac{f_s}{2}Hz$, spaced linearly $\frac{f_s}{N}Hz$ appart. So each complex point is actually representing data for a range of frequencies. The Fourier coefficient $X[\omega_k]$ represents data for frequencies $\frac{kf_s}{N}Hz$

to $\frac{(k+1)f_s}{2N}Hz$. Because of this, sometimes data for multiple frequencies can be analyzed as if it were a single frequency, and after frequency domain modification has occurred, be reconstructed into the time domain as if it were such. This is another limitation of the DFT. Because western musical harmony is built off of power multiples of a fundamental frequency, i .e. exponential, and the DFT spectrum is spaced linearly, we sometimes get not enough information about certain frequencies, and too much about others. The lower bins of the DFT contain octaves of frequency information, while the higher bins contain cents of frequency information. This is an unfortunate pairing, since the lower half of the spectrum usually contains more important musical information. Spectral manipulation of low frequencies is sometimes impossible without altering other frequencies present in the signal.

For example, if you had an FFT with a frame size set to $N = 1024$ samples, and a sampling rate of 44100 samples per second, the frequency range per bin would be $\frac{f_s}{N} = \frac{44100}{1024} = 43.066Hz$. So if you had an input time signal with a frequency of $20Hz$ in it, as well as its first harmonic, $2 * 20 = 40Hz$, it would be impossible to alter the frequency information of the $20Hz$ part of the signal without also altering the $40Hz$ part of the signal. If you know what sort of frequency domain manipulation you would like to do, any information about the input signal is very valuable, since you can then make more intelligent decisions on FFT variables. So in the above case, if you knew before analysis that you wanted to manipulate the $20Hz$ part of the signal, you would wisely choose a DFT size of $N = 4096$ so that $\omega_1 = \Delta f = \frac{f_s}{N} = \frac{44100}{4096} = 10.77Hz$. Now the $20Hz$ and $40Hz$ components of the signal would have their own spectral bins, and data for each frequency could be varied independently of the other.

The fundamental frequency of the DFT has a large effect on the spectral data produced by the DFT. The more similar the DFT's fundamental frequency ω_1 and the frequency content of the input signal are, the cleaner the data produced by the DFT is going to be. Consider 2

signals,

$$f_1 = sin(172.265625Hz), \text{ and } f_2 = sin(173Hz)$$

being analyzed by a DFT where $N = 2048$, so $\omega_1 = \frac{f_s}{N} = \frac{44100}{2048} = 21.533Hz$. f_1 is the 8^{th} harmonic of the DFT's fundamental frequency, while f_2 is not related to the harmonic series of the DFT. The DFT gives the following amplitude spectra

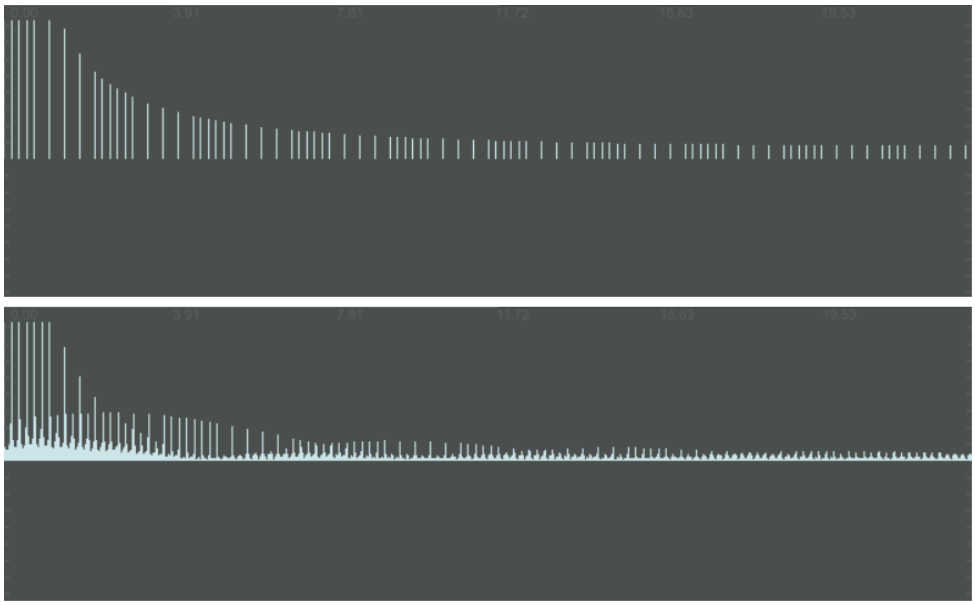


Figure 2.0.5. Spectra generated from a 2048 point FFT with a fundamental frequency of 21.5332031 Hz. *top* Amplitude spectrum of f_1. *bottom* Amplitude spectrum of f_2.

Here we see that the DFT is inherently biased towards signals whose frequency content more closely resembles the harmonic series of the DFT's fundamental frequency ω_1.

So at first glance it would seem always advantageous to choose a larger N, since then we would always have more accurate frequency domain data. However, when processing real time musical signals, we must keep in mind the cost of higher frequency resolution. Our frequency domain data takes longer to be produced, as well as longer to be resynthesized, the larger N is. Because we need a larger N for more precise frequency domain data, higher frequency resolution comes at the cost of longer computation time .So in order to obtain more accurate *pitch* information on a real time signal, we must sacrifice the precise timings that events in the signal will occur.

And if we think of events in a signal which are time specific as *rhythm*, we see that frequency domain resolution of pitch and time domain rhythm are inversely proportional.

2.1 Properties of the FFT

2.1.1 Linear Transform

The Fourier Transform is a *linear transformation*, meaning that it is *homogeneous* and *additive*. For a transform to be homogeneous, it must be the case that scalar multiplication in one domain results in identical scalar multiplication in the other domain. We can easily intuit this: if we increase or decrease the amplitude of a signal, it seems obvious that the frequencies that make up that signal would also all increase or decrease, respectively, in amplitude. This in fact is the case, as we see in the following

$$k \cdot x(t) = k \cdot \int X(\omega) e^{-i\omega t} d\omega = \int k \cdot X(\omega) e^{-i\omega t} d\omega$$

For rectangular coordinates, this means that both the real and imaginary parts of each $X(\omega)$ are multiplied by k, since if $X(\omega) = x + iy$, then

$$k \cdot X(\omega) = k \cdot (a + bi) = k \cdot a + k \cdot bi$$

For polar coordinates, if $Mag[X(\omega)] = r$, then we see that $Mag[k \cdot X(\omega_k)]$ is

$$Mag[k \cdot X(\omega)] = \sqrt{(ka)^2 + (kb)^2} = \sqrt{k^2a^2 + k^2b^2}$$

$$= \sqrt{k^2(a^2 + b^2)} = \sqrt{k^2}\sqrt{a^2 + b^2} = k \cdot r = k \cdot Mag[X(\omega_k)]$$

For a transform to be *additive*, it must be the case that addition in one domain corresponds to addition in the other domain. If two time signals $x_1(t)$ and $x_2(t)$ have frequency domain representations $X_1(\omega)$ and $X_2(\omega)$, respectively, we see that

$$x(t_1) + x(t_2) = \int X_1(\omega) e^{-i\omega t} d\omega + \int X_2(\omega) e^{-i\omega t} d\omega$$

$$= \int (X_1(\omega) e^{-i\omega t} + X_2(\omega) e^{-i\omega t}) d\omega = \int (X_1(\omega) + X_2(\omega)) e^{-i\omega t} d\omega$$

For rectangular coordinates, we simply sum $X_1(\omega)$ and $X_2(\omega)$ point by point, in other words

$$X_1(\omega_k) + X_1(\omega_k) = (a_1 + b_1 i) + (a_2 + b_2 i) = (a_1 + a_2) + (b_1 + b_2)i$$

While the algebra works nicely for rectangular representations, there is unfortunately no simple computation for polar coordinates. Intuitively we know to simply add the two vectors corresponding to $X_1(\omega_k)$ and $X_2(\omega_k)$ in the classic "nose to tail" fashion. However, mathematically this always results in converting to rectangular coordinates. We will come back to polar and cartesian coordinate representations and calculations in Chapter 4.

2.2 Phase

In order to motivate why we should investigate phase in the first place, let us start off with a simple observation. Consider the signal $x[t_n]$ and its Fourier Transform $X[\omega_k]$. Now let's generate two new signals $x_1[t_n]$ and $x_2[t_n]$ from the Inverse Fourier Transform of $X[\omega_k]$, such that $x_1[t_n]$ is constructed from the magnitude spectrum of $X[\omega_k]$ and a set of random numbers for its phase information; while $x_2[t_n]$ is constructed from the phase of $X[\omega_k]$ and a set of random numbers for its magnitude information.

We see that $x_2[t_n]$, which retained the phase information, shares the edges of the original $x[t_n]$, while $x_1[t_n]$, which retained the magnitude information, does not resemble the $x[t_n]$ at all. From this we can learn that it is the phase information that encodes *when events happen* in a signal. As we build up to looking at the Phase Vocoder, keep in mind that the *time* of the signal is precisely what we are concerned with. Therefore it makes sense that we should be processing the phase information of a signal!

2.2.1 Unique Reconstruction

In the previous example, each signal's magnitude spectrum was different from the other's. One reconstructed signal's magnitude spectra was random, and the other signal's magnitude spectra was that of a square wave. However, what if two signals have the same magnitude spectrum, does that mean they have the same time representation? The magnitudes alone are not enough

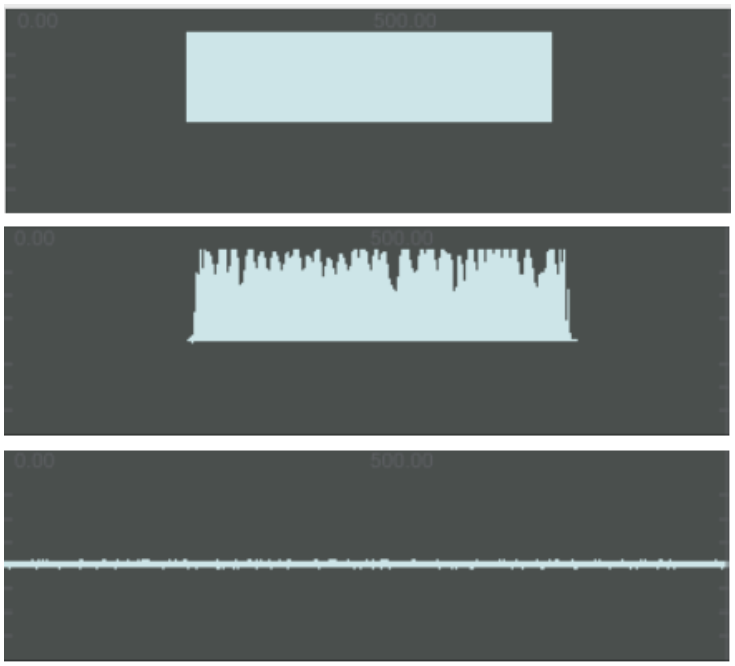


Figure 2.2.1. *Top* original square wave. *Middle* wave synthesized from phase information of a. but random magnitude values. *Bottom* wave synthesized from magnitude information of a. but random phase values. See appendix for code.

information to properly reconstruct the time signal they represent. As we will see, signals with identical magnitude spectra can have very different time domain representations. Consider the signal $x_1 = 3cos(\frac{2\pi t}{N}) + sin(\frac{11\pi t}{N})$ and $x_2 = 3sin(\frac{2\pi t}{N}) + cos(\frac{11\pi t}{N})$. Clearly these two signal have the same frequency content of 2π radians per second, and 11π radians per second. If we take the Discrete Fourier Transform with $N = 256$ points, we obtain the following plots.

Although x_1 and x_2 have identical magnitude spectra, the time signals are very different. The phase information is clearly crucial for reconstructing the correct time domain signal. Such an observation is of crucial importance as we perform frequency domain manipulation, and tells us that "correct" phase values give us "correct" time signals. While cosine and sine have phase values and offsets of their own, phase here refers to the phase of a complex sinusoid and is found by plotting the sine component vs the cosine component and looking at the angle the resultant vector makes with the positive cosine/real axis. Thus our real and imaginary or cosine and sine components are always at zero phase.

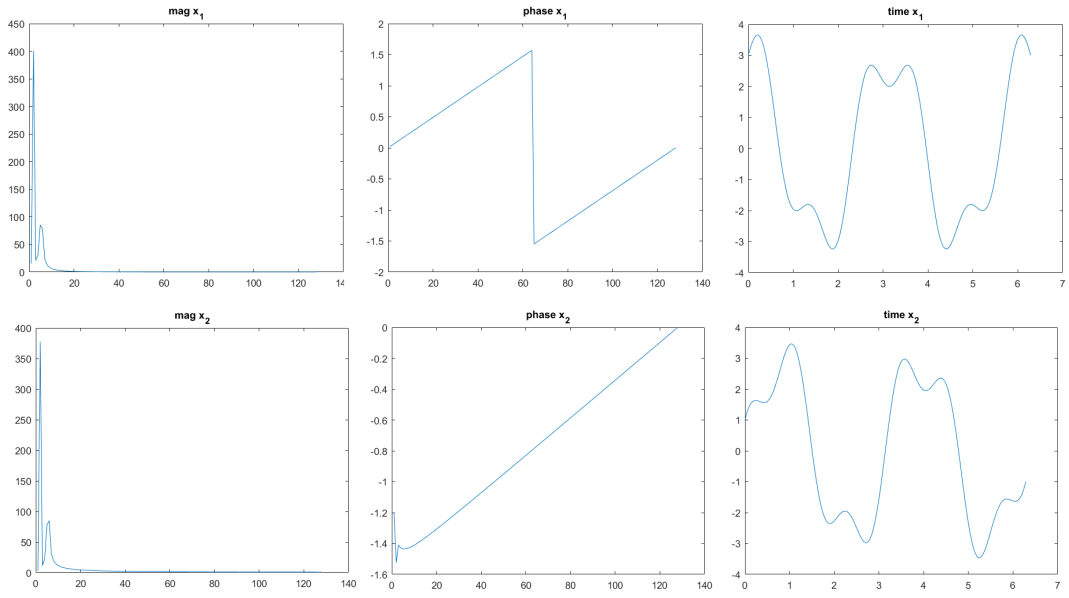


Figure 2.2.2. Magnitude and Phase spectra for x_1 and x_2, as well as their time domain representation.

2.3 Fourier Transform as a Change of Basis

Let us look again look at our equation for the Discrete Fourier Transform

$$x[t_n] = \frac{1}{N}\sum_{n=0}^{N-1} X[\omega_k]e^{i\omega_k t_n}$$

Our time domain signal is identical to the sum of N sinusoids. Therefore we can imagine our signal as existing in an N dimensional function space, where each function e^{ω_k} represents one dimensional in the space. What about if we want to know the amplitude of one particular frequency, in other words the contribution of 1 of the N dimensions to the whole signal? We want to pick out a particular term of a finite sum. What is a function that is zero everywhere except at one point/sample where it is 1?

2.3.1 The Delta Function

We use *cos* and *sin* for a vector in $\mathbb{R}^2$ to find out out much the x and y components contribute to the whole vector, respectively. Similarly we use the delta function in a Fourier Transform to find how much the ω_k^{th} component contributes to our time signal. Let us first look at different forms the delta function can take. Consider the following,

$$\frac{1}{N}\sum_{n=0}^{N-1} e^{i\frac{k-m}{N}2\pi n}$$

This is the sum of a complex sinusoid at frequency $\frac{k-m}{N}$ Hz, divided by the number of points which we are adding. This summation represents the average value of the function over one full period. For any $(k-m) \neq 0$, the average is equal to zero, since the average of a sine and cosine wave over one full period is zero. However, when $(n-m) = 0$, we see that

$$\frac{1}{N}\sum_{n=0}^{N-1} e^{i\frac{0\cdot 2\pi}{N}n} = \frac{1}{N}\sum_{n=0}^{N-1} 1 = \frac{1}{N}(1+1+...+1+1) = \frac{1}{N}(N) = 1$$

Thus we see that the summation $\frac{1}{N}\sum_{n=0}^{N-1} e^{i\frac{k-m}{N}2\pi n}$ is the same as a delta function centered at $k = m$, denoted $\delta_{(k,m)}$.

Now let's look again at our formulas for the DFT and IDFT

$$x[t_n] = \sum_{k=0}^{N-1} X[\omega_k]e^{-i\omega_k t_n},\ X[\omega_k] = \frac{1}{N}\sum_{m=0}^{N-1} x[t_m]e^{i\omega_k t_m}$$

Let's look at what happens when we replace $X[\omega_k]$ in our expression for $x[t_n]$ with its finite summation.

$$x[t_n] = \sum_{k=0}^{N-1}\frac{1}{N}\sum_{m=0}^{N-1} x[t_m]e^{i\omega_k t_m}e^{-i\omega_k t_n} = \sum_{k=0}^{N-1}\frac{1}{N}\sum_{m=0}^{N-1} x[t_m]e^{i\omega_k (t_m - t_n)}$$

Recall that $t_n = \frac{n}{f_s}$ for $n = 0, 1, 2, ..., N-1$ and $\omega_k = \frac{2\pi k}{T}$ for $T = \frac{N}{f_s}$, so we see that

$$= \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} x[t_m]e^{i\omega_k (t_m - t_n)} = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} x[t_m]e^{i\frac{2\pi k f_s}{N}(\frac{m-n}{f_s})} = \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} x[t_m]e^{i\frac{k}{N}2\pi(m-n)}$$

Since the two summations have the same endpoints, by the properties of finite sums, we know the two summations as well as the scalar $\frac{1}{N}$ can be rearranged such that

$$= \sum_{m=0}^{N-1} x[t_m]\frac{1}{N}\sum_{k=0}^{N-1} e^{i\frac{k}{N}2\pi(m-n)}$$

It is important here to note the crucial step in any form of the Fourier Transform that just took place: *the summands are switched!* We went from first summing through the time indices to first summing through the frequency indices. Furthermore, note the innermost summation in

the previous equation: it is exactly the form of the delta function which we introduced earlier! Finally we see that,

$$= \sum_{m=0}^{N-1} x[t_m]\delta_{(m,n)} = x[t_n]$$

Which is what we started out with! As we see, the DFT is a transform, as we have just went from the time domain, to the frequency domain, and then again returned our original sample from the time domain.

So if we want to know the m^{th} Fourier coefficient, we simply multiply our DFT by $\frac{1}{N}\sum_{n=0}^{N-1} e^{i\frac{-m2\pi}{N}n}$, where it follows that for

$$X[\omega_m] = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{N-1} X[\omega_k]e^{i\omega_k t_n}e^{-i\omega_m t_n} = \frac{1}{N}\sum_{n=0}^{N-1} X[\omega_k]\sum_{k=0}^{N-1} e^{i\frac{2\pi(k-m)}{N}n}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} X[\omega_k]\delta_{k,m} = X[\omega_m]$$

Thus we see how the delta function isolates a component of the N dimensional frequency domain representation of our signal. In this way multiplying by the spectrum by the summation $\sum_{n=0}^{N-1} e^{i\frac{2\pi(-m)}{N}n}$ tells us how much the m^{th} frequency component $X[\omega_m]$ is contributing to the signal's total energy.

2.4 The Fourier Transform in Practice

2.4.1 The Fast Fourier Transform

Up until now we have been thinking about the Fourier Transform and its discrete form the DFT. However, most (if not all) cases of using the Fourier Transform in practice use the Fast Fourier Transform (FFT). The Fast Fourier Transform produces the same data as the DFT, except at a much faster speed. This is due to a few tricks that can be done with discrete data which we will see later. The inverse direction is called the Inverse Fast Fourier Transform (IFFT). Because the FFT allows us to process data into frequency domain data much quicker, there are a few new techniques that can be used to improve the performance of frequency domain processing. We will see these in the following sections.

2.4.2 Windowing

When we use the Fast Fourier Transform, *windowing* is applied to both the input and output of the FFT. When a signal is windowed, it is multiplied by another signal. Windowing is a form of amplitude modulation and serves two purposes. First of all it makes the signal look *more* periodic. Secondly, windowing allows us to *overlap-add* multiple instances of the FFT running at once. We don't want to window a signal by any other arbitrary signal; there are certain windowing properties that, when met, make the FFT work more like we want it to. Common windowing signals include the *Hanning, Hamming*, and *Blackman* windows. The following are formulas for calculating a window of length N.

$$\text{Hanning Window} = .5[1 - cos(\frac{2\pi n}{N-1})] = sin^2(\frac{\pi n}{N-1})$$

$$\text{Hamming Window} = .54 - .46cos\Big(\frac{2\pi n}{N-1}\Big)$$

$$\text{Blackman Window} = .42 - .5cos\Big(\frac{2\pi n}{N-1}\Big) + 0.08cos\Big(\frac{4\pi n}{N-1}\Big), \quad 0 \leq n \leq N$$

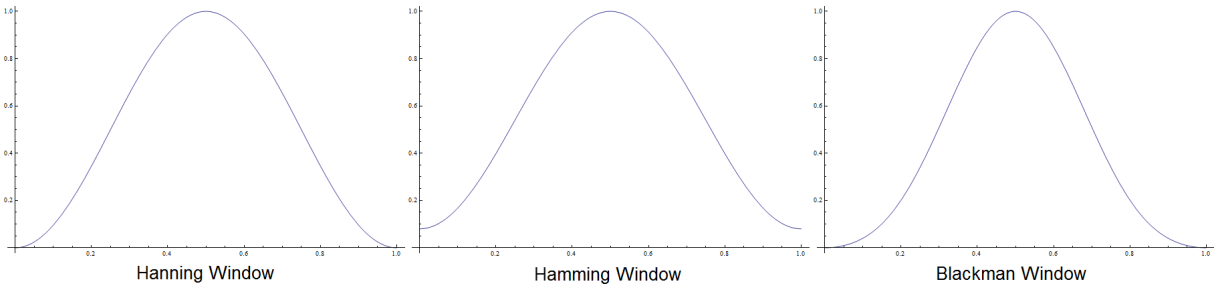


Figure 2.4.1. Common windowing functions

2.4.3 Periodicity

The Fourier Transform is operating under the assumption that whatever signal it is analyzing is periodic, in other words, that the endpoints of the signal are "connected". The Fourier Transform "thinks" a signal is periodic because it is comparing it to periodic signals! Namely $e^{i\omega t}$. This is a pretty big problem for most real-world, real-time applications, since not many signals are periodic. Windowing is one way to make a signal look more periodic. All of the above mentioned windows have the property that the end points approach zero. Because of this, our signal begins

to look more periodic, since both ends are approaching zero.

Why are jumps at the endpoints important? Because the Fourier Transform will assume that they are meaningful, and try to account for them in the spectrum with high frequencies. The effect of time domain sharp transitions on the signal's Fourier Transform is common. The most common, and simplest, example is known as the *Gibbs Effect*: the result of modeling a square wave as a sum of sinusoids. As the Fourier Transform tries to reconstruct the edges of the signal (where the signal changes from 1 to 0), higher and higher frequencies are added. However, the Fourier reconstructed signal constantly overshoots the original value of the time signal at these sharp transitions. As we will see, as more frequencies are added, the width of this overshoot narrows, however the height remains the same at about 9% of the amplitude of the square wave [3].

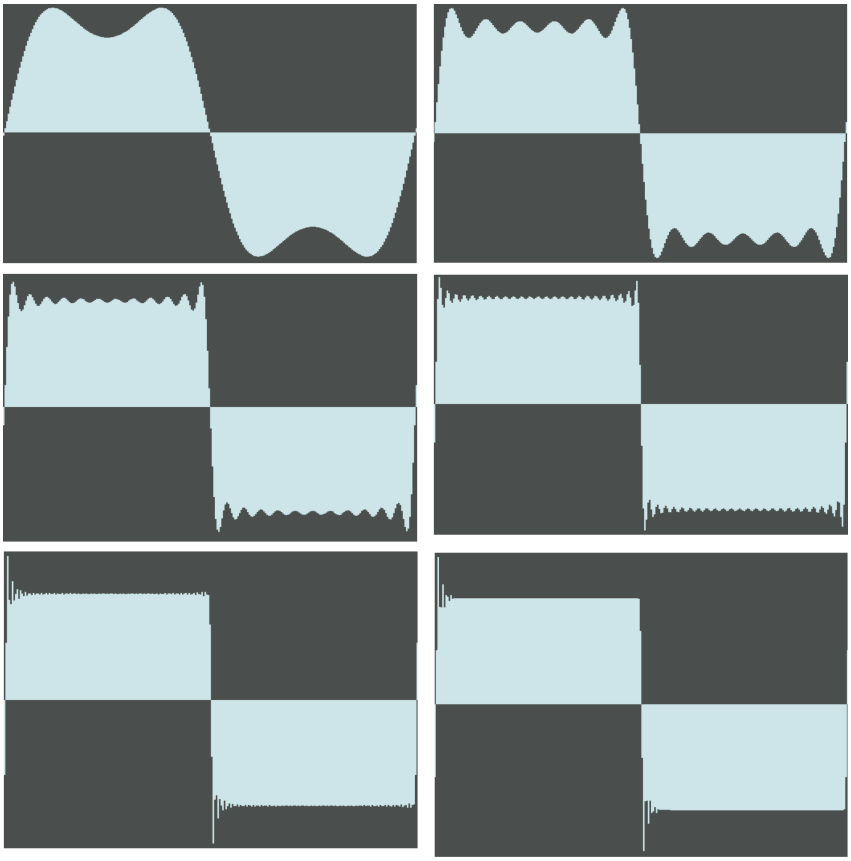


Figure 2.4.2. The Gibbs effect as more sinusoids are used to reconstruct a square wave

Windowing reduces sharp transitions between the beginning and end of an analysis signal, thus reducing the the amount of frequencies the FFT thinks are in the signal, and would add to try and account for large, fast changes in amplitude. We can see the effect windowing has on the signal from figure 2.0.5, where we compared at a signal which has a similar harmonic series to the FFT to a signal that doesn't.

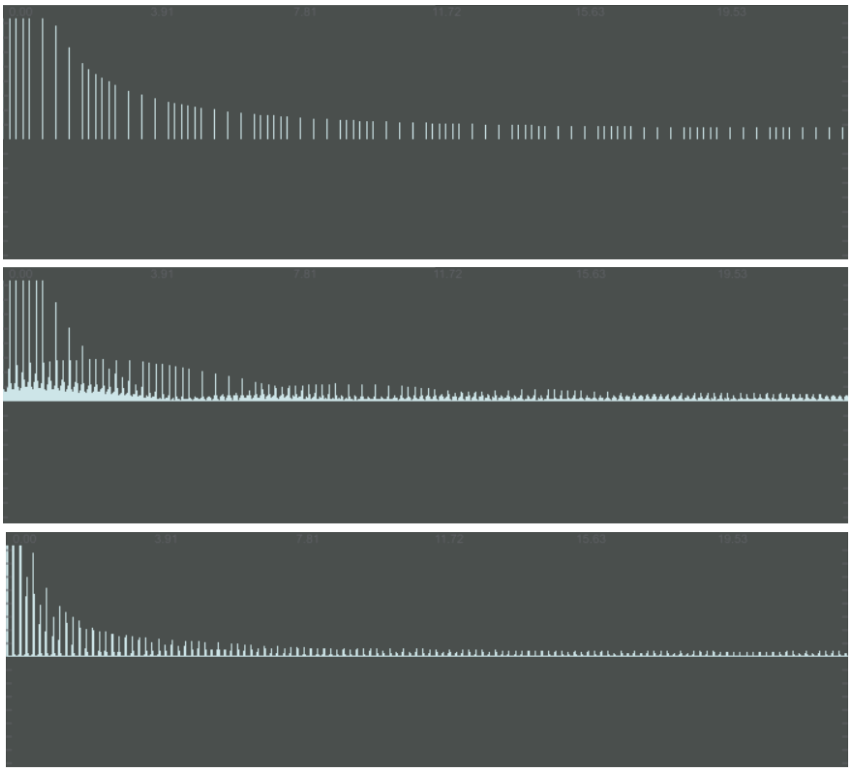


Figure 2.4.3. Same figure as in figure 2.3.2 but the *bottom* plot shows the spectrum of the windowed 173 Hz signal.

As we can see, a lot of the noise has been removed from the spectrum as a result of applying a Hanning window to the 173 Hz signal. This is because the input signal now looks much more periodic to the FFT. However, the plot is still not nearly as clean as the unwindowed signal whose frequency matches a multiple of the FFT's fundamental frequency, which is naturally periodic with respect to the FFT.

However, by making our input signal look more periodic, we have added a new frequency to the spectrum that is not in the original signal: the windowing function's frequency, which is

also the fundamental frequency of the FFT. However, because of a nice property of windowing functions, we will see that these additional frequencies will be gone when the constructed time domain signal reaches our ears.

2.4.4 Overlap Add Synthesis

Windows that satisfy the *constant-overlap-add (COLA)* property allow us to join multiple instances of the IFFT when synthesizing a new time domain signal[2]. Overlap adding has a few advantages. Firstly, our windowing frequency that was added to the input signal to the FFT is now "undone". Secondly, since we are windowing our output IFFT, and the ends of the window approach zero, we hear the transition between synthesis frame u and synthesis frame $u+1$ much less than if the output were unwindowed, thus making the output sound more like a continuous waveform. Each of these IFFT instances, and consequently FFT instances, is offset from the others by a *hop factor*, R, usually between 2 and 8. It is often said that each FFT instance is $\frac{360}{R}$ degrees out of phase from the last instance. The COLA property states that for a window w, it must be the case that

$$\sum_{m=0}^{R-1} w(n - m\frac{N}{R}) = 1, \text{ for } 0 \leq n \leq N-1$$

in order to overlap-add R FFT/IFFT instances. An obvious first window that comes to mind is the triangular window, described by the function

$$w[n] = 1 - \frac{2}{N} \cdot \Big|\frac{N}{2} - n\Big|$$

Let's show that this window satisfies the COLA property for a hop factor of 2. Consider $w[n] + w[n + \frac{N}{2}]$,

$$= \Big(1 - \frac{2}{N}\Big|\frac{N}{2} - n\Big|\Big) + \Big(1 - \frac{2}{N}\Big|\frac{N}{2} - (n + \frac{N}{2})\Big|\Big)$$

let's assume that $n < \frac{N}{2}$. It follows that

$$= \Big(1 - \frac{2}{N}(\frac{N}{2} - n)\Big) + \Big(1 - \frac{2}{N}\Big|\frac{N}{2} - n - \frac{N}{2})\Big|\Big) = 1 - 1 + \frac{2n}{N} + \Big(1 - \frac{2}{N}(n)\Big)$$

$$= \frac{2n}{N} + 1 - \frac{2n}{N} = 1$$

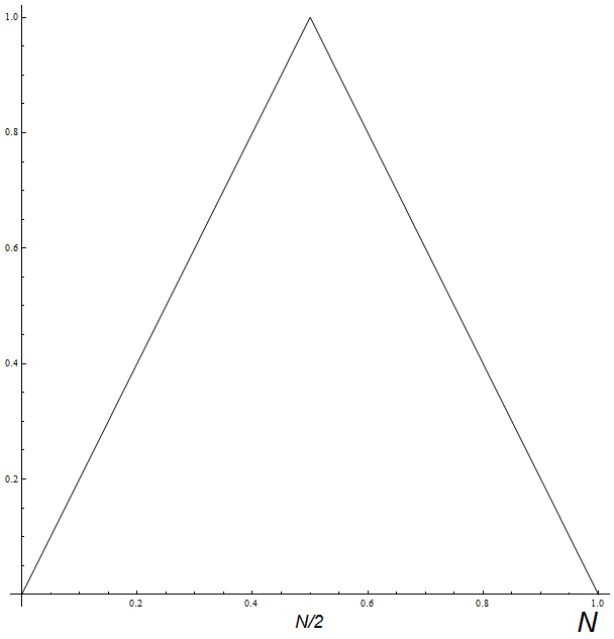


Figure 2.4.4. Triangular Window

We see that the same argument is also true for $n > \frac{N}{2}$, since $w[n + \frac{N}{2}] = w[n - \frac{N}{2}]$ by the fact that for point $w[m] = w[m + N]$, since w is length N, and the distance of $n + \frac{N}{2}$ to $n - \frac{N}{2}$ is $(n + \frac{N}{2}) - (n - \frac{N}{2}) = n + \frac{N}{2} - n + \frac{N}{2} = N$. Finally, since addition is commutative, we see that

$$w[n] + w[n + \frac{N}{2}] = w[n + \frac{N}{2}] + w[n] = w[n - \frac{N}{2}] + w[n] = w[n^*] + w[n^* + \frac{N}{2}]$$

for $n^* = n - \frac{N}{2}$, which we already showed was true above. So we have shown for a hop factor $R = 2$, the triangular window satisfies the COLA property. However, what about for a hop factor $R > 2$? The same proof applies for any even hop factor $R = 2q$, by grouping the $2q$ windows, into q pairs of windows. For a hop factor $R = 2q$, where $q \geq 2$, we add the additional step of including a factor of $\frac{1}{q}$, since each pair of windows will have a gain of 1, and we are summing q of them. So if we want to maintain unity gain, the total sum must be scaled by $\frac{1}{q}$. Figure 2.4.6 shows 4 overlapping triangular windows as the composition of 2 pairs of windows.

It can be shown that the Hanning window also satisfies this *COLA* property nicely. It follows that

$$h[n] + h[n + (N-1)/2] = .5[1 - cos(\frac{2\pi n}{N-1})] + .5[1 - cos(\frac{2\pi(n + (N-1)/2)}{N-1})]$$

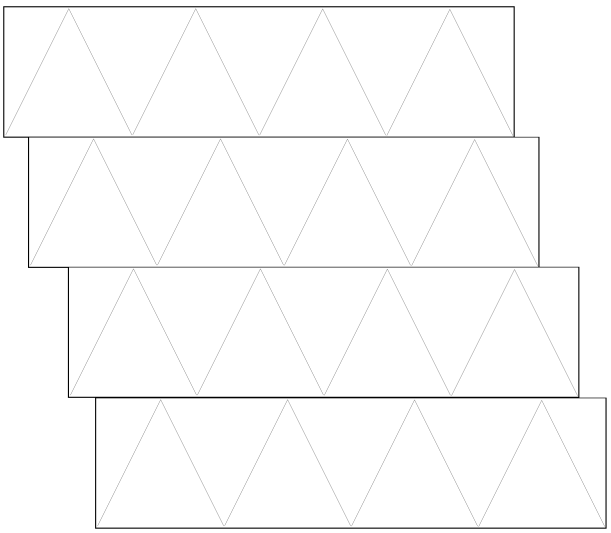


Figure 2.4.5. Traditional Triangle Window Overlap Add diagram, Hop Factor of 4, each window 90 degrees out of phase from the last.

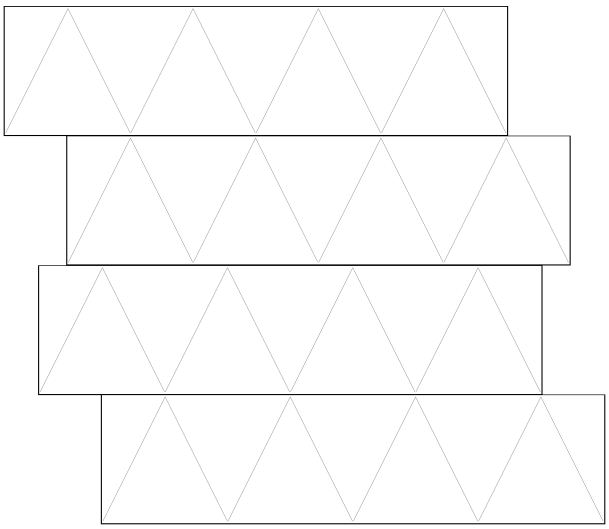


Figure 2.4.6. Hanning Triangle Overlap Add diagram reordered to show the diagram as a composition of pairs of windows, each window 180 degrees out of phase with its pair.

$$= .5[1 - cos(\frac{2\pi n}{N-1})] + .5[1 - cos(\frac{2\pi n}{N-1} + \frac{\pi(N-1)}{N-1})]$$

$$= .5[1 - cos(\frac{2\pi n}{N-1})] + .5[1 + cos(\frac{2\pi n}{N-1})]$$

$$= .5 - .5 \cdot cos(\frac{2\pi n}{N-1}) + .5 + .5 \cdot cos(\frac{2\pi n}{N-1}) = 1$$

Once again, we use a hop factor $R = 2$. The same argument as above applies for even hop factors $R = 2q$, including the scalar $\frac{1}{q}$.

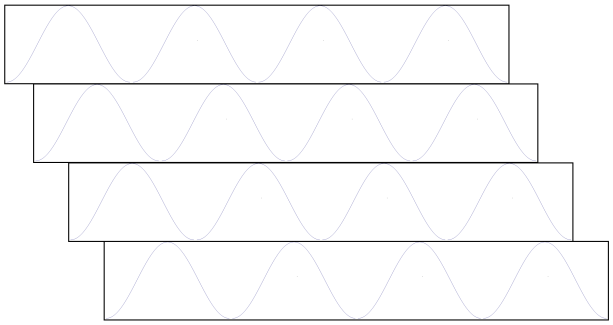


Figure 2.4.7. Traditional Hanning Window Overlap Add diagram, Hop Factor of 4, each window 90 degrees out of phase from the last.

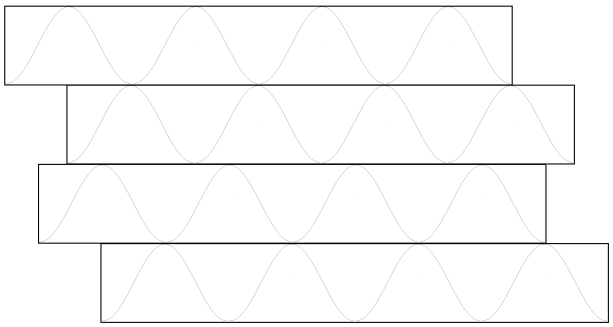


Figure 2.4.8. Hanning Window Overlap Add diagram reordered to show the diagram as a composition of pairs of windows, each window 180 degrees out of phase with its pair.

Both the FFT input $x[t]$ and the IFFT output $y[t]$ are windowed with signals that satisfy the COLA property. The first problem of windowing, the addition of windowing frequencies to the signal, is taken care of simply by unwindowed overlap-add synthesis, because the COLA property is satisfied for the input window, which is adding frequencies to the signal in the first place. In order to show this, it follows that by windowing $x[n]$ by $w[n]$ with $R = 2$ and frame size N, and no spectral modification between analysis and resynthesis, in other words $\mathcal{F}^{-1}(\mathcal{F}[x[n] * w[n]]) = x[n] * w[n]$, that every group of $\frac{N}{2}$ samples from $x[t]$ in FFT instance 1 will be overlapped with the same $\frac{N}{2}$ samples from $x[t]$ in FFT instance 2, except windowed 180 degrees out of phase from each other, such that

$$\mathcal{F}^{-1}(\mathcal{F}[x[n] * w[n]]) = \sum_{n=0}^{\frac{N}{2}} \Big[w[n] * x[n] \Big] + \sum_{n=0}^{\frac{N}{2}} \Big[w[n + \frac{N}{2}] * x[n] \Big]$$

$$= \sum_{n=0}^{\frac{N}{2}} \Big[w[n] * x[n] + w[n + \frac{N}{2}] * x[n] \Big] = \sum_{n=0}^{\frac{N}{2}} \Big[\Big(w[n] + w[n + \frac{N}{2}] \Big) x[n] \Big]$$

$$= \sum_{n=0}^{\frac{N}{2}} 1 \cdot x[n] = x[n]$$

Thus the COLA property "undoes" the additional frequencies added to the signal by the window, simply by overlap adding. However, we window the output signal $y[t]$ anyway because it removes audible discontinuities between adjacent IFFT synthesis frames. And because (let's stick with a simple hop factor of $R = 2$), once again, half of the data each FFT is analyzing is also being analyzed by the other FFT, the COLA property ensures that we there is no change in amplitude for these samples because of the windowing or overlap-add process, and the resultant signal sounds like a continuous waveform.

2.5 How the Fourier Transform is Calculated

Here we will look at a Java implementation of the Discrete Fourier Transform. In the following sections, we will highlight and go through certain sections of the code which embody the concepts we have been working with.

2.5.1 DFT Calculation

The DFT works by comparing the input signal to a set of sines and cosines. Here, by comparing we mean multiplying the input signal by a sine wave and a cosine wave at frequency $\omega_k = \frac{2\pi k}{T}$, for $0 \leq k < \frac{N}{2}$, sample by sample, and summing the resultant products. The value of the sum will tell us how much cosine and sine are in the input signal. The larger the sum, the more similar the input signal is to the sine or cosine at that frequency ω_k. In the following example, we will look at one frame of a simple signal which we create. This is not the type of Fourier Transform one would use in real time, however, it is a simple version which is easy to understand.

First we start by creating our signal. In this example, our signal will be

$$x[t] = -cos(\frac{16\pi t}{256}) + sin(\frac{23\pi t}{256})$$

t is between 0 and 256, since $x[t]$ is sampled 256 times, and frequency is in units *radians per second*. We will only look at the first half of the spectrum, which is the only information useful

to us now, since the second half is a mirrored copy of the first. We see the time domain representation in the following figure

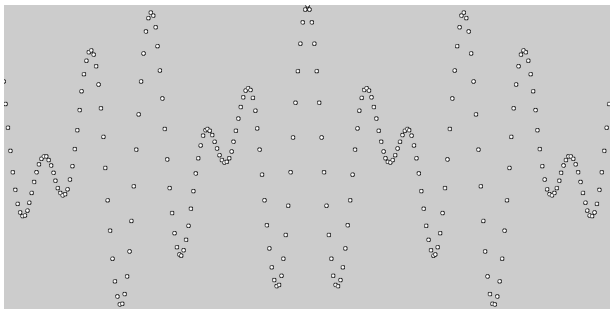


Figure 2.5.1. $x[t] = -cos(\frac{16\pi t}{256}) + sin(\frac{23\pi t}{256})$

To calculate the spectrum of the above signal, we must first create two arrays, one for the real or cosine component, and another for the imaginary or sine component. Each of these arrays has length $\frac{N}{2}$. Why $\frac{N}{2}$ and not N? Because we are taking N points and transforming them into another N points, and we are storing these points in 2 objects, it makes sense that each object have size $\frac{N}{2}$, so that in total there are still N data points. Next we create two FOR loops, one nested inside of the other. The outer FOR loop counts through the frequency indices i, from $0 - \frac{N}{2}$, and the inner FOR loop counts through the time domain samples j from $0 - N$, at the frequency specified by the outer FOR loop, and multiplies the signal by these sinusoids. The real array accumulates the product of the signal times a cosine wave at frequency i, and the imaginary array accumulates the negative of the product of the signal times a sine wave at frequency i.

```
//counts through each frequency domain index
for (int i = 0; i < num/2+1; i++) {
  //counts through each time domain sample
  for (int j = 0; j < num; j++) {
    real[i] = real[i]+samples[j]*cos(2*PI*i*j/num);
    imag[i] = imag[i]-samples[j]*sin(2*PI*i*j/num);
  }
}
```

Figure 2.5.2. Java nested FOR loop in Processing 3 to calculate the real and imaginary spectral arrays

Next, in order to visualize the amplitude spectrum, we must go to the world of the 3D complex sinusoids, where instead of data for scaled sines and cosines at different frequencies, we

have data on the amplitude and phase of 3D sinusoids at different frequencies. We obtain the amplitude spectrum simply by the Pythagorean Theorem and obtain the following plot.

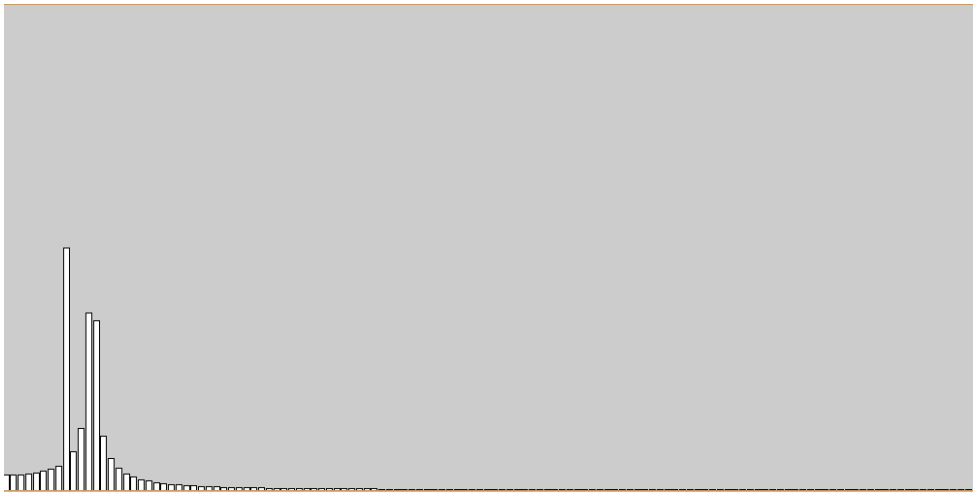


Figure 2.5.3. Amplitude spectrum for $x[t] = -cos(\frac{16\pi t}{256}) + sin(\frac{23\pi t}{256})$. Clearly we see a peak at 8Hz, 11Hz and 12Hz.

So we have 128 spectral bins. Earlier we stated that the center frequency ω_k of bin k was $\frac{kf_s}{N}$, where f_s was the sampling frequency and N was the frame size. Here $N = 256$. Usually the sampling rate for audio is $f_s = 44100$ samples per second. In general our sampling rate is the number of samples we record divided by the amount of time it takes to record that many samples. Here we are sampling the signal 256 time over a time interval of T. However, here we see that the rate at which the signal is created and analyzed does not have any effect on the result! Whether we add a sample every 100 millisecond or every 2 seconds, does not change anything because the signal is already prepared. All that matters is there are 256 samples to represent the signal. So, T in this example is 1, and $f_s = \frac{N}{T} = \frac{256}{1}$. Therefore the center frequency of ω_k is $\frac{k \cdot 256}{256} = k$. The frequency width of each bin is $(k)\ Hz - (k-1)\ Hz = 1Hz$.

2.5.2 IDFT Calculation

The Inverse Discrete Fourier Transform is strikingly similar to the Forward Discrete Fourier Transform. This is one of the beauties of the Fourier Transform! We saw this earlier in the formulas for the forward and inverse Fourier Transform, and the same is true for discrete computer

calculation. In this direction, we start with two arrays of length $\frac{N}{2}$, and want to synthesize a time domain signal of length N from the array data. This direction of the Fourier Transform is perhaps easier to conceptualize since we are "constructing a time domain signal as a sum of sinusoids", which is often how Fourier Theory is informally described. First we create an array to store the synthesized signal. Then we use the same structure of two FOR loops, one nested inside of the other. Again, the outer loop counts through the frequency indices $0 < i < 128$, while the inner loop counts through the time indices $0 < N < 256$. The sine or cosine wave chosen by the outer loop is scaled by the corresponding real or imaginary Fourier Coefficient stored in the array, and the array to store the synthesized signal accumulates the sum of these terms. Lastly, each sine and cosine that we are summing is also scaled by the number of them that there are, namely $\frac{N}{2}$. Here we see the Java code for the aforementioned process. And finally

```
//reconstruct time domain signal
for(int i = 0; i < num/2. +1; i++)
{
  for(int j = 0; j< num; j++)
  {
    sig[j] += (real[i]*cos(2*PI*i*j/num) -imag[i]*sin(2*PI*i*j/num))/num;
  }
}
```

Figure 2.5.4. Time domain signal accumulates the scaled sine and cosines to create the synthesized signal.

let's compare our synthesized signal with the original signal

2.5.3 Spectral Leakage

The cause of trouble when using the Discrete Fourier Transform, and algorithms that use the Fourier Transform such as the Phase Vocoder, is *Spectral Leakage*. Spectral Leakage is the effect from having discrete frequency bins, each which correspond to a range of frequencies, rather than a particular one. We have already seen this in the previous sections, such as in figure 2.4.3, where signals less periodic with respect to the FFT exhibit more spectral leakage than signals that are more periodic with respect to the FFT. We also saw how windowing can help reduce spectral leakage. As we will see in the following chapter, spectral leakage will fill bins that are in reality empty, thus distorting the signal when we synthesize it back into the time domain. Improvements to the Phase Vocoder acknowledge this fact and try to minimize the distortion. In order to build up to such algorithms, we will start with a simple example to show the problem

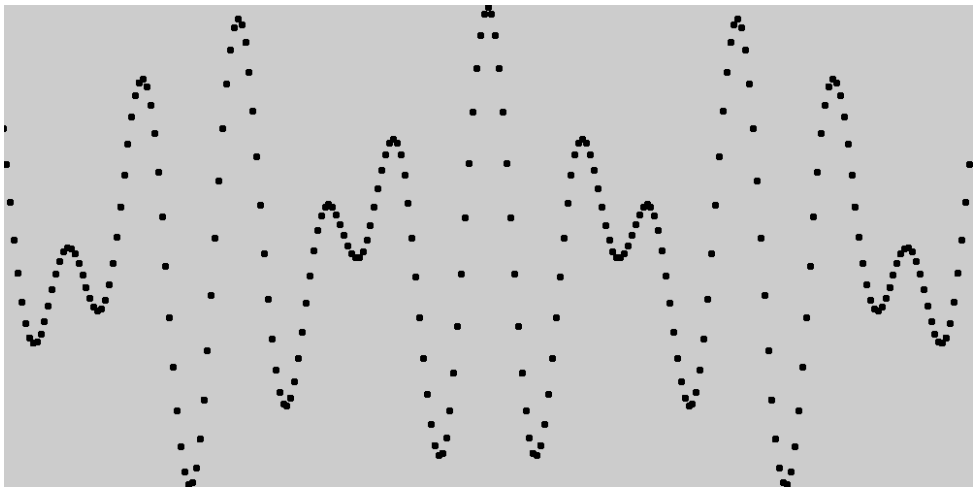


Figure 2.5.5. Synthesized signal from the DFT spectral data

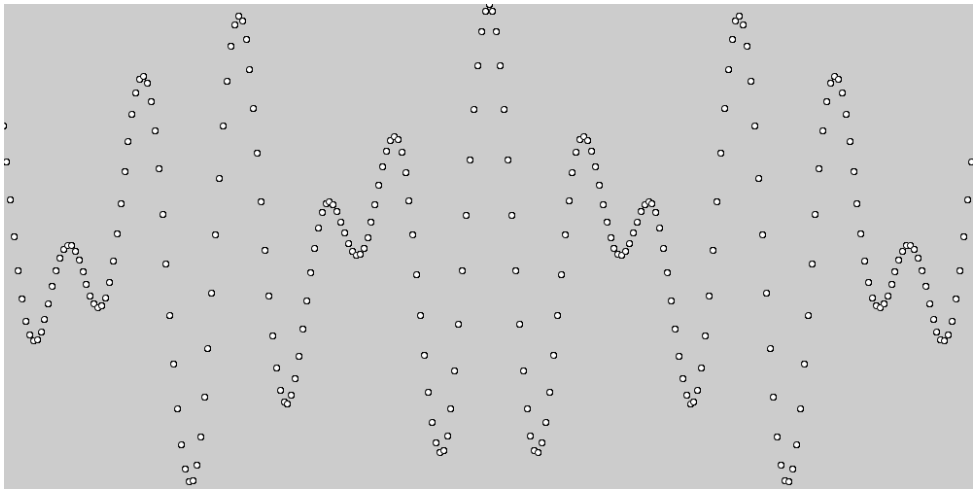


Figure 2.5.6. Original Time Domain signal

at hand, and to motivate understanding for how algorithms in the following sections are working.

Consider a 256 sample, discrete time signal of the function $f_1[t] = cos(\frac{2\pi * t}{T})$.
By using the aforementioned method of calculating a signals frequency spectrum, we obtain the following amplitude spectrum Clearly, because $f_1[t]$ is exactly one of the basis functions that the DFT is comparing it to, the amplitude spectrum of $f_1[t]$ looks nice.

Now consider a second signal, $f_2[t] = cos(\frac{3\pi t}{T})$ and see the time domain and frequency domain representation of $f_2[t]$. Clearly there is only one frequency present in the $f_2[t]$, however, due to spectral leakage, one would think from looking at the amplitude spectrum that many frequencies

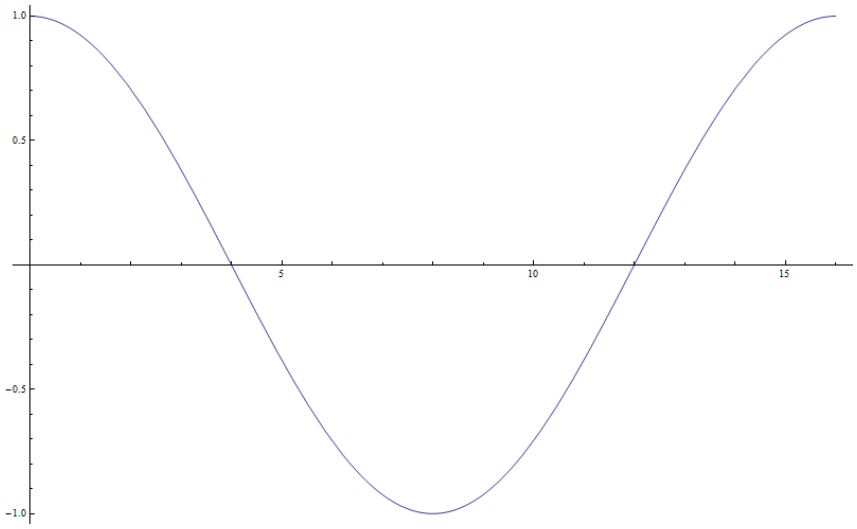


Figure 2.5.7. $f_1[t] = cos(\frac{2\pi t}{T})$

Figure 2.5.8. 8 point amplitude spectrum of $f_1[t]$, with a peak at bin 2.

are present. Here because $f_2[t]$ does not resemble exactly any of the basis functions the DFT is comparing it to, we see that $f_2[t]$ ends up kind of looking like all of them.

Now this isn't always bad. Let us see if the reconstruction of $f_2[t]$ resembles the original signal at all. Here we will overlay the original signal as black points, overtop of the resynthesized signal as white circles.

So the Discrete Fourier Transform is good at performing exact reconstruction of a signal, even if the frequency data is inaccurate, as long as no modification is made in the frequency domain.

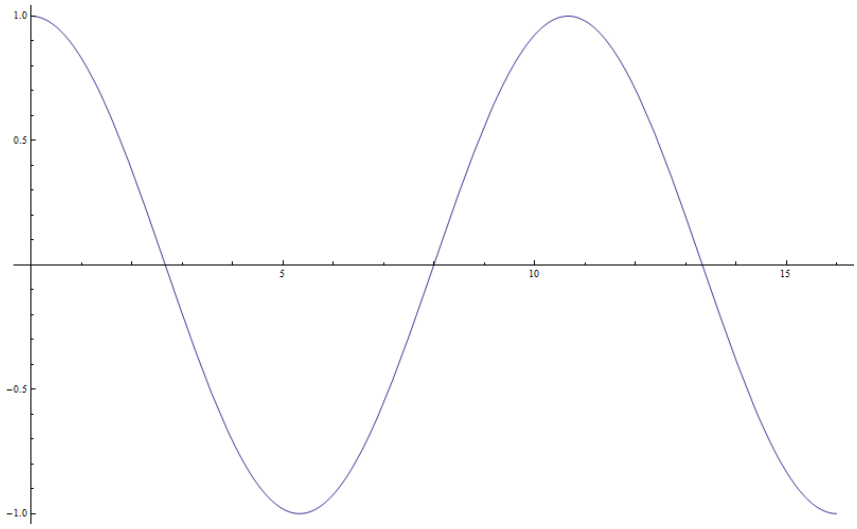


Figure 2.5.9. $f_2[t] = cos(\frac{3\pi t}{T})$

Figure 2.5.10. 8 point amplitude spectrum of $f_2[t]$, with spectral leakage throughout the spectrum.

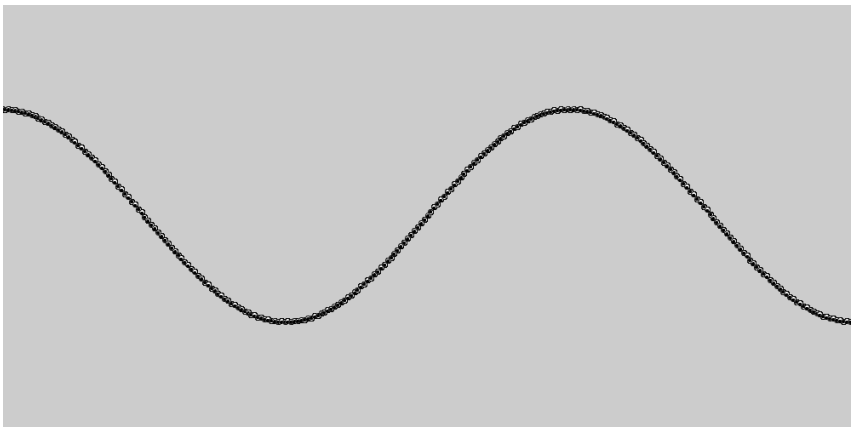


Figure 2.5.11. The two signals agree nicely, even though the spectrum was erroneously calculated.

However, as we will see, if we start to change the frequency domain data, the reconstructed signal will not always turn out how we want it.

3
The Phase Vocoder

Most time domain methods used to modify the duration of a signal also modify the component frequencies of the signal. Think about playing a record at various speeds. When it is played slower, all of the pitches are scaled down relative to how much you are slowing it down. Similarly, when played faster, all of the pitches are scaled up relative to the speed change (namely twice as fast or half as slow corresponds to uniform frequency shift of up or down an octave, respectively.) This is known as time and frequency dependence. In this chapter, we will look at the Phase Vocoder: a frequency domain algorithm which allows us to modify a signal's time and frequency independently of each other. The Phase Vocoder was introduced in 1966 by James Flanagan and R. M. Golden. Since then, many DSP engineers have put their own spin on the algorithm in order to try and achieve better time and frequency independence. Here we will look at the basic idea of the algorithm, as well as a simple modification made to the PV to try and instill vertical phase coherence, a problem that arises as a result of spectral leakage and sinusoids moving through the spectrum. Then later we will look at more advanced, and complicated, approaches to achieving better vertical phase coherence.

Let's start with a few simple observations. In 2.2 we looked the Fourier Transform of a square wave. From subsets of the frequency domain data we synthesized two new signals, one signal

retained amplitude information, while the other retained phase information. We then noticed that the timing of events was preserved in the signal which retained the phase information of the square wave. Thus concluding that the phase information in some way encodes the timing of events in a signal. So when we think about what we are trying to achieve, time/frequency independence, it seems natural that the phase information of a signal's frequency domain representation should be at the center of it. Which in fact it is.

We stated earlier that the FFT usually processes samples in chronological order, in other words, after samples 0 through $N-1$ have been transformed into frequency domain data, samples N through $2N-1$ are the next samples processed. However, this does not always have to be true. In fact we can process frames of time domain samples in any particular order, as long as each frame represents a section of the original signal. Thus we see that the next frame analyzed is not necessarily the next frame in time! This fact is also at the core of the Phase Vocoder's theory of operation.

3.1 A Simple Example

Before we get into the math of the Phase Vocoder, let's first look at a simple example, and then follow up with an informal description of what the algorithm is doing, in order to gain some intuition on why the Phase Vocoder might do what it does. To start, remember that one frame of FFT data has the correct short term amplitude and phase information. And since we don't have to read through a sound sample in order, one might ask why we cannot just read FFT frames out of order and reconstruct a time stretched/contracted signal from that data. The frequency information for each N point frame would be correct, and because of the COLA property, we wouldn't hear the edges between IFFT frames. So it is conceivable that by simply reconstructing FFT frames out of chronological order, time/frequency independence could be achieved. Let's see what would result if such an approach were taken to a simple sine wave at the FFT's fundamental frequency, ω_1.

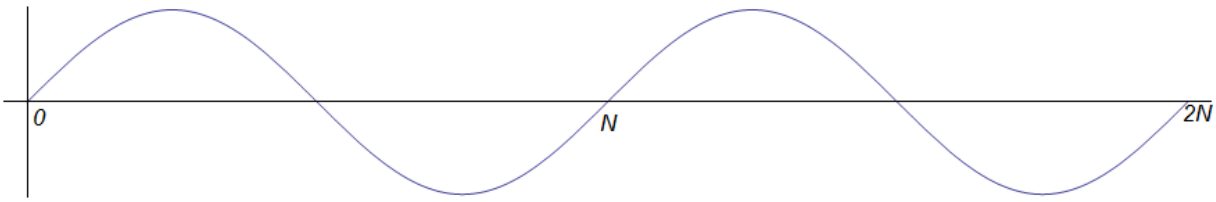


Figure 3.1.1. Simple time domain signal $sin(\omega_1)$.

In the following figure, we have a hop factor of 2 and no spectral modification is made, so the FFT input and IFFT output. We are trying to achieve a time stretch of 2 on a signal that is $2N$ samples long thus generating a signal that is $4N$ samples long, which has the same frequency as content as the $2N$ signal. In order to achieve this time stretch we read frames of the signal out of order. For a time stretch of 2, we must read through the signal half as fast. The figure below illustrates this process, and the following table shows the sample indices through which each frame reads.

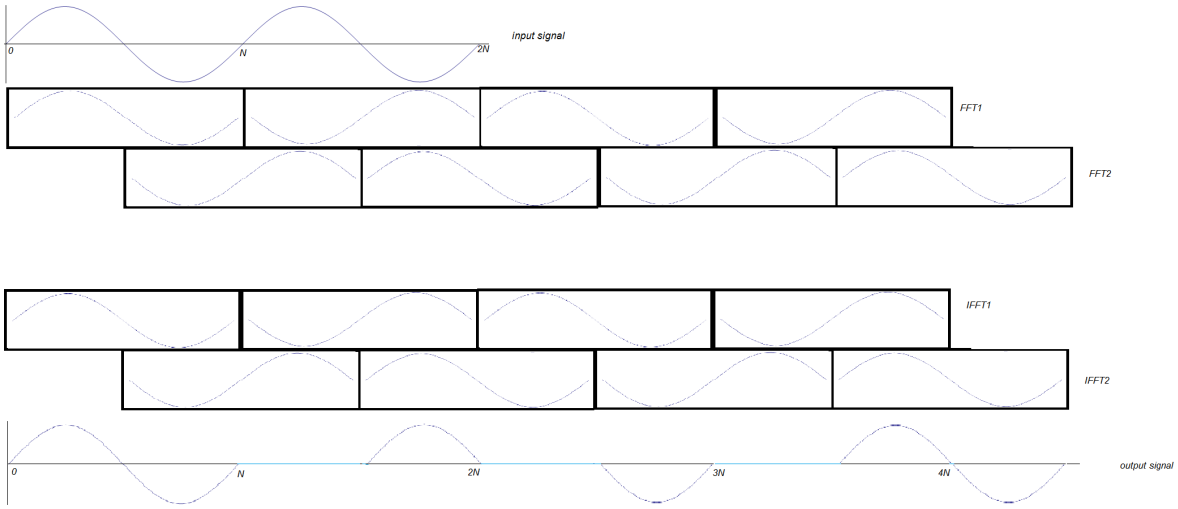


Figure 3.1.2. Time stretching by reading FFT frames out of order with no spectral modification.

frame	FFT1 samples	FFT2 samples
1	$0 \rightarrow N-1$	
2		$(\frac{N}{2}-1) \rightarrow (\frac{3N}{2}-1)$
3	$(\frac{N}{2}) \rightarrow (\frac{3N}{2}-1)$	
4		$N \rightarrow 2N-1$
5	$N \rightarrow 2N-1$	
6		$(\frac{3N}{2}) \rightarrow (\frac{5N}{2}-1)$
7	$(\frac{3N}{2}) \rightarrow (\frac{5N}{2}-1)$	
8		$2N \rightarrow 3N-1$

As we can see from the output signal at the bottom of the figure, time modification independent of frequency did not result from simply reading the frames out of order. So just the fact that we can read frames out of order does not guarantee time/frequency independence. The output signal above resembles the input signal for half frame sections, so it retains some frequency relationship to the input. However, its phase of the output sine wave is clearly not continuous. The result is a chopped up and respaced version of the input. Hardly the length $4N$ signal of frequency ω_1 we were hoping for.

In order to construct the signal we want, we must look at the phase derivative between adjacent analysis FFT frames and add it to the phase from the most recently synthesized IFFT frame. In [13] Flanagan makes the clever observation that a sinusoids phase at any time can be found by summing the phase derivative up until the desired time. Now if we combine this observation, with our previous one; in other words that phase can be calculated by accumulating phase derivatives, and we don't have to read frames in chronological order, and use them to create a new signal, we see in the following figure that the synthesized signal is exactly what we want it to be. The summing of phase derivatives smoothes the discontinuities we saw in the above figure, in the same way that integration creates smooth functions. The following table has the FFT phase values of the Fourier coefficient $X[\omega_1]$ and the corresponding IFFT synthesis phase for $Y[\omega_1]$, which are phases for each Fourier coefficient's corresponding 3D complex sinusoid.

frame	FFT1 phase	FFT2 phase	phase difference	IFFT1 phase	IFFT2 phase
1	$\frac{\pi}{2}$		0	$\frac{\pi}{2}$	
2		$-\frac{\pi}{2}$	π		$-\frac{\pi}{2}$
3	$-\frac{\pi}{2}$		π	$\frac{\pi}{2}$	
4		$\frac{\pi}{2}$	π		$-\frac{\pi}{2}$
5	$\frac{\pi}{2}$		π	$\frac{\pi}{2}$	
6		$-\frac{\pi}{2}$	π		$-\frac{\pi}{2}$
7	$-\frac{\pi}{2}$		π	$\frac{\pi}{2}$	
8		$\frac{\pi}{2}$	π		$-\frac{\pi}{2}$

So the basic idea is the following: if we measure how much the phase changed between two analysis time points, spaced $\frac{N}{2}$ samples apart, then the output phase should change by the same

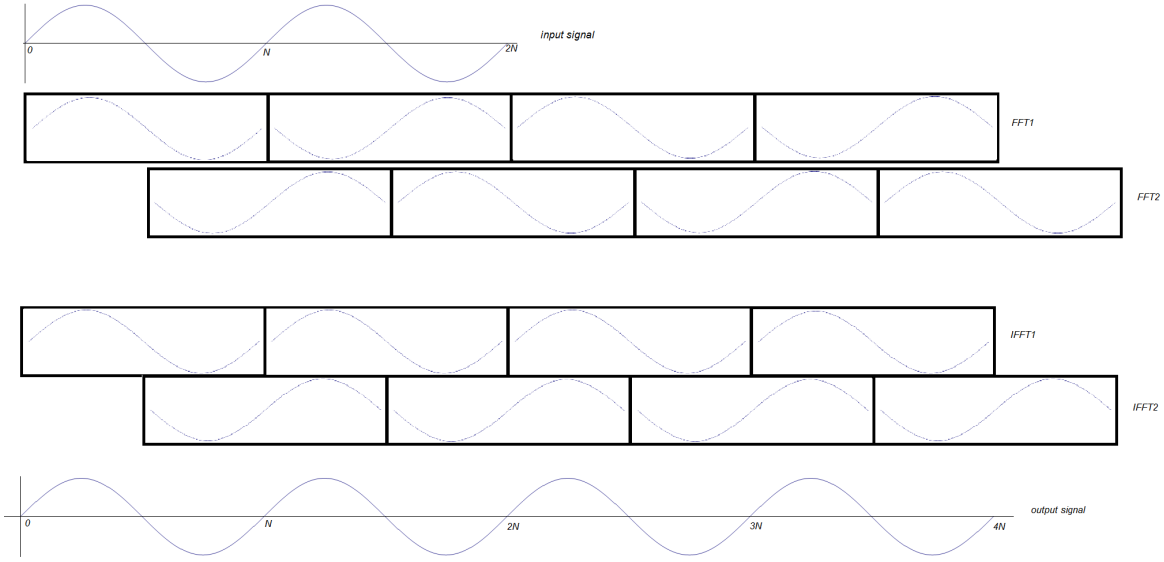


Figure 3.1.3. Time stretching by reading FFT frames out of order. First take the phase difference between adjacency analysis frames, and the calculate the synthesis phase by adding the difference to the most recently synthesized frame's phase.

amount, since the synthesis points are also spaced $\frac{N}{2}$ samples apart. So we just add the difference we calculated from the analysis frames to the synthesis phase calculated $\frac{N}{2}$ samples ago, in order to create the current synthesis phase. The above example as the advantage that the time modification factor is equal to $\frac{1}{R}$, so each FFT simply recalls the samples it previous analyzed to calculate the phase difference. In instances where this is not the case, each FFT instance will need another FFT instance paired with it, simply to calculate the previously analyzed samples. However, we keep the above example simple in order to see the mechanics of the Phase Vocoder.

We see this approach works for a simple input signal, a single sine wave. However, what about for any arbitrary signal? Think back to the whole concept of Fourier theory: any arbitrary signal can be modeled as a sum of sines and cosines! Therefore it is perfectly reasonable (and true) that this algorithm will have the same result for any input signal, since the calculation takes place channel by channel, in other words sinusoid by sinusoid. So if we bring back our oscillator bank analogy from chapter 2, we see that the Phase Vocoder keeps the oscillators humming continu-

ously. And as we will see shortly, we retain the unaltered amplitude information for each current FFT's analysis frame (which is reading through the signal at the rate of time modification), and therefore the amplitudes will be properly scaled to the desired time modification. In short: if we think of a signal as the sum of a bank of oscillators, the Phase Vocoder keeps the oscillators oscillating continuously, and scales each oscillator's envelope to the desired time modification.

3.2 Theory of Operation

Now let's formalize what we have formerly informally have shown is a pretty good way to effect a signal's time/frequency content independently. Flanagan gives a nice simple formula in his original Phase Vocoder paper, where he models a signal as the summed output of a bank of bandpass filters. We will start with this formula, and then rather than thinking of the a bank of n bandpass filters, we will think of a bank of n sinusoids. First we should note the advantage a physical system model such as a bandpass filter bank has. The output of a physical filter is not only dependent on the current input, but also all of the previous inputs up to the current one. So in order to correctly construct the output of a physical filter at a particular moment in time, one must go back to when the input started, and sum all of the outputs up until the present moment. If $f_n(t)$ is the output of the n^{th} bandpass filter, where its impulse response is $g_n(t) = h(t)cos(\omega_n t)$ and $h(t)$ is the impulse response of a physical low pass filter, at time t, then we see that

$$f_n(t) = \int_0^t f(\lambda)h(t-\lambda)cos(\omega_n(t-\lambda))d\lambda$$

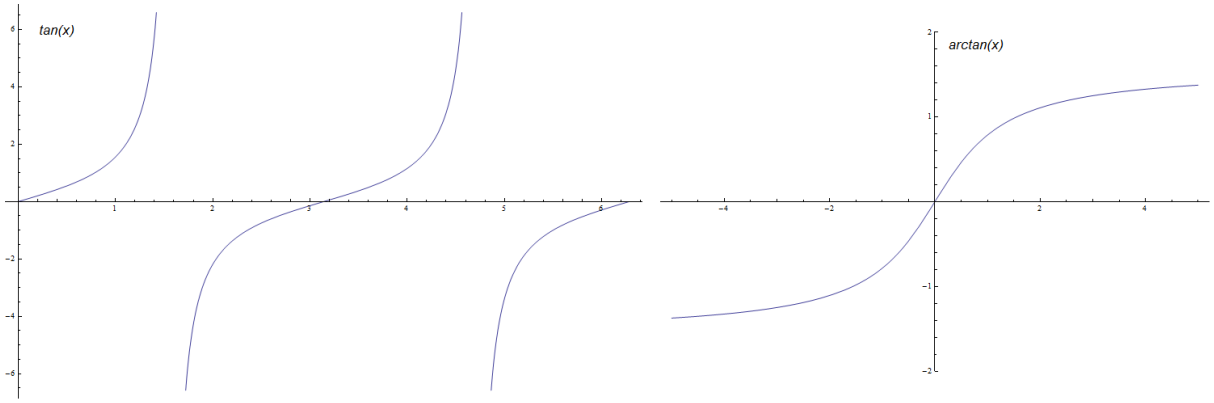
Then he makes the clever move to show that this is the real Fourier Transform of a signal $f(t)$ such that

$$f_n(t) = Re\Big[e^{i\omega_n t}\int_0^t f(\lambda)h(t-\lambda)e^{-i\omega_n\lambda}d\lambda\Big]$$

By then denoting the Fourier Transform of $f(t)$ by $\mathcal{F}(\omega_n, t)$, he shows that

$$f_n(t) = Re[e^{i\omega_n t}\mathcal{F}(\omega_n, t)] = \big|\mathcal{F}(\omega_n, t)\big|cos(\omega_n t + \phi(\omega_n, t))$$

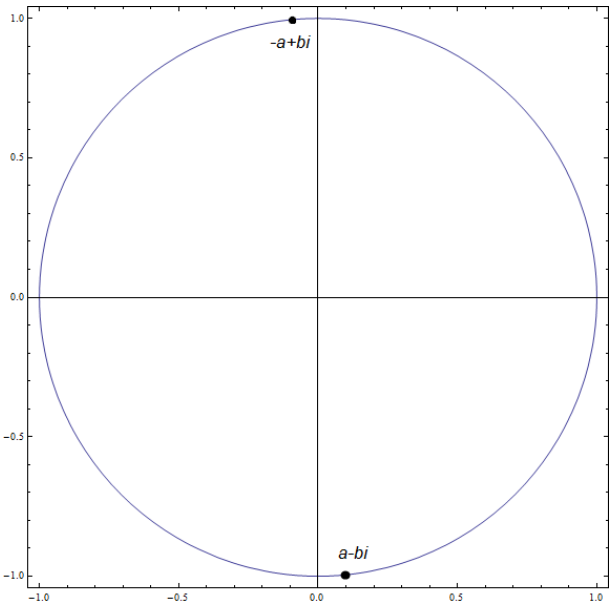
where $\phi(\omega_n, t)$ is the short-time phase spectra, and both $\left|\mathcal{F}(\omega_n, t)\right|$ and $\phi(\omega_n, t)$ are evaluated at ω_n. Normally ϕ is calculated using the *arctan* function. However, there is a huge problem with this. First, since we must choose a range to restrict our angle θ when calculating $\theta = arctan(\frac{b}{a})$, we know there exists asymptotes at the endpoints of the range. Crossing the asymptote will result in huge jumps in *tan* and *arctan*. We see this in the following figure which shows the *tan* and *arctan* functions.



We see that $arctan(\frac{b}{a}) = \theta$ is restricted from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$, and that for some $X[\omega_k] = -a + bi$, that

$$\angle(-a + bi) = arctan\Big(\frac{b}{-a}\Big) = arctan\Big(\frac{-b}{a}\Big) = \angle(a - bi)$$

Clearly the point $-a + bi \neq a - bi$, and their arguments differ by π. Because of the restricted range of *arctan*, the argument of the former gets confused with the argument of the latter. However, notice the derivative on either side of the asymptote. While the value of *arctan* will jump drastically when we go across the discontinuity, the derivatives on either side of the discontinuity are very similar, and in fact continuous! This is perhaps more noticeable in the graph for *tan*, but simply think of the the graph of *arctan* as the graph of *tan* rotated 90 degrees. Therefore, harking back to our physical filter model where we summed the outputs of the inputs across time to find the current output, if we sum the phase derivatives across time to calculate the current phase of ω_n, since the phase derivative is continuous, our phase calculation will be much



smoother and better behaved than the *arctan* function, and not exhibit jumps of π radians. This is the primary advantage of calculating phase by accumulating phase derivatives and is the first of two monumental observations by Flanagan. He goes on to rewrite the above equation in terms of the phase derivative,

$$f_n(t) = \big|\mathcal{F}(\omega_n, t)\big| cos(\omega_n t + \tilde{\phi}(\omega_n, t)) \text{ where } \tilde{\phi}(\omega_n, t) = \int_0^t \dot{\phi}(\omega_n, t) dt$$

for the phase derivative $\dot{\phi}(\omega_n, t)$. This approach does have the disadvantage of losing any additive phase constants, however, Flanagan notes the the loss of the phase constant is not particularly noticeable. If instead of the output of a band pass filter, we think of a sinusoid at frequency ω_n and its unit circle plot from chapter 2, we see that our sinusoids motion around the unit circle will be continuous, instead of exhibiting phase jumps of π. We can think of the additive phase constant as having the effect of rotating our points from figure 2.0.2 around the unit circle, while still maintaining the inner point relationships.

Let's restate the above equations in terms of complex sinusoids, instead of bandpass filters. Let us denote the analysis FFT spectral data of an input signal $x[t_n]$ for the k^{th} frequency over frame u as $X[t_a^u, \omega_k]$. Similarly, the output signal $y[t_n]$ will be synthesized from the IFFT data $Y[t_s^u, \omega_k]$, where the subscripts a and s denote the analysis or synthesis time points respectively.

We see that

$$x[t_n] = \sum_{k=0}^{N-1} |X[t_a^u, \omega_k]| e^{-i(\omega_k t_n + \angle \tilde{X}[t_a^u, \omega_k])}$$

where

$$\angle \tilde{X}[t_a^u, \omega_k] = \sum_{p=0}^{u} \angle X[t_a^p, \omega_k] - \angle X[t_a^{p-1}, \omega_k]$$

Thus we can reconstruct a signal from its phase derivative spectrum, rather than just its phase spectrum, which will give us more well behaved approximation.

Here we see that we are thinking of sinusoids as functions of phase, which we showed was another way to think of them in chapter 1. Let's try to understand frequency and phase a bit more, and see what relationships we can draw between the two. Frequency has units radians or cycles per period of time. Phase has units radians, degrees, or "cycle-index", which runs from 0-1. All of these ways of describing phase tell you where a sinusoid is in its cycle. Now let's look at what we get when we take the difference of a sinusoid's phase between time points t_b and t_a over the time period $t_b - t_a$.

$$\frac{\phi(t_b) - \phi(t_a)}{t_b - t_a}$$

The above equation has units change in cycle over change in time, which shakes out to cycles per period of time, in other words frequency. So if we express the change in the phase over a time period of a sinusoid at frequency ω_k we see that

$$\frac{\phi_k(t_b) - \phi_k(t_a)}{t_b - t_a} = \omega_k(t)$$

Let's think about how the left hand side of the equation looks. Notice that this is the definition for the derivative of a function! Therefore we can draw the following conclusion

$$\frac{d}{dt}\phi(t) = \omega(t)$$

which loosely implies that

$$\int \omega(t)dt = \phi(t)$$

So far we have only looked at the math to redefine the Fourier Transform in terms of the phase derivative. This is the first realization Flanagan has in his paper. The second one is what this fact implies for time stretching and compression. He writes that an attractive feature of the Phase Vocoder is that the operations for expansion and compression of the time and frequency scales can be carried out by simple scaling of the phase-derivative [13]. Laroche and Dolson give a nice formula in [4] for a time scaling factor of $\alpha = \frac{t_s^u}{t_a^u}$, which uses the frequency/phase relationships we saw above.

$$\phi(\omega_k, t_s^u) = \phi_s(\omega_k, 0) + \int_0^{t_s^u} \omega_k\Big(\frac{\lambda}{\alpha}\Big) d\lambda$$

where ω_k is the *instantaneous frequency* of the k^{th} sinusoid. This is the ideal phase for ω_k at time t_s^u. If we do some u-substitution, we see that

$$\text{let } u = \frac{\lambda}{\alpha} \Rightarrow du = \frac{d\lambda}{\alpha} \Rightarrow d\lambda = \alpha \cdot du$$

$$\text{and furthermore } \frac{t_s^u}{\alpha} = \frac{\alpha t_a^u}{\alpha} = t_a^u$$

So our above integral becomes

$$\int_0^{t_s^u} \omega_k\Big(\frac{\lambda}{\alpha}\Big) d\lambda = \int_0^{t_a^u} \omega_k(u)\alpha \cdot du = \alpha \int_0^{t_a^u} \omega_k(u) du$$

Thus we see that

$$\phi(\omega_k, t_s^u) = \phi_s(\omega_k, 0) + = \alpha \int_0^{t_a^u} \omega_k(u) du$$

$$= \phi_s(\omega_k, 0) + \alpha\big[\phi_k(t_a^u) - \phi_k(0)\big]$$

What this is telling us is that the phase evolution up to a desired time $t_s^u = \alpha t_a^u$ is equal to α times the phase evolution up to t_a^u. We can intuit why this is true, since change of phase is constant with respect to time. We can show this by looking at the phase of ω_k equal to 2π radians per T_k seconds, $\phi_k(t) = \frac{2\pi t}{T}$ such that

$$\phi_k(\alpha t) = \frac{2\pi\alpha t}{T_k} = \frac{\alpha}{1}\frac{2\pi t}{T_k} = \alpha\phi_k(t)$$

So again switching to notation more common with DSP, it follows that

$$\alpha\angle X[t_a^p, \omega_k] = \angle X[\alpha t_a^u, \omega_k]$$

Let's synthesize a signal $y[t_n]$ from $Y[t_s^u]$ by scaling the time of $x[t_n]$ by α when $\alpha > 1$ results in time stretching and $\alpha < 1$ results in time compression, such that

$$\angle Y[t_s^u, \omega_k] = \alpha\angle\tilde{X}[t_a^u, \omega_k] = \alpha \sum_{p=0}^{u} \angle X[t_a^p, \omega_k] - \angle X[t_a^{p-1}, \omega_k]$$

$$= \sum_{p=0}^{u} \angle X[\alpha t_a^p, \omega_k] - \angle X[\alpha t_a^{p-1}, \omega_k]$$

for $t_s^u = \alpha t_a^u$. We see that this can be simplified a bit by noticing that

$$= \sum_{p=0}^{u-1} \Big[\angle X[\alpha t_a^p, \omega_k] - \angle X[\alpha t_a^{p-1}, \omega_k]\Big] + \angle X[\alpha t_a^u, \omega_k] - \angle X[\alpha t_a^{u-1}, \omega_k]$$

$$= \angle Y[t_s^{u-1}, \omega_k] + \angle X[\alpha t_a^u, \omega_k] - \angle X[\alpha t_a^{u-1}, \omega_k] = \angle Y[t_s^u, \omega_k]$$

So we can calculate the current synthesis phase $\angle Y[t_s^u, \omega_k]$ by adding the difference between the analysis phases $\angle X[t_a^u, \omega_k]$ and $\angle X[t_a^{u-1}, \omega_k]$ to the previous synthesis phase $\angle Y[t_s^{u-1}, \omega_k]$. This is precisely the approach we took when looking at the sine wave in the beginning of the chapter. So we calculate our time stretched signal $y[t_n]$ where

$$y[t_n] = \sum_{k=0}^{N-1} |X[t_a^u, \omega_k]| e^{-i(\omega_k t_n + \angle Y[t_a^u, \omega_k])}$$

Which is what we demonstrated in our initial example.

3.3 Some Notes on MAX/MSP Implementation of the Phase Vocoder

A few notable tutorials exist showing the object oriented implementation of the Phase Vocoder in Pure Data and Max/Msp. Namely Miller Puckette's "Phase Vocoder Time Bender", and Cort Lippe and Richard Dudas' "The Phase Vocoder Part I & II". These are wonderful examples of the phase vocoder, that give some serious DSP power to even the most casual patching enthusiast. However, when one is trying to abstract some of the bigger picture ideas from these tutorials, or see the manifestations of algorithms from articles on the PV, there are a few details that get

brushed over in the tutorials, or are not well documented in PD or Max documentation. Here we will clarify some of the details that are otherwise left out of such tutorials, but are integral to a full understanding of the PD and MAX/MSP Phase Vocoder patches.

3.3.1 Sampling Rate and pfft~

In our first step of the Phase Vocoder, we count through the sample whose timing we are modifying. When we see how this is done in PD and Max/Msp, there is a curious scalar of $\frac{1}{R}$ that persists across programs, and isn't given more of an explanation than *"for 4 times overlap"*. When building the Phase Vocoder without *pfft~* in Max or without *block~* in Pure Data, there seems to be some timing issues when we include this $\frac{1}{R}$ scalar. It doesn't seem to add up. If we are accumulating indices in Max at $\frac{1}{R}$ the rate, shouldn't the sample last for $hopfactor$ times longer? Which it does when we implement the algorithm outside of *pfft~* with the factor of $\frac{1}{R}$. And in Pure Data, shouldn't by counting through the same about of samples in $\frac{1}{R}$ the amount of time reduce the duration of the sample by a factor of R? Which it does when we use the Phase Vocoder outside of a subpatch.

However, because of some undocumented properties of *pfft~* and *block~* (*block~* is slightly better documented than *pfft~*), these scalars are in fact necessary, and are a result of many Fourier Transform instances running at once. Because the Phase Vocoder is running R FFT instances, the PV must process R times as much data in the same amount of time. Because of this, the sampling rate of these objects is different than the larger Phase Vocoder patch that they are apart of, namely R times larger. We see in the table below, that by including the factor R in our sample location calculation, the signal counting through the sample at the higher sampling rate, and the original sampling rate will agree at all possible points, *in time!* Consider 2 signals, f_1 and f_2, where f_2 is sampled 4 times faster than f_1.

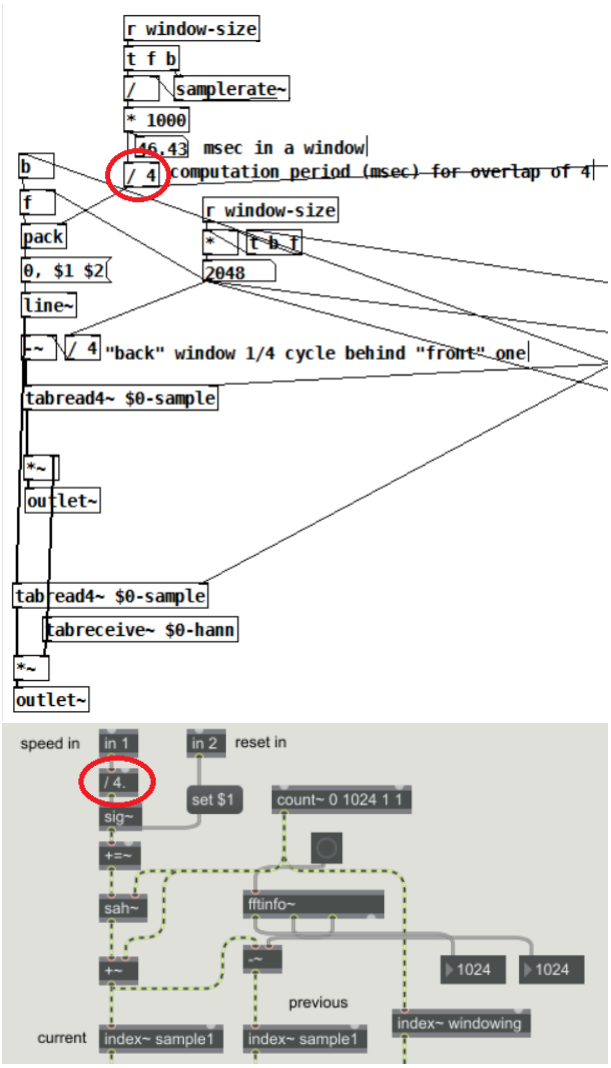


Figure 3.3.1. factor of $\frac{1}{R}$ included in two different Phase Vocoders. *top* PV from Miller Puckette's "Phase Vocoder Time Bender" tutorial. *bottom* PV from Cort Lippe and Richard Dudas' "Phase Vocoder Tutorial Part I".

sample number	f_2	f_1
0	0	0
1	0.25	1
2	0.5	2
3	0.75	3
4	1	4
5	1.25	5
6	1.5	6
7	1.75	7
8	2.0	8

We see that index by index the two signal do not agree. However let's think about when these sample land in time, when f_1 is sampled at 44100 Hz and f_2 is sampled at 176400 Hz. So 1 sample in f_2 represents $.00566ms$ and 1 sample in f_1 represents $.0226ms$. When we look at the samples in time, we see that they agree at each possible moment in time.

$t(ms)$	$f_2[t]$	$f_1[t]$
0.00	0	0
.00566	0.25	
.01133	0.5	
.01701	0.75	
.0226	1	1
.02834	1.25	
.03401	1.5	
.03968	1.75	
.04535	2	2

Thus we see that even though one signal is an up sampled version of the other, that they still agree in time.

3.3.2 Pitch Shifting

So far in our investigation into time/frequency independence we have only looked at modifying time without affecting a signal's frequency. However, it is also possible to modify the frequency and keep the timing of a signal the same. Phase vocoder pitch-shifting is rather straightforward, at least from a DSP perspective. For a given sample sampled at f_s, in order to modify its pitch by a factor of β, we simply count through the sample at a rate of βf_s and perform the Phase Vocoder algorithm normally otherwise. This is the same effect as playing a record faster or slower to change its pitch, as we pointed out earlier.

3.4 Cartesian vs. Polar Complex Math

Earlier, we looked at the linearity of the Fourier Transform using both cartesian/rectangular coordinates, and polar coordinates. This same duality exists for all complex operations using the Fourier Transform, including the Phase Vocoder. When we are subtracting or adding phases, we can either use the cartesian or polar representation of the Fourier coefficients. Clearly, polar is more intuitive, since the phase information is laid out in plain sight. However it should be noted

that the cartesian coordinate version is computationally far cheaper than the polar coordinate algebra, because of the problems we pointed out earlier with the *tan* and *arctan* functions, which are used when working with polar coordinates. However, the cartesian operations are perhaps a bit muddier conceptually. We will look at the cartesian algebra for the Phase Vocoder, and the steps for an accurate, and ultimately faster Phase Vocoder algorithm.

3.4.1 Complex Multiplication

We will start by looking at the polar representation of Fourier coefficients in order to gain some intuition on what algebraic operations do to the amplitude and phase. Consider the two complex numbers $z_1 = A_1 e^{i\theta_1}$ and $z_2 = A_2 e^{i\theta_2}$. When we multiply them together, we get the following

$$z_1 \times z_2 = A_1 e^{i\theta_1} \times A_2 e^{i\theta_2} = A_1 A_2 e^{i(\theta_1+\theta_2)}$$

Multiplying two complex numbers multiplies the amplitude and adds the phase. Now if we were to think of this is terms of real and imaginary parts, we see that for $z_1 = a + bi$ and $z_2 = c + di$

$$z_1 \times z_2 = (a + bi)(c + di) = ac - bi + i(ad + bc)$$

So because this is equivalent to $A_1 A_2 e^{i(\theta_1+\theta_2)}$, the phase of the complex number $ac-bi+i(ad+bc)$, must be $\theta_1 + \theta_2$ and the amplitude is $A_1 A_2$.

3.4.2 Complex Division

Complex multiplication takes care of our second step in the Phase Vocoder, adding the phases of two complex numbers, however we are first finding the phase difference between two complex numbers. In order to do this with cartesian coordinates, we must perform complex division. Again, let's first look at the polar representation

$$\frac{z_1}{z_2} = \frac{A_1 e^{i\theta_1}}{A_2 e^{i\theta_2}} = \frac{A_1}{A_2} e^{i(\theta_1} e^{-\theta_2)} = \frac{A_1}{A_2} e^{i(\theta_1-\theta_2)}$$

Dividing two complex numbers divides the numerator's amplitude by the denominator's amplitude, and subtracts the denominator's phase from the phase of the numerator. The cartesian algebra for complex division is a bit trickier, however, we will see that the desired result is still

achieved. Again consider the two complex numbers $z_1 = a + bi$ and $z_2 = c + di$. We see that dividing z_1 by z_2 yields

$$\frac{z_1}{z_2} = \frac{a+bi}{c+di}$$

Since real denominators are easier to do algebra with than complex denominators, let us multiply both the numerator and the denominator by the complex conjugate of z_2, $\bar{z_2} = (c - di)$

$$\frac{a+bi}{c+di} \times \frac{c-di}{c-di} = \frac{(a+bi)(c-di)}{(c+di)(c-di)} = \frac{(ac+bd)+i(bc-ad)}{c^2+d^2}$$

While this may look like a busier form of the complex number we started with, there exists some superfluous information as well as a nice symmetry which makes this form very useful. First off, notice that the real valued scalar $\frac{1}{c^2+d^2}$ is not needed. The phase of $\frac{(ac+bd)+i(bc-ad)}{c^2+d^2}$ is exactly the same as $(ac + bd) + i(bc - ad)$. We can see this by first breaking up the real and imaginary parts of the number

$$\frac{(ac+bd)+i(bc-ad)}{c^2+d^2} = \frac{(ac+bd)}{c^2+d^2} + \frac{i(bc-ad)}{c^2+d^2}$$

Now let us look at its *arctan* in order to find the phase information

$$arctan\left(\frac{\frac{(ac+bd)}{c^2+d^2}}{\frac{i(bc-ad)}{c^2+d^2}}\right) = arctan\Big(\frac{ac+bd}{bc-ad}\Big) = arg\big(ac+bd+i(bc-ad)\big)$$

While it is not true that all scalars $q \in \mathbb{R}$ don't effect a complex number z's argument, it is true that all scalars $q^* \in \mathbb{R}$ such that $q^* > 0$ leave the complex number z's phase unchanged! And since we are squaring both c and d, it must be the case that $q^* = c^2+d^2 > 0$ for all $c, d \in \mathbb{R}-\{0\}$.

Because the scalar $\frac{1}{c^2+d^2}$ has no effect on the phase, and right now we aren't concerned with the amplitude of $\frac{z_1}{z_2}$, let's throw out this factor of $\frac{1}{c^2+d^2}$ and only worry about $ac+bd+i(bc-ad)$. The second advantage of this form is how much it looks like cartesian multiplication. We have simply switched the subtraction and addition sign between each real and imaginary part. Recall that we used the complex conjugate of z_2 in order to get a form of $\frac{z_1}{z_2}$ that we liked. Since we end up ignoring the denominator, we see that dividing z_1 by z_2, as far as phase information is concerned, is just multiplying z_1 by the complex conjugate of z_2, $\bar{z_2}$. Again we can think to the

polar form of these numbers. The complex conjugate of a number z, has an argument $-arg(z)$. So if we want to subtract the arguments of two complex numbers, we see that it is the same as adding the argument of the conjugate, which is multiplying by the conjugate in cartesian form. Now we have two similar cartesian formulas which add or subtract the phases of two complex numbers while avoiding the use of the *arctan* function.

3.4.3 Inverse Modulus Scaling

Earlier we made the quick move to throw out the amplitude information, which could have seemed like a convenient way to get the phase information to look nicer and more workable. We were able to do this because in the Phase Vocoder, the only amplitude information we care about is the amplitude of the current frame we are analyzing. However, the problem of amplitude distortion is important, because we do care about at least one amplitude, and we only care that it stays the same. In order to keep amplitude information a Fourier coefficient unchanged, we must give whichever Fourier coefficient we are manipulating it with unity amplitude. Let us once again start with polar coordinates to keep the algebra simple and gain some intuition. Suppose for $z_1 = A_1 e^{i\theta_1}$ and $z_2 = A_2 e^{i\theta_2}$ we want a complex number with the sum of the phases of z_1 and z_2, but only the amplitude of z_1. By simply multiplying z_1 and z_2 we obtain

$$z_1 \times z_2 = A_1 e^{i\theta_1} \times A_2 e^{i\theta_2} = A_1 A_2 e^{i(\theta_1+\theta_2)}$$

The resultant phase is correct, but the amplitude is $A_1 A_2$, not just A_1. So in order to achieve this, we must first multiply z_2 by $\frac{1}{|z_2|} = \frac{1}{A_2}$. We now see that by multiplying z_1 by $\frac{1}{A_2} z_2$ we obtain

$$z_1 \times \frac{1}{A_2} z_2 = A_1 e^{i\theta_1} \times \frac{A_2}{A_2} e^{i\theta_2} = A_1 e^{i(\theta_1+\theta_2)}$$

Now the product has the amplitude of z_1 and the phase is the sum of the phase of z_1 and the phase of z_2, as we wanted. Now let's look at the cartesian form. For a complex number $z_2 = c + di$,

$$\frac{1}{|z_2|} = \frac{1}{A_2} = \frac{1}{\sqrt{c^2 + d^2}}$$

So to perform the above operation in cartesian coordinates, it follows that the complex number

$$(a+bi)\times\frac{1}{\sqrt{c^2+d^2}}(c+di)=\frac{ac-bd+i(ac+bd)}{\sqrt{c^2+d^2}}$$

has the amplitude of z_1 but the phase of $arg(z_1)+arg(z_2)$. Such is the operation performed when updating the phase for a channel in the Phase Vocoder. In this context, z_1 has the amplitude of the current analysis Fourier coefficient, and the phase increment that z_2 should advance by.

Now suppose we want a complex number whose argument is $arg(z_1)-arg(z_2)$ but whose amplitude is $|z_1|$. We see that by scaling z_2 by $\frac{1}{|z_2|}$, we obtain the following

$$z_1\div\frac{1}{A_2}z_2=\frac{A_2A_1e^{i\theta_1}}{A_2e^{i\theta_2}}=A_1e^{i(\theta_1-\theta_2)}$$

Now let us move to the cartesian representation. So for $z_2^*=(c+di)\frac{1}{|z_2|}$, we see that

$$\frac{z_1}{z_2^*}=\frac{a+bi}{\frac{1}{|z_2|}(c+di)}=\frac{a+bi}{c+di}\cdot\sqrt{c^2+d^2}$$

$$=\Big(\frac{(ac+bd)+i(bc-ad)}{c^2+d^2}\Big)\frac{\sqrt{c^2+d^2}}{1}=\frac{(ac+bd)+i(bc-ad)}{\sqrt{c^2+d^2}}$$

Again, this look remarkably similar to our cartesian complex multiplication, again the operations between the real and imaginary numbers are switched. So in the Phase Vocoder, for all operations with complex numbers which aren't the "current analysis" Fourier coefficient, we scale them by the inverse modulus in order to avoid amplitude distortion. Ultimately, if we think of our oscillator bank analogy for the Fourier Transform, this inverse modulus scaling ensures that the amplitude envelopes of each oscillator are preserved and scaled for any time modification factor α. We will be using these equations to calculate Phase Vocoder Fourier coefficients in Max/Msp or Pure Data.

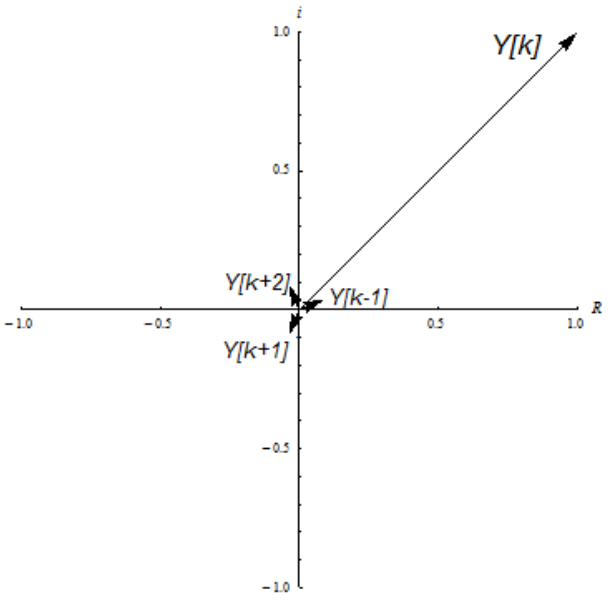
3.5 Miller Puckette's Phase Locking

In Chapter 2 we saw the effect of spectral leakage on a signal's spectrum. Sinusoids which don't fit nicely into the FFT's harmonic series end of affecting multiple bins. Under no spectral modifications, this does not matter, as we saw earlier in chapter 2 with figure 2.5.11. However, as we

change the frequency domain data, such as with algorithms like the Phase Vocoder, these false frequencies become more present in the output signal. *Vertical Phase Coherence* is the phase agreement of adjacent FFT bins so that false frequencies calculated as a result of the discrete nature of the FFT at least share the same time characteristics of the original signal. Spectral leakage is one cause of vertical phase coherence failure.

Miller Puckette addressed the problem of vertical phase coherence in his 1995 paper "Phase-locked Vocoder" by setting the synthesis phase of each channel of the Phase Vocoder to the sum of itself, and the synthesis phase of its two adjacent neighbors. The idea being that for two low amplitude bins on either side of a peak in the spectrum, the phase of the peak channel will stay relatively unchanged, while bringing the phase of the lower amplitude bins much closer to the phase of the peak. We will explore this idea with a simple example.

Consider a signal with the following 4 synthesis Fourier coefficients for channels $k-1, k, k+1, k+2$ with the following values.



bin	*Amplitude*	*Phase*	*Complex Number*
$k-1$	.1	$\frac{\pi}{8} \approx .3927$	$.0924 + .0393i$
k	1.0	$\frac{\pi}{4} \approx .7854$	$.707 + .707i$
$k+1$	.1	$\frac{11\pi}{8} \approx 4.3197$	$-.0393 - .0924i$
$k+2$	.1	$\frac{5\pi}{8} \approx 1.9635$	$-.0393 + .0924i$

Given this set of synthesis vectors, we will calculate the updated synthesis phases to be processed by the PV using Miller Puckette's idea.

$$\angle Y(k) = \angle\Big(Y(k-1) + Y(k) + Y(k+1)\Big)$$

$$\angle Y(k+1) = \angle\Big(Y(k) + Y(k+1) + Y(k+2)\Big)$$

it follows that

$$\angle Y(k) = \tan^{-1}\frac{.0393 + .707 - .0924}{.0924 + .707 - .0393} = .7104 \; radians$$

$$\angle Y(k+1) = \tan^{-1}\frac{.707 - .03939 - .0393}{.707 - .0924 + .0924} = .84417 \; radians$$

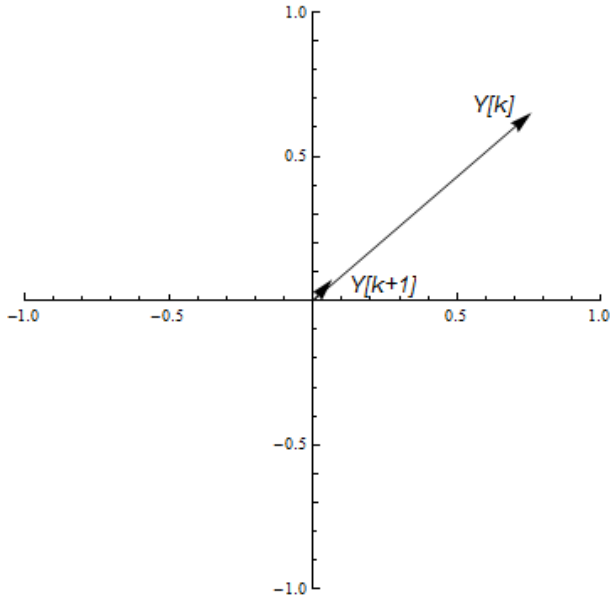


Figure 3.5.1. new synthesis FFT vectors after the phase has been corrected

We can see that the phase of bin $k+1$ has gravitated towards the phase of bin k, while bin k's phase has stayed relatively unchanged. Remember, the equation

$$y[t_n] = \sum_{k=0}^{N-1} Y[t_s^u, \omega_k] e^{i\omega_k t_n}$$

is just giving us one real number in the time domain, and we're getting the number from summing all of our complex vectors in nose to tail fashion. So the more the irrelevant vectors agree with the relevant ones, the closer of an approximation the finite sum is to the actual signal we are trying to construct.

4
Laroche's Approaches

Perhaps the most significant improvement to the Phase Vocoder was introduced by Jean Laroche and Mark Dolson in 1999 called *Identity Phase Locking*. Most subsequent improvements of the PV are just modified Identity Phase Locking algorithms. Here we will look at how Laroche and Dolson attempted to instill vertical phase coherence, as well as adjustments made to their foundational approach in the past two decades. We saw above that the output phase of the IFFT is calculated by adding the phase difference between analysis time points to the most recent phase calculated for each frequency ω_k. This assumes that the phase changed uniformly between the analysis points, as a result of that assumption, uniform change is reflected in the output. However, what if a sinusoid's phase changed non-uniformly over the N sample time period, in other words, its frequency changed over the course of analysis. While we saw earlier that discrete sampling limited our frequency resolution, now we see here that it also limits our time resolution. If we look at the N sample spectrum of a discrete time signal, we have no idea when the peaks in the spectrum occured within the N sample window. And if a sinusoids frequency changes within an analysis frame, it is entirely possible (and pretty likely) that it will effect multiple spectral bins, in which case, the phase for a *set* of bins should change continuously, as the sinusoid moves through each bin, rather than changing continuously bin by bin. This was the core observation made by Laroche and Dolson which prompted their Identity Phase Locking

and Scaled Phase Locking algorithms. Sinusoids whose frequency changes within an analysis frame is the second source of vertical phase coherence failure.

4.1 Identity Phase Locking

In Jean Laroche and Mark Dolson's 1999 paper *Improved Phase Vocoder Time-Scale Modification of Audio*, new techniques were proposed expounding upon the simple idea set forth by Miller Puckette. Laroche and Dolson's Identity Phase Locking preserves the relationship between analysis and synthesis phases for spectral peaks, and mimics that relationship for channels neighboring the peak. Laroche and Dolson define a *Spectral Peak*, denoted ω_{k_l}, as a channel which has a larger amplitude than the amplitudes of the 2 bins on each side of it. These other two channels on either side of the peak are said to neighbor the peak. Neighboring channels lie in the peak's region on influence. The size of the region of influence is the distance from one peak to the next, and is not necessarily the same size on either "side" of the peak, we will see what this means later. For all channels ω_k which lie in channel ω_{k_l}'s region of influence, their synthesis phases are computed using the following equation

$$\angle Y(t_s^u, \omega_k) = \angle Y(t_s^u, \omega_{k_l}) - \angle X(t_a^u, \omega_{k_l}) + \angle X(t_a^u, \omega_k)$$

Just from a glance we can see that this technique has an obvious computational advantage. Because we are calculating the synthesis phase for each channel based on the synthesis phase of a spectral peak and the analysis phases of each, we don't need to perform trigonometric calculations in order to get synthesis phases on any channels except for the peaks. First we calculate the synthesis phase for all peak channels using the regular phase vocoder algorithm. Then the difference between the peak channel's synthesis phase and analysis phase is calculated.

$$\theta = \angle Y(t_s^u, \omega_{k_l}) - \angle X(t_a^u, \omega_{k_l})$$

From that we generate a new complex sinusoid

$$Z = e^{i\theta}$$

and multiply the analysis Fourier coefficients in the peak's region of influence by it in order to calculate the synthesis phase for each Fourier coefficient.

$$Y(t_s^u, \omega_k) = Z \cdot X(t_s^u, \omega_k)$$

4.2 Identity Phase Locking in Max/Msp

Here we will look at implementing Identity Phase Locking in Max/Msp for real-time frequency/time independent manipulation. The following sequence of steps is given in [4], which breaks down Identity Phase Locking.

1. Coarse Peak Picking Stage
2. Calculation of $\angle Y(t_s^u, \omega_{k_l})$ for peak channels ω_{k_l} only
3. Calculate the angle θ and the phasor $e^{i\theta_{k_l}}$
4. Calculate $Y(t_a^u, \omega_k) = X(t_a^u, \omega_k)e^{i\theta_{k_l}}$ for channels k in k_l's region of influence.

While the list of steps is short, and operations are simple enough, creating a patch that processes audio in real time with the desired result in not as trivial. The first obstacle that must be overcome is knowing information about the current analysis frame, before we analyze it.

4.2.1 Double Buffering

Double buffering is a technique used to address the problem we just mentioned, and is common practice in image processing but has other applications as well. Double buffering writes data to one buffer, while reading from another buffer. When we are finished reading the data from one buffer, we then begin writing data to it, and then read from the buffer we were just writing to. Now we can see that if we are looking at sample n, and we want to know data about sample $n + 2$, we simply look two samples ahead in the buffer which we have already written the data to. However, this has the obvious disadvantage of introducing a frame of lag. So if we want to look at a certain set of data, at a certain time, we must process the data one frame ahead of

when we want to analyze it.

The first place we will use double buffering is in the peak picking stage of Identity Phase Locking. We need to know the spectral amplitude peaks in an analysis frame so we know which bins whose raw Phase Vocoder synthesis phase we should calculate. To do this, we need to know the channel with the largest amplitude in a set of 5, so we want to find the channels k such that k has a larger amplitude than channels $k-2$, $k-1$, $k+1$, and $k+2$. By writing the spectral amplitude data into a buffer one frame, and then reading from it the next, we can simultaneously look at channels k and $k+2$. And since we are reading from a sound file, this isn't hard at all. Once we know the spectral peaks we can then go ahead and calculate their synthesis phase in order to generate the phasors $Z = e^{i\theta}$.

4.2.2 Phasor Retro Fitting

The second obstacle to overcome when performing Identity Phase Locking is using the correct phasor when calculating the updated synthesis phase for each bin. The problem arises when we want to stop using phasor Z_l and want to start using phasor Z_{l+1}. The point where we switch phasors is either set to the middle frequency $\omega = \frac{\omega_{k_l} + \omega_{k_{l+1}}}{2}$, or the middle bin $\frac{k_{l+1} - k_l}{2}$. Earlier for the peak picking stage, we simply wanted to know if a channel was a peak or not, this was simple enough because we had a fixed neighborhood size, and we were performing the calculation on the sample we were working with. However, it is not guaranteed that $\frac{k_{l+1} - k_l}{2} = \frac{k_{l+2} - k_{l+1}}{2}$. We need to know if a channel is halfway between two peaks or not; and if it is, to start using the next phasor, which up until now we haven't calculated yet! Consider the following table which has some possible analysis Fourier coefficients and illustrates how we are trying to distribute the phasors.

k	$\lvert X = [t_a^u, \omega_k]\rvert$	$Y[t_s^u]$
0	0.1	$e^{i\theta_1} X[t_a^u, \omega_0]$
1	1	$e^{i\theta_1} X[t_a^u, \omega_1]$
2	0.1	$e^{i\theta_1} X[t_a^u, \omega_2]$
3	0.1	$e^{i\theta_1} X[t_a^u, \omega_3]$
4	0.1	$e^{i\theta_6} X[t_a^u, \omega_4]$
5	0.1	$e^{i\theta_6} X[t_a^u, \omega_5]$
6	1	$e^{i\theta_6} X[t_a^u, \omega_6]$
7	0.1	$e^{i\theta_6} X[t_a^u, \omega_7]$
8	0.1	$e^{i\theta_9} X[t_a^u, \omega_8]$
9	1	$e^{i\theta_9} X[t_a^u, \omega_9]$
10	0.1	$e^{i\theta_9} X[t_a^u, \omega_{10}]$
11	0.1	$e^{i\theta_{13}} X[t_a^u, \omega_{11}]$
12	0.1	$e^{i\theta_{13}} X[t_a^u, \omega_{12}]$
13	1	$e^{i\theta_{13}} X[t_a^u, \omega_{13}]$

We see that even though the neighborhood size is fixed, the distance between peaks, and therefore their neighborhoods, can vary. In the above table we see a couple of different cases for neighborhood boundary distribution. The neighborhood boundaries are marked by the small space between rows. We see that the distance between the first peak ω_1 and the second peak ω_6 is 4 bins. Therefore, if we mark the boundary as $\frac{k_{l+1}-k_l-1}{2}$ above the lower peak, the decision on where to switch phasors is trivial. We stop using $e^{i\theta_1}$ at ω_4 and start using $e^{i\theta_6}$, where $Y[t_s^u, \omega_4] = e^{i\theta_6} \cdot X[t_a^u, \omega_4]$. The size of the region of influence above ω_1 and below ω_6 is 2 bins.

Moving on to the next peak, ω_9, we see that the distance between ω_6 and ω_9 is different than the distance between ω_1 and ω_6. However, even though the distance between these two peaks is different than between the previous two peaks, the decision on where to stop using one phasor and start using the next is still trivial. We stop using $e^{i\theta_1}$ at bin $k_6 + \frac{k_9-k_6-1}{2} = 8$. The region of influence size above ω_6 and below ω_9 is 1 bin.

The size of the region of influence above ω_9 and below the next peak ω_{13} is not as trivial as it was for the first two peaks. Even though the first two peaks had different sized regions of influences, it was still easy to pick out the boundary, since the amount of bins in between the peaks was an even number. However, we see that there are an odd number of bins between peak

ω_9 and ω_{13}. The size of the region of influence above the lower peak, will be different than the size of the region of influence below the higher peak. Therefore we must make a decision on how to divide the bins in between. In this case, as we saw in the above table, we give make the region of influence above the lower peak the smaller portion of the difference, and the region of influence below the higher peak the larger portion. This decision is based on the logarithmic distribution of musical spectral data. Later we will an Identity Phase Locking approach which takes this consideration a step further. But for now, while the difference isn't overwhelming, we see that giving the higher peak a larger region of influence mimics a logarithmic neighborhood size a bit more.

One approach to navigating these possible cases of varying and lopsided regions of influence, is to know the distance above and below every peak to the next, before we calculate their phasors. The following table outlines the steps were such an approach to be taken.

frame	process	known
n	identify peak amplitudes	
$n+1$	calculate difference between peaks	peak amplitudes
$n+2$	calculate phasors and synthesis phase	difference between peaks

However, we are able to cut out one step based on another property of double buffering. Because the data we are writing won't be used until the next frame, we can temporarily write incorrect data, as long as it is corrected by the next frame. What this means is that instead of simultaneously writing phasor data for the peak and the bins in its region of influence above and below, which requires knowing how large the region of influence above and below is, we can instead only write phasor data for the peak channel and the bins in its lower region of influence, because those are the only peaks we already know. These doesn't save us the extra double buffer, however, it does save us an extra step. I call this process *Phasor Retrofitting* and the process is outline in the following table

frame	process	known
n	identify peak amplitudes	
$n+1$	phasors for lower regions of influence	peak amplitudes
$n+2$	calculate synthesis phase	phasors

Interestingly, there is no guarantee that there is more than one spectral peak. If we think of the graph of the amplitude spectrum, we see that the peaks occur at points of inflection, where the derivative changes from positive to negative. The peaks are the points of inflection such that each of the two points below have a positive derivative, and each of the two points above a have negative derivative. Therefore, if an amplitude spectra has either strictly positive or strictly negative derivatives, there would only be one peak. Now such a spectrum is extremely unlikely, however, what we can learn from this observation is that there is no guarantee on the density of peaks in the spectrum. The distribution of spectral peaks can be as high as 1 every 3 bins, or as low as 1 out of all bins.

4.3 Scaled Phase Locking

In the same paper, [4], Laroche and Dolson propose an improvement that builds upon their Identity Phase Locking, *Scaled Phase Locking*. Scaled Phase Locking recognizes that if a sinusoid changes from bin k to bin $k+1$, that in order to properly ensure that the peak has a continuous phase value, the phase difference calculated should be

$$\angle X[t_a^{u-1}, \omega_{k+1}] - \angle X[t_a^{u-1}, \omega_k]$$

and should be accumulated to $\angle Y[t_s^{u-1}, \omega_k]$. We can see the aforementioned case in the figure 4.3.1. So our equation for updating the phase of a peak becomes

$$\angle Y[t_s^u, \omega_{k+1}] = \angle Y[t_s^{u-1}, \omega_k] + \angle X[t_a^u, \omega_{k+1}] - \angle X[t_a^{u-1}, \omega_k]$$

As pointed out by Laroche and Dolson, such an approach would correctly calculate $\phi(\omega_k, t_s^u)$ from section 3.2. In the above figure, we can see the simplest case of sinusoidal movement: a sinusoid moving in one direction, one bin at a time. However, we can imagine far more complicated trajectories a single peak could take. Suppose a violin player glissandos from $E_5 = 659.3Hz$ to $B_7 = 3951.07Hz$ in one frame's time; for a FFT frame size of 4096 samples this would be

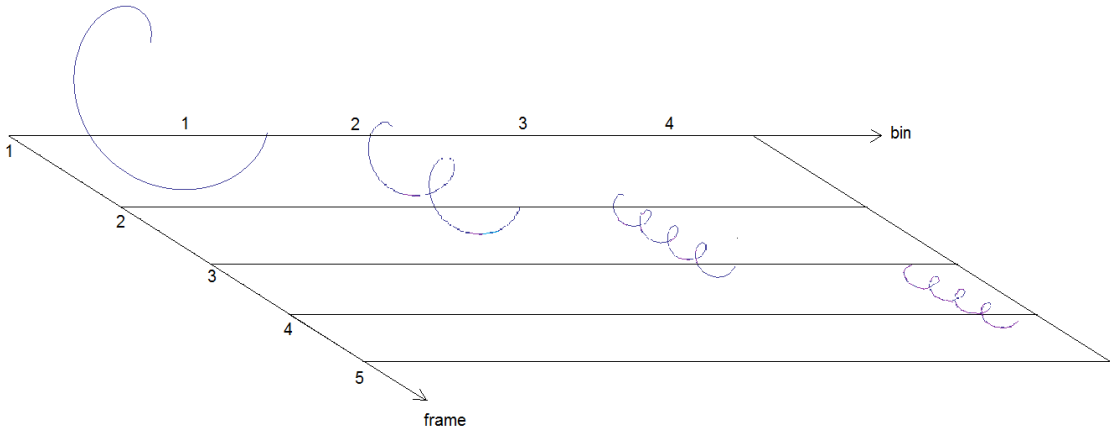


Figure 4.3.1. sinusoid moving through the spectrum

from bin 61 to bin 366. How is the Scaled Phase Locking supposed to know to look so far down the spectrum, and skip possible peaks in between, to pick out its previous location and phase update that value? It is at this point that one must sacrifice the freedom of real-time manipulation, in order to construct a time modified signal, which is outside the scope of this project. The heuristics needed to link peaks to their predecessors in a previous frame, and/or distinguish them from the onset of new peaks not present in the previous frame are not complete enough to accomodate any arbitrary signal in real time. One must first record the raw spectral data, then, knowing the trajectory of peaks in the spectrum, guide the Scaled Phase Locking algorithm to follow these peaks through time according to how they change, as well as note when a new peak arrives in the spectrum. While Scaled Phase Locking is not general enough for any arbitrary signal to be manipulated in real time, it is the most substantial step towards perfect sinusoidal tracking in order to instill vertical phase coherence in the Phase Vocoder.

4.3.1 PhaVoRIT

In the 2006 paper by Thorsten Karrer, Eric Lee, and Jan Borchers "PhaVoRIT: A Phase Vocoder for Real-Time Interactive Time-Stretching" some important observations are made about the structure of musical signals that inform how we perform phase updating in the Phase Vocoder. Namely how the nonlinearity of human hearing should inform our approach to peak picking in

the spectrum.

An important observation to make when Phase Locking a musical signal, is the proximity of intentional components in the spectrum. We want to avoid flagging a channel as noise, and updating its phase as such, when in fact it is an intentional part of the analysis signal, and should have its phase updated as if it were a peak. This is an issue for lower bins in the spectrum, and for low bass tones, as identified by [11]. When a low note is present in a signal being analyzed by the PV, and parts of its harmonic series fall in the region of influence of another spectral peak, the harmonic's phase is updated as if it were noise in the spectrum, which is incorrect. For example, if for an FFT size frame size of 4096, there exist a $20Hz$ tone, located in bin 2 of the spectrum, and it has a larger amplitude than its first harmonic, $40Hz$, located in bin 4 of the spectrum, we see that the first harmonic falls under the region of influence of the fundamental, given a neighborhood of 5 channels as proposed by Laroche and Dolson in [4]. Thus its phase is updated based on how much the fundamental's phase has changed from synthesis to analysis, which is half as much as the phase for the first harmonic should have changed. In order to avoid such miscalculations, [11] suggests adjusting the regions of influence based on where they fall in spectrum. Clearly a $20Hz$ difference won't affect a $400Hz$ fundamental or any of its harmonics, so it should not be treated as such, yet as we saw $20Hz$ does matter for bins lower in the spectrum. Such an observation results in deciding the size of each region of influence as a function of frequency. The larger the frequency of a peak, the larger its region of influence should be.

Karrer, Lee, and Borchers took such an approach in [11] for Laroche and Dolson's Scaled Phase Locking. Here we will take such an approach to the more general, more real-time, Identity Phase Locking. In the patch, when a neighborhood changes from 2 bins to 4 bins is a variable parameter set by the user. Same for switching from 4 bins to 6 bins, and from 6 bins to 8 bins. The following table shows a pretty good distribution of each region of influence, as showed by informal listening tests.

bin range	frequency range (Hz)	neighborhood size
$0-50$	$0-538.5$	unaffected
$51-100$	$549.5-1077$	2
$101-200$	$1087.8-2154$	4
$201-1000$	$2164.7-10770$	6
$1001-2048$	$10780.7-22050$	8

5
Other Algorithms

Here ends our purely mathematical pursuit of time modification. This chapter will look at results for time modifications if we loosen the constraints of strict mathematical accuracy. While we have been focusing on literature from physicists, engineers, and mathematicians thus far, there exists a whole other body of literature written by composers and academics who are thinking about these ideas in a different way. Some of these include Matt Sargent, Cort Lippe, JoAnn Kuchera-Morin, and Iannis Xenakis. Included in this chapter is Granular Synthesis, the Grain Vocoder, and a new approach to the Phase Vocoder, the Saphe Covoder. And while we may become less textbook with our use of math, this newfound freedom will bring with it some interesting mathematical perspectives on how we interact with some of the ideas we have already explored.

5.1 Granular Synthesis

So far our topics of investigation have all included some form of the Fourier Transform. When using Fourier's theory with computers we have diverted to the Discrete Fourier Transform, which, as we have seen, creates a whole lot of problems as a result of trying to model a signal after perfectly period sinusoids of any frequency for an infinite amount of time. We will see that by abandoning the pursuit of theoretical limits, and embracing the limits of the human

auditory and perceptual systems, some curious results come about when considering the different representations of a signal.

5.1.1 Dennis Gabor

The physicist Dennis Gabor wrote about human perception as it relates to communications in [16] and then later specifically regarding acoustics in [17]. His realization was that the Fourier view of sound in accurately describes how we experience it. The frequency description is timeless, where waves are perfectly periodic and have infinite duration; and the time description is infinite with respect to its component frequencies. The ear does not experience all real number frequencies between 20 and 20000 Hz, nor do we have a limitless perception of sound in time. Gabor presented his idea of sound being comprised of what he calls elementary particles or elementary signals, as a perceptual measurement of signals. Signals, or particles, are represented in an information diagram by their characteristic cell, which is an area plotted from the frequency vs time axes of the information diagram.

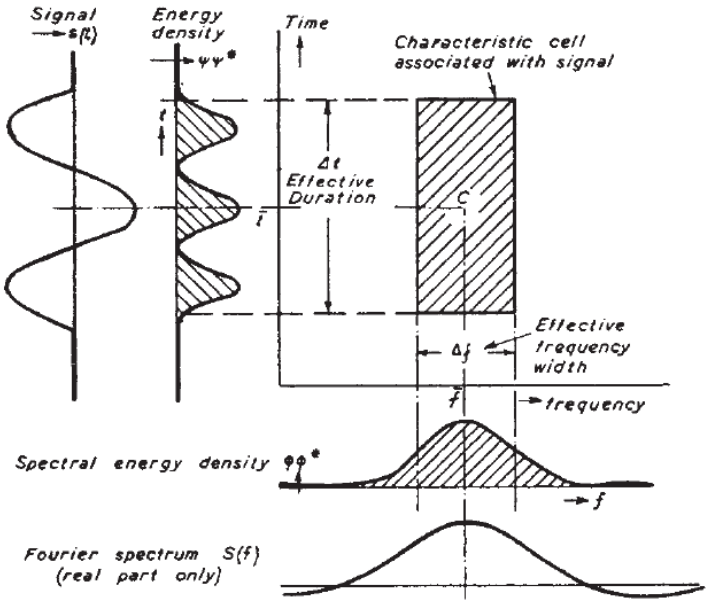


Figure 5.1.1. Original figure of the characteristic cell of an elementary particle in the information diagram from Gabor's paper [17].

Given a certain range of frequencies on the x-axis, there exists an effective time on the y-axis needed in order to perceive frequencies in that range. We can all verify this from experience.

Frequencies from a wide range will sound the same if we only hear them for a short time. However, given a longer period of time to listen, the ear is able to pick out precise frequencies. We can see this illustrated in the following figure, where within a short enough window, frequencies ranging from f to $4f$ all look about the same.

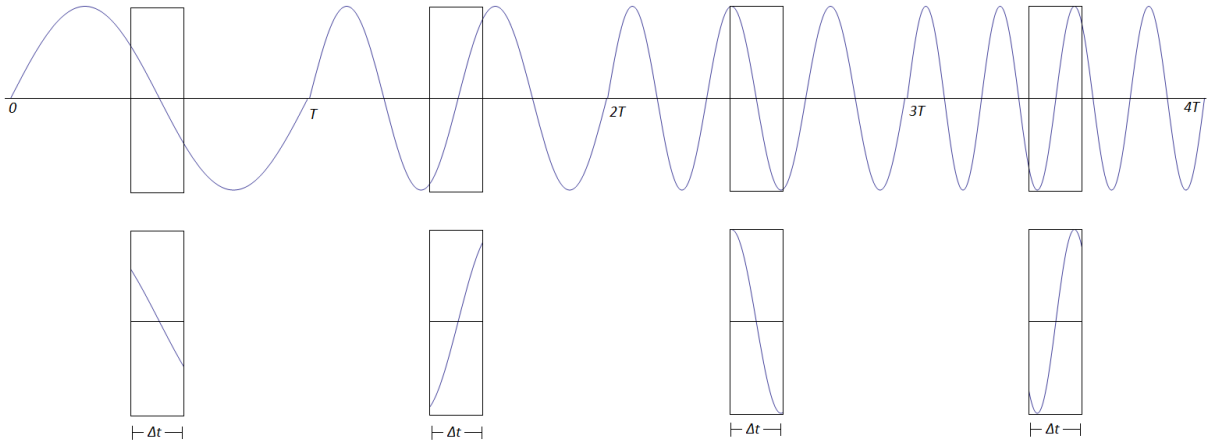


Figure 5.1.2. Small Δt results in a large Δf.

Gabor writes that when we plot these frequency ranges vs. effective duration, that the area of the resultant rectangle is at least 1, in other words, $\Delta t \Delta f \geq 1$. Below we see the cell associated with the property shown above.

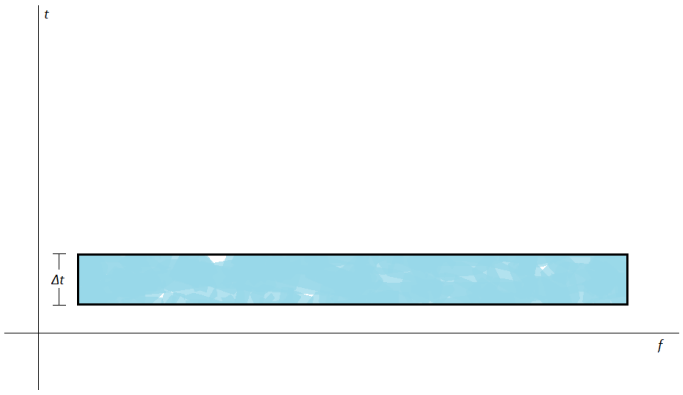


Figure 5.1.3. Characteristic cell associated with figure 5.1.2.

We call a signal whose characteristic cell has area 1, in other words $\Delta t \Delta f = 1$, elementary particles. Gabor gives a formula for an elementary particle $s(t)$, where

$$s(t) = e^{-\alpha^2(t-t_0)^2} e^{i2\pi f_0 t}$$

and the Fourier Transform of $s(t)$ is

$$S(f) = e^{-(\frac{\pi}{\alpha})^2(f-f_0)^2} e^{i2\pi t_0 f}$$

for a real constant α where $\Delta t = \frac{\sqrt{\pi}}{\alpha}$ and $\Delta f = \frac{\alpha}{\sqrt{\pi}}$. These elementary signals are illustrated in [17] with the following figure,

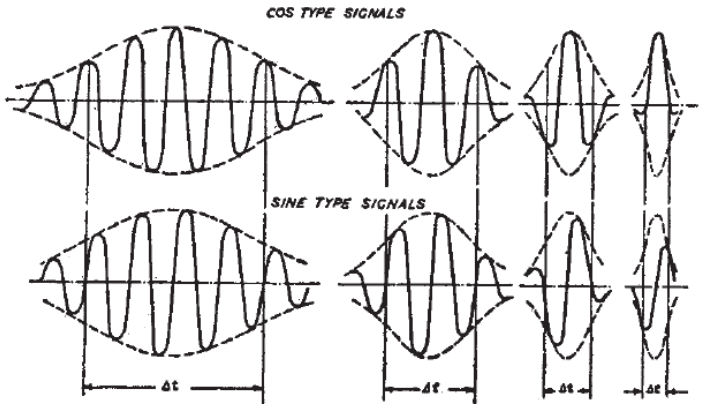


Figure 5.1.4. Original figure of a characteristic signal in the time domain from Gabor's paper [17].

We see that the $\lim_{\alpha \to 0}$ is a simple harmonic oscillation, and that the $\lim_{\alpha \to \infty}$ is the delta function. Thus for $\alpha = 0$ we simply obtain the Fourier analysis, and for $\alpha = \infty$ we obtain a series of delta functions, which is equal to $s(t)$ itself. However, for $0 < \alpha < \infty$, we obtain the Gabor Transform of a signal. We see that the Gabor Transform of a signal lies somewhere in between its time domain representation and its frequency domain representation. Like the Fourier Transform, the Gabor Transform is linear, therefore it is additive and homogeneous. Gabor's theory states that we can represent any arbitrary signal in terms of elementary particles. In frequency-time space we can plot these elementary particles, each with a complex amplitude factor $c_{i,k}$, to create a matrix-like object shown in the following figure

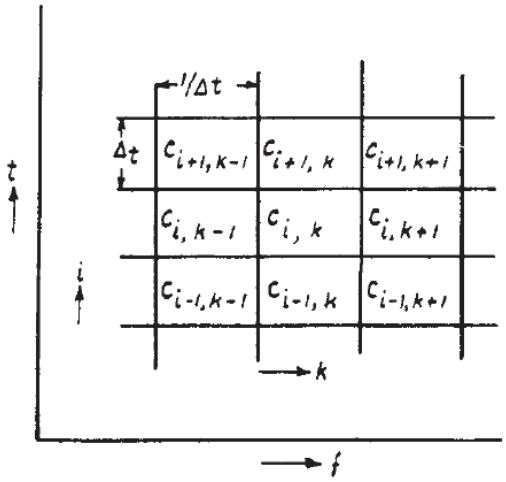


Figure 5.1.5. From [17]; matrix-like orientation of elementary particles used to represent a signal.

Gabor's original motivation for his theory was to save bandwidth for signal communication. If multiple frequencies are indistinguishable from each other to the human ear, as we saw with the short Δt window, then it is redundant to include all of them. Therefore bands of the spectrum which are perceptually identical for a given Δt are left empty except for one frequency within that band. Gabor writes that while this style of signal representation is not complete, it contains most of the subjectively important features [17]. Thus the Gabor Transform became the groundwork for Granular Synthesis.

5.1.2 Theory of Operation

Granular Synthesis algorithms, or Granulators, break a sample up into grains, typically between 10ms and 2000ms. There are two types of grains, *synthetic* and *sampled*. Synthetic grains synthesize their signal from a wavetable, while the sound for a sampled grain comes from a stored sample, which each grain reads from. Here we will focus on the latter form of Granular Synthesis, since that has been our focus so far. There are three main approaches to granular synthesis of a stored sound: *repetition*, *reordering*, and *interweaving* [8]. Repetition takes a sample and feeds it into a delay line with feedback, or repeats the read location in the sound file. Reordering changes the order in which individual grains appear in the original sample. This is usually done by speci-

fying a range for the granulator to read from, where grains are not chosen in chronological order within the range. Interweaving involves multiple samples which the granulator is reading from, and mixes the grain from each in a single output signal. Granular Synthesis is unique because it retains both frequency domain information (event timing, duration, envelope shape, waveform shape) and frequency domain information (short term spectrum of the waveform inside the grain, and the period of the waveform) [9]. The idea of granular synthesis is simple, however, due to the fact that a granulator can be processing thousands of grains a second, the process as a whole is not as trivial as it seems at the grain by grain level.

Earlier we looked at the Gabor Transform and how it became the basis for the idea of granular synthesis. However, we did not mention how Gabor's theory effects the time and frequency of the signal it approximates. We said that the Gabor Transformation was a linear transformation, in other words that for two time signals $s_0(t)$ and $s_1(t)$ and their respective Gabor Transforms $G_0(t, f)$ and $G_1(t, f)$, it is the case that

$$a \cdot s_0(t) + b \cdot s_1(t) = a \cdot G_0(t, f) + b \cdot G_1(t, f)$$

for $a, b \in \mathbb{R}$. Suppose we want to obtain a signal $s_2(t)$ by time stretching $s_1(t)$ by a factor of 2, but retain the frequency content of the original signal. We know that the Gabor Transform, $G_1(t, f)$, of $s_1(t)$ contains all of the perceptually important information need to construct $s_1(t)$. Therefore we can speculate that some linear combination of time shifted G_1's, converted back into the time domain, might create a signal close to what we imagine $s_2(t)$ would be like. This clearly does not suffice as a rigorous mathematical proof. Gabor himself writes that "[e]xpansion into elementary functions is in general a rather inconvenient mathematical process [17]." However, it does at least get us thinking about the Gabor Transform and its properties enough to perhaps try such an approach on our own, after which we find the result to quite resemble the $s_2(t)$ we hoped to create, at least on the level of raw human perception.

Granular Synthesis time modification is similar in some ways to how the Phase Vocoder modifies time. The same trick applies: the next grain processed is not necessarily the next grain in time. To start, the granulator chooses a point in the sample, say t_0, and then creates grains by counting forward the grain duration amount of time, say n, through the sample from that point to point t_n. Because the granulator processes the signal out of time, the next grain does not necessarily start at t_{n+1}. If time expansion is desired, the next grain would start at some point t_i where $0 < i < n$. If time compression is desired, then the next starting point t_i would lie later in the sample where $i > n$. However, if no time modification is desired, then simply $t_i = t_{n+1}$.

If we want to pitch-shift, then the rate at which we count forward through the sample from our starting point is adjusted accordingly, and is tied to the sampling rate of the sample, as well as the sampling rate of the DSP software, as we saw when looking at $pfft\sim$ and $block\sim$. Like in the Phase Vocoder, we simply count through the sample faster or slower than it was sampled. Again, analogous to playing a record faster or slower.

5.1.3 Characteristics

Sound produced through granular synthesis is often described as "organic". Ross Bencina writes,"The metaphor of *sonic clouds* has been used to describe sounds generated using Granular Synthesis ... gestures evocative of accumulation / dispersal, and condensation / evaporation may be created [8]." In [8], John Strawn takes this analogy so far as to specify types of granular synthesis sonics clouds as cumulus or stratus! Again, the theory behind granular synthesis only requires that the result satisfy some degree of human perception.

One of the clear advantages of Granular Synthesis is the absence of the digital artifacts that distract from acoustic samples which are the plague of Frequency domain approaches such as the Phase Vocoder. In return, we acquire a whole new set of sonic characteristics not present in our original sample. The tradeoff is that these characteristics are controllable by parameters which dictate how the granular synthesis operates. Now the modification to the original sample

is intentional and can be artistic, rather than distracting, as we see in the case of the Phase Vocoder. Next we will look at parameters in a Granulator which contribute and can be adjusted to change the overall output sound.

5.1.4 Parameters

The main parameters in a Granulator are the Grain Density, Grain Duration, Grain Amplitude, Envelope Shape, Panning, and Pitch. For most of these parameters, there are two ways which we can decide their value: fixed/intervalic, or stochastic/random within a range. Whether a parameter is decided randomly or not has a large effect on the overall sound of the granulator. There are sometimes thousands of grains being synthesized in a second, and consequently it is impossible to specify the parameters individually for each grain. Because of this, the processes of grain generation, parameter setting, and synthesis are often unified as one process, where parameters are adjusted on the global level [8].

The *Grain Density* parameter has perhaps the largest effect on the overall sound of Granular Synthesis. Grain Density dictates the number of grains per unit of time, usually per second. Grain generation, which consequently results in grain density, has two different approaches. There exists stochastic grain onset, and periodic grain onset. A stochastic grain onset algorithm uses random numbers to generate grains, while periodic grain onset generates grains at a regular interval. In either case, the less dense the grain density, the more each grain is recognized by the listener. Consequently, the effect of the envelope on the grain is stronger the lower the density, because there are less grains to obscure the envelope's shape. We can think about seeing the shadows of people walking behind a screen. The fewer people there are passing by at once, the easier it is to make out each individual. Inversely, the higher the grain density, the less distinguishable each grain's envelope is. Once again, if there are more shadows occupying the same space, each shadow becomes harder to distinguish from the others. Therefore, the envelope shape parameter is more important, the lower the grain density parameter is.

The grain density is analogous to the hop factor of overlap adding instances of FFT. Similar to how the grain envelope's role in the overall output of granular synthesis depends on the grain density; the choice of the FFT windowing function becomes more important the smaller the hop factor is. This is because the larger the hop factor is, the more we hear sound from under the peak of the windowing function. And if the hop factor is low, more of the sound in our output is coming from the edges of the window, which is where unwanted time domain discontinuities lie.

Grain Duration specifies how long each grain is, usually in milliseconds. Depending on the length of the grain, it will be easier or harder to recognize the original sample. The longer each grain lasts, the easier it will be to hear recognize the sample it comes from. Inversely, the shorter the grain is, the harder it will be to identify. This is the inverse relationship between Δt and Δf we saw in the Gabor Transform. Because of this, shorter grain sizes are often used to produce a certain effect, rather than to represent the original sample accurately. Strawn writes that when the grain duration is smaller, they create "crackling and popping textures [8]."

Grain Duration is analogous to the spectral frame size used in the FFT. Here we can also see the effect Gabor's observation has on the FFT. The smaller the frame size, in other words the smaller Δt is, the more similar different frequencies look to the FFT's perception of a signal. As we stated earlier, the reconstructed signal from an FFT with a small frame size, after spectral modifications have taken place, is less accurate than the same signal from an FFT with a larger frame size, which is what we have seen to also be the case with granular synthesis.

Envelope Shape plays a large role in the sonic characteristic of each grain, and consequently the sound of the granulator as a whole. Envelopes such as the Pulse Envelope, shown below, create "resonant grains that sound like woodblock taps in sparse textures when the grain duration is less than 100ms [8]." The effect of the envelope changes drastically based on the grain duration. In general, when the grain duration is small enough (about 50ms or so), envelopes

start to look and act like impulses, since they are shrunk in time. Because they are shrunk so much in time, envelopes which did have smoother transitions now have sharp edges, and these edges have large spectral effects, albeit for only a short amount of time.

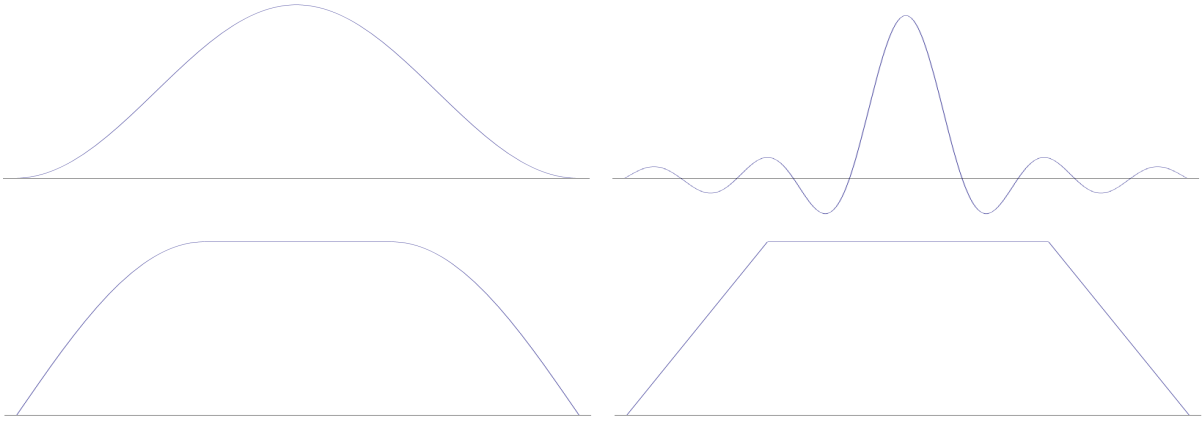


Figure 5.1.6. *top left* Quasi-Gaussian grain envelope. *bottom left* Same as top but with a longer top. *top right* Pulse envelope. *bottom right* Trapezoid Envelope.

Panning, Delay, and *Feedback* are all parameters controllable by the user. Like most granulator parameters, they can either be set stochastically or to be fixed. Of course limits can be set on the stochastic decision too. Adjusting these parameters have the effect of making a more coherent output sound, albeit at the cost of holding true to the original sample.

The *Pitch* of a grain can be changed by simply reading through the sample at a faster or slower rate, resulting in a higher or lower pitch, respectively. Obviously this will change the amount of information from the original sample that is in the grain. So if we want to keep the original amount of information from the original sample that is in each grain, then the grain duration must be adjusted according to the pitch shifting factor. For example, if we want to pitch-shift by a factor of 2, but want still want the grain to represent 1000ms of information from the original sample, then the grain duration should be set to 500ms. If the grain duration is left untouched, then the sample will represent information from 2000ms, over a grain duration of 1000ms.

5.2 The Grain Vocoder

After having thought about elements of the FFT in a different way through granular synthesis, it is a fun exercise to come back to the FFT and Phase Vocoder, and treat them as a Granulator of sorts.

This approach mainly affects the overall amplitude envelope for each frequency. For the normal Phase Vocoder, by using the amplitude information of the current frame we are reading, which is dictated by the time scale modification factor, the long term amplitude envelopes for each frequency are scaled to to exactly resemble the envelope in the original signal. However, by introducing variance in the read location of our sample, this is no longer the case. While the amplitude envelope for each frequency of the output will have a similar general profile to the envelope in the original signal, the two car vary drastically if we compare smaller sections of the two envelopes.

One result of this is a more organic sound when the speed is set to 0. In the normal Phase Vocoder, when the speed is set to zero, in order to freeze a certain sound, the output is not very pleasant. This is because the amplitude for each frequency band is held at a constant. However, given a sample whose amplitude envelope per frequency changes smoothly, by adding a bit of variance to where we are reading from in the sample, the output also changes smoothly. The effect is a sort of continuous splattering, rather than a jammed machine.

5.3 The Saphe Covoder

As we pointed out in section 4.3.1, we want to avoid intentional spectral peaks being phase updated as if they were discrete spectral noise. In that example, we looked at a low tone and its harmonics. There is also the case of harmonics of a single voice in the signal lying close to the fundamental or harmonics of another single voice in the signal, and the phase of one or more of those peaks being incorrectly updated if it lies in the region of influence of another, louder

peak. We will say such a signal exhibits *Spectral Crowding*. In cases of high spectral crowding due to multiple harmonic sources, it would be extremely helpful to be able to separate each sound source, phase update each individually, and then recombine the results to form a new time modified signal. A similar approach has been taken by [15], where signals are searched for harmonic elements and percussive elements. Elements of each type are grouped and separated, treated independently, and rejoined in the output signal. Here I will propose a similar approach, however, instead of flagging sources as either harmonic of rhythmic, we will go the step further and separate each harmonic source from the others entirely.

5.3.1 Cepstrum Analysis

Cepstrum Analysis performs the aforementioned operation: separating distinct sound sources within a signal. This is possible due to the periodic structure of musical harmonics. Most musical instruments have a harmonic structure for a fundamental f_0 of the form $f_0, 2f_0, 3f_0, 4f_0, ...$ Others, such as the clarinet, only have the odd multiples of the fundamental f_0, these are $f_0, 3f_0, 5f_0, 7f_0, ...$ Either way, these frequencies are periodic in the spectrum. So far we have seen the Fourier Transform group periodicity in the time domain, into one point in the frequency domain. By performing a Fourier Transform on the spectrum of a time signal, we are able to group periodic structures in the spectrum, into one point in the Cepstrum.

Consider the following spectrum of a sawtooth wave $f_1(t)$ with a frequency $\frac{128\pi}{T}$ for a time period T seconds.

Figure 5.3.1. Spectrum of a $\frac{128\pi}{T}$ radians per second sawtooth wave.

Now suppose we performed a Fourier Transform on this amplitude spectrum. The Fourier Transform would compare this spectrum to a set of sinusoids just like it we have seen it do for time signals throughout this project. This is illustrated in the following figure,

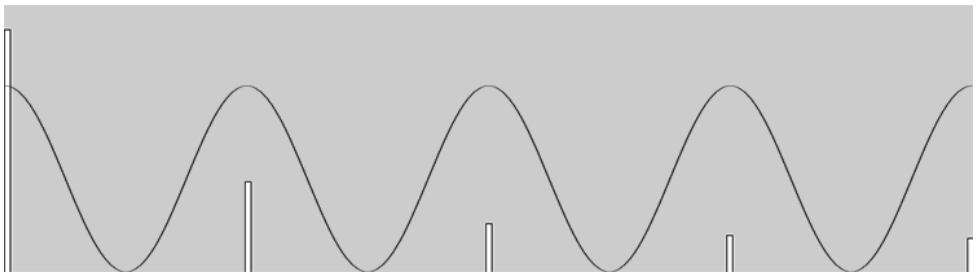


Figure 5.3.2. Comparison of the spectrum from Figure 5.3.1. to a cosine wave.

and would produce a graph similar to the following figure.

Figure 5.3.3.

This is the idea of Cepstrum Analysis.

Cepstrum Analysis was first proposed by Bogert, Healy, and Tukey in [19] as a method for analyzing echoes in order to study seismic activity. Since then Cepstrum analysis continues to be widely used in earth sciences, as well as analysis of machinery and speech processing. From [18], we get the following equations for calculating the Cepstrum.

$$c(\tau) = |\mathcal{F}\big[log\big(F_{xx}(f)\big)\big]|^2$$

where

$$F_{xx}(f) = |\mathcal{F}[f_x(t)]|^2$$

is the power spectrum of a time signal $f_x(t)$. [18] also gives us the following definition for the power cepstrum

$$c_p(\tau) = \mathcal{F}^{-1}\big[log\big(F_{xx}(f)\big)\big]$$

These formulas work well for pure analysis purposes. However, if reconstruction back into the time domain is desired, the Complex Cepstrum must be used. The Complex Cepstrum is calculated with the following formula.

$$c_c(\tau) = \mathcal{F}^{-1}[log\big(F_x(f)\big)]$$

$F_x(f)$ is the complex spectrum of $f_x(t)$. Therefore we calculate the complex logarithm as follows

$$log\big(F_x(f)\big) = log\big(|F_x(f)|\big) + i\angle F_x(f)$$

Because Cepstrum analysis uses the Fourier Transform, it has a similar vocabulary, however, the terms mean slightly different things in the time domain than they mean for a single Fourier Transform. The peak in Figure 5.3.3. is at a *quefrency* bin (from "frequency") and is called a *rahmonic*, since it represents a structure of harmonics. Specifically we are looking at the *gamnitude* (from "amplitude"). Since this is a polar representation, we also have a *saphe* (from "phase"). We can perform the same operations in the cepstrum that we could in the spectrum. Cepstral filtering is called *liftering*. There is both *long-pass* (from "high-pass") and *short-pass* (from "low-pass"). You go so far as to find mention of *alanysis* (from "analysis") and *repiod* (from "period"). However, these latter two are less common than the others. The title of this section follows a similar linguistic pattern as already exists in the field.

Consider the spectrum of a signal $f(t) = f_1(t) \times f_2(t)$ where $f_1(t)$ is our sawtooth wave of frequency $\frac{128\pi}{T}$ radians per second from earlier, and $f_2(t)$ is a sawtooth wave of frequency $\frac{100\pi}{T}$ radians per second.

The Fourier Transform will analyze it in the following way

And produce the following Cepstrum

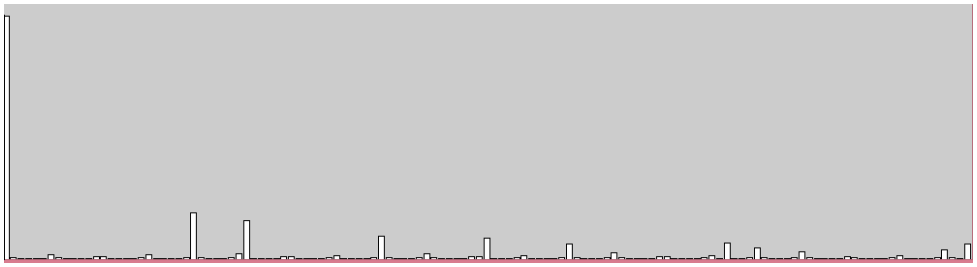


Figure 5.3.4. Spectrum of $f(t) = f_1(t) \times f_2(t)$.

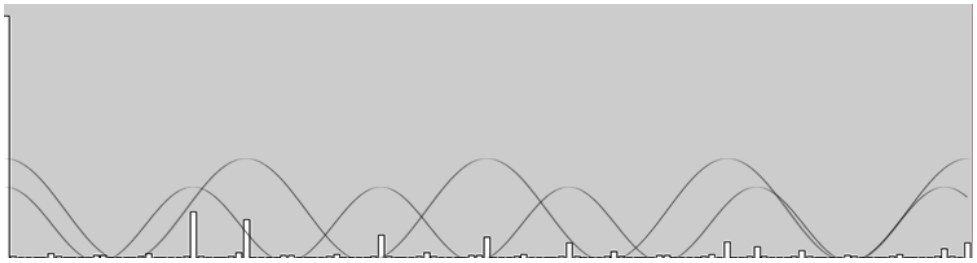


Figure 5.3.5. Fourier Analysis Figure 5.3.4.

Figure 5.3.6. Cepstrum of the signal $f(t) = f_1(t) \times f_2(t)$.

We can see that there is a peak corresponding to each harmonic source., and that frequency and quefrency are inversely proportional. High frequencies correspond to low quefrencies, and low frequencies correspond to high quefrencies. This is because the harmonics of a higher frequency are spaced further apart than the harmonics of a lower frequency. As we stated from the onset of this section, we want to separate these harmonic sources. In order to separate the two peaks, we create two new cepstra, each one with only one of the two peaks, where the other peak has been liftered. We then perform an Inverse Fourier Transform in order to convert back into the frequency domain where normal spectral manipulation can resume. We are able to simply subtract a quefrency from the cepstrum because we are taking the logarithm of our spectrum.

Suppose we a have some signal $f_1(t)$ where

$$f_1(t) = f_0(t) \cdot h(t)$$

for two signal $f_0(t)$ and $h(t)$. Then we know that

$$\mathcal{F}[f_1(t)] = \mathcal{F}[f_0(t)] \cdot \mathcal{F}(h(t)]$$

If we take the logarithm of both sides of the equation we see that

$$log\big(\mathcal{F}[f_1(t)]\big) = log\big(\mathcal{F}[f_0(t)] \cdot \mathcal{F}[h(t)]\big) = log\big(\mathcal{F}[f_0(t)]\big) + log\big(\mathcal{F}[h(t)]\big)$$

Either of the *log*s can be removed. This is our liftering step. Then we can re-obtain the spectrum from the Inverse Fourier Transform of e to the power of the liftered cepstrum. This process of separating harmonic sources is called *deconvolution*. Several of the aforementioned applications of Cepstrum Analysis use deconvolution.

5.3.2 Theory of Operation

Here I propose a method for time modification of musical signal using Identity Phase Locking in the Phase Vocoder, and the fact that there likely exists harmonic distortion of sound sources as a result of spectral crowding. Following the pattern of coining terms, it is called the Saphe Covoder. The following diagram outlines the conceptual steps for performing saphe covoding.

There are some obvious limitations to this approach and problems that need solutions. We notice that the number of FFT/IFFT pairs need to perform the Phase Vocoder algorithm varies depending on how many sound sources there are. Ideally you would always be able to process harmonic sources individually, however this requires that we first know how many sources there are, which limits our ability to operate in real-time. One possible solution is to set a maximum number of sources the Saphe Covoder is able to process, say q, where the Saphe Covoder is running $q + 1$ FFT/IFFT instances. If a signal had more than q harmonic sources, say p, the first q FFT/IFFT instances would process the first q harmonic sources, and the remaining

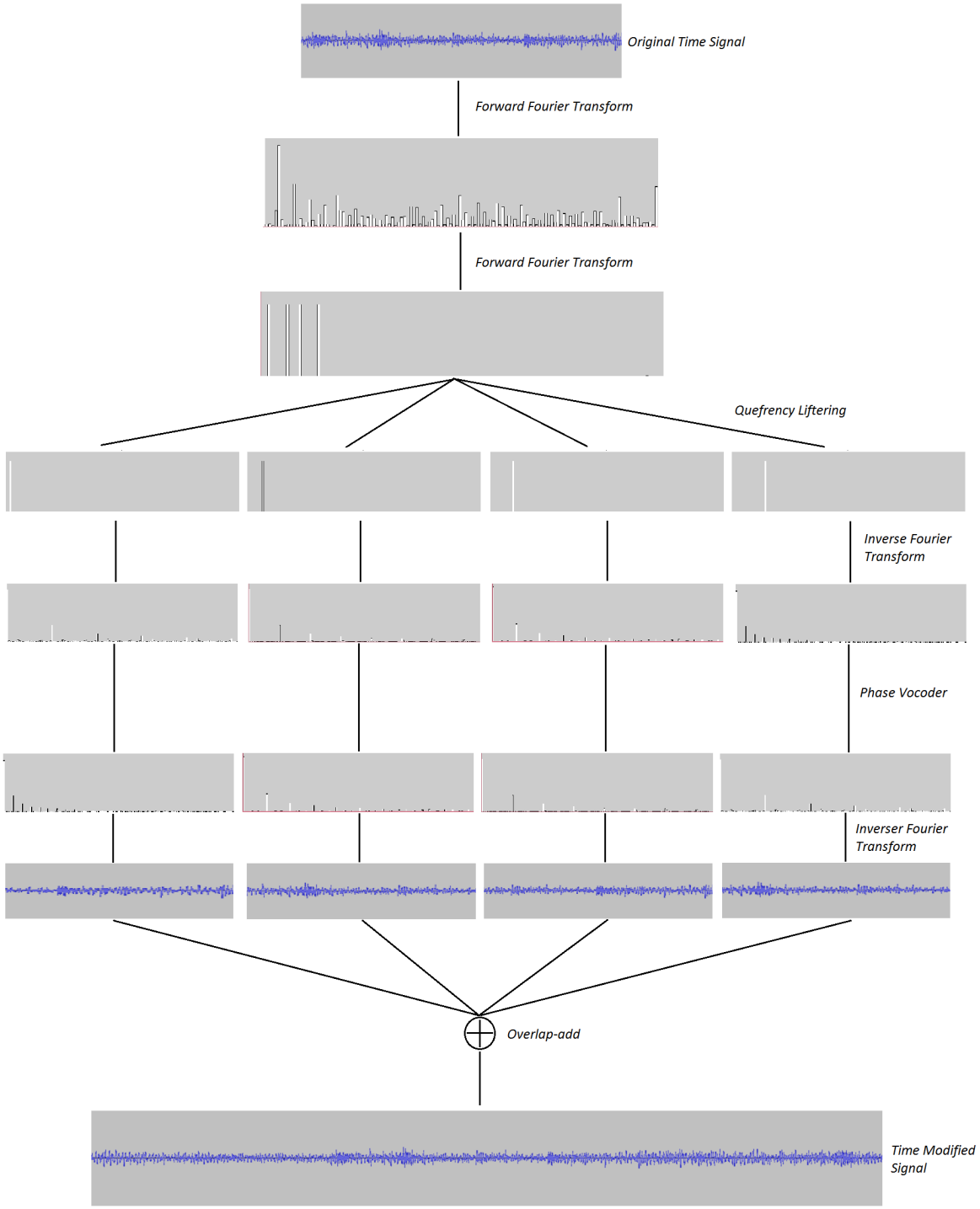


Figure 5.3.7. Saphe Covoder Operation Diagram

$p-q$ peaks would be processed by a dump FFT/IFFT instance, just like a normal Phase Vocoder.

Justification for such an approach could be that for signals with many harmonic sources, the timbre of each individual instrument is less distinguishable. The result of this approach would be that the first q sources would be properly phase updated, and that spectrum of the other combined $p - q$ sources would be less spectrally dense than the spectrum of all the original q sources anyway, and thus exhibit less harmonic distortion than the classic Phase Vocoder. Furthermore, the harmonic distortion that it did exhibit would be far less noticeable, since there is more sound to mask to inaccuracies. This is similar to how we were thinking about the perception of envelope shape as it relates to grain density in granular synthesis.

The decision then becomes which harmonic sources to isolate, and which to group together. At first glance a good choice seems to be isolating the higher rahmonics, since they correspond to lower frequency fundamentals, which we have seen to exhibit the most problems in the classic phase vocoder; and grouping the lower rahmonics, which correspond to higher frequency fundamentals, since, as a result of the logarithmic distribution of musical information, the upper end of the spectrum is naturally less dense than the lower half. The dump spectrum of only the higher frequency sources would be far less crowded than the dump spectrum of only the lower frequency sources. We can see that the computational costs of such would rise proportional to q. For a Saphe Covoder which would process $q + 1$ peaks, and running at a hop factor of R, the total number of FFT/IFFT pairs running simultaneously would be $R(q + 1)$.

The Cepstrum has an interesting perceptual meaning. Like Cepstrum Analysis, our ears group together harmonics, and we interpret them as the timbre of an instrument. For example, earlier we mentioned the clarinet, and that it only produces odd harmonics. The absence of the even harmonics results in the warm tone of the clarinet, and distinguishes it from other instruments. The saxophone on the other hand creates harmonics at all integer multiples of the fundamental frequency. As a result, the saxophone has a much brighter timbre than that of the clarinet. Therefore, it seems like a Cepstrally conscious Phase Vocoder would create a more musical

output, as a result of less harmonic distortion because of spectral crowding, and that the timbre of time modified musical signals would stay intact. In this way, the Saphe Covoder seems to operate on the same level as the Gabor Transform: the level of human perception. While the cost of computing increases significantly, it seems that the computational considerations taken by DSP engineers of the 1990's and 2000's are far less relevant, and that as a result of faster more accurate computers, the opening of the 3^{rd} decade of the 21^{st} century should bring with it more musical time-modified signals than ever before.

Appendix A

Code

```
float[] samples, real, imag, mag, newsamps;
int num = 256;
float max;
int index = 0;

void setup() {
  size(500, 500);
  samples = new float[num];
  newsamps = new float[num];
  real = new float[num/2+1];
  imag = new float[num/2+1];
  mag = new float[num/2+1];

  for (int i = 0; i < samples.length; i++) {
    samples[i] = sin(100*PI*i/num);
  }
  newsamps = toFdomain(samples);
  //counts through each frequency domain index
  for (int i = 0; i < num/2+1; i++) {
    //counts through each time domain sample
    for (int j = 0; j < num; j++) {
      real[i] = real[i]+samples[j]*cos(2*PI*i*j/num);
      imag[i] = imag[i]-samples[j]*sin(2*PI*i*j/num);
    }
  }
  max = mag[0];
  for (int i = 0; i < num/2+1; i++) {
    mag[i] = sqrt(real[i]*real[i]+imag[i]*imag[i]);
    println(i, mag[i]);

    if (mag[i]>max) {
      max = mag[i];
      index = i;
    }
  }
  println(max);
  println(index);
}

void draw() {
  rectMode(CORNERS);
  for (int i = 0; i < mag.length; i++) {
    //auto scales drawing
    rect(i*width/mag.length, height, i*width/mag.length+width/mag.length-1, .5*height + .5*height*(1-mag[i]/max));
  }
```

Figure A.0.1. Java DFT Calculation

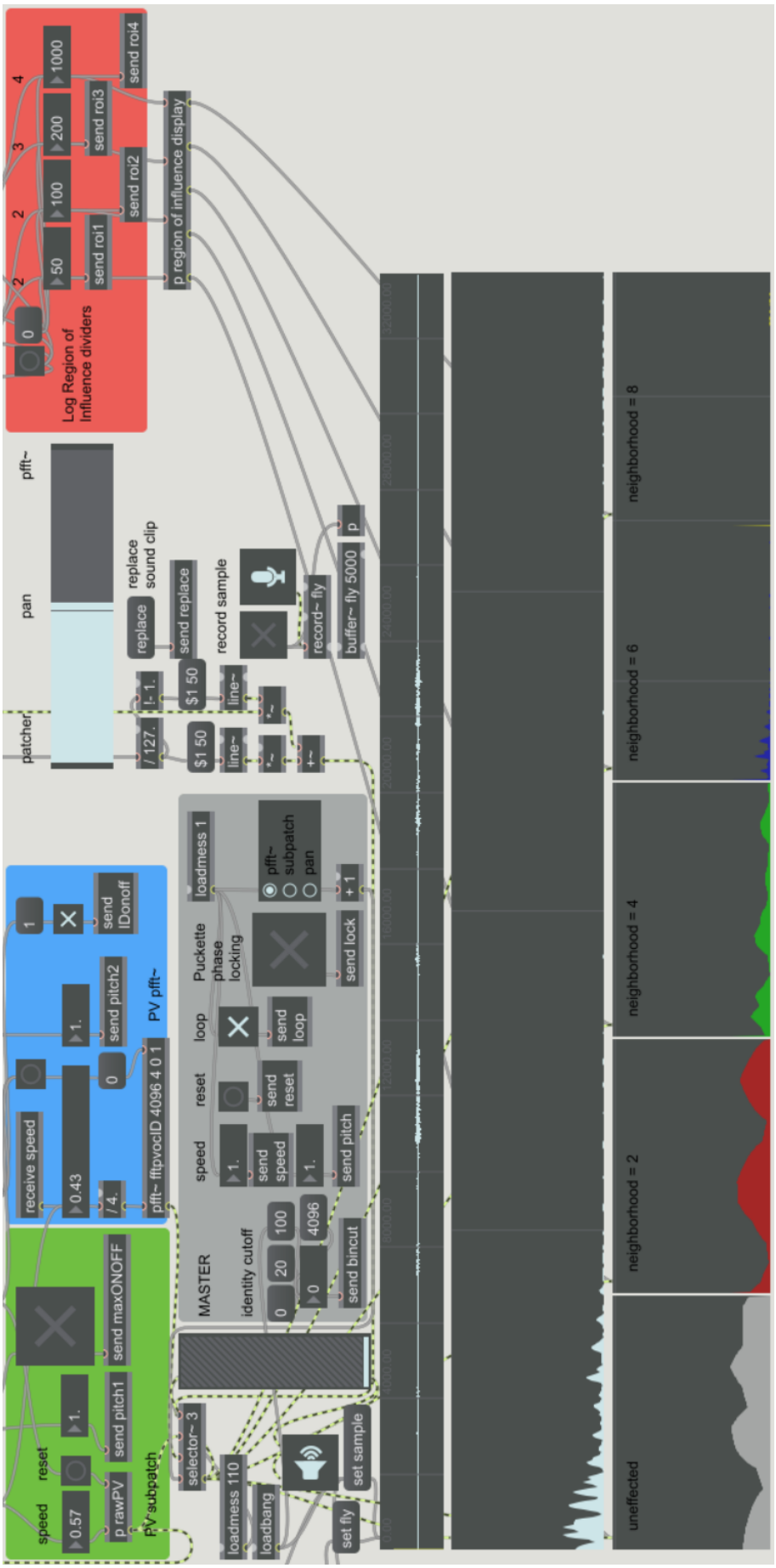


Figure A.0.2. Max/Msp Phase Vocoder Patch

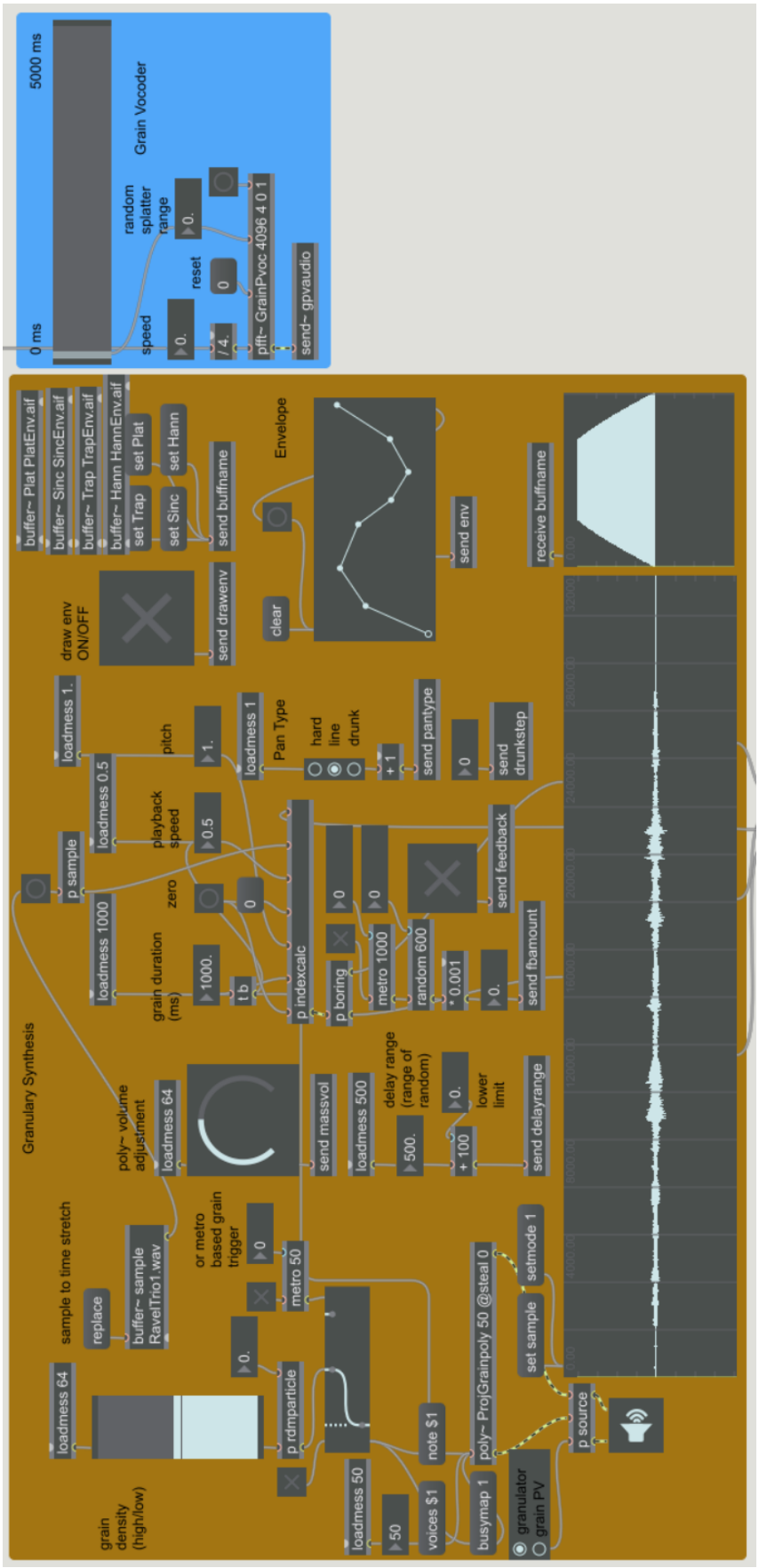


Figure A.0.3. Max/Msp Granulator Patch

Appendix B
Notation and Equations

B.0.3 Notation

Hop Factor	R
Frame Index	u
Frequency Index	k
Time Point	t_n
Sampling Period	T
Sampling Frequency	f_s
Discrete Fourier Analysis Time Instant	t_a^u
Discrete Inverse Fourier Synthesis Time	t_s^u
Instant	ω_{k_l}
FFT Analysis Point	$X[t_a^u, \omega_k]$
FFT Analysis Phase	$\angle X[t_a^u, \omega_k]$
IFFT Synthesis Point	$Y[t_s^u, \omega_k]$
IFFT Synthesis Phase	$\angle Y[t_s^u, \omega_k]$
Peak Channel	ω_{k_l}
Gabor Transform	$G(t, f)$

B.0.4 Equations

Fourier Transform	$\mathcal{F}(x(t)) = X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt$
Discrete Fourier Transform	$X[\omega_k] = \sum_{n=0}^{N-1} x[t_n]e^{-j\omega_k t_n}$
Inverse Fourier Transform	$\mathcal{F}^{-1}(X(\omega)) = x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}d\omega$
Discrete Inverse Fourier Transform	$x[t] = \sum_{n=0}^{N-1} X[\omega_k]e^{j\omega_k t_n}$
Magnitude	$A = Mag(X[\omega_k]) = \lvert X[\omega_k]\rvert = \sqrt{a^2+b^2}$
Phase	$\theta = \phi(a+bi) = arg(a+bi) = tan^{-1}(\frac{b}{a})$
Euler's Identity	$A \cdot e^{i\theta} = Acos(\theta) + Aisin(\theta)$
Sampling Period	$T = \frac{1}{f_s}$
Discrete Frequency Point	$\omega_k = k \cdot \frac{2\pi}{T}$
Hanning Window	$h[n] = .5[1 - cos(\frac{2\pi n}{N-1})]$
Hamming Window	$h[n] = .54 - .46cos\Big(\frac{2\pi n}{N-1}\Big)$
Blackman Window	$h[n] = .42 - .5cos\Big(\frac{2\pi n}{N-1}\Big) + 0.08cos\Big(\frac{4\pi n}{N-1}\Big),$
Constant Overlap Add Property	$\sum_{m=0}^{R-1} w(n - m\frac{N}{R}) = 1,\ \text{for } 0 \le n \le N-1$
Instantaneous Frequency	$\phi(\omega_k, t_s^u) = \phi_s(\omega_k, 0) + \int_0^{t_s^u} \omega_k\Big(\frac{\lambda}{\alpha}\Big)d\lambda$
Identity Phase Locking Equation	$\angle Y(t_s^u, \omega_k) =$
	$\angle Y(t_s^u, \omega_{k_l}) + \angle X(t_a^u, \omega_k) - \angle X(t_a^u, \omega_{k_l})$
Identity Phase Locking Phasor	$e^{i\theta}$ where $\theta = X[t_a^u, \omega_{k_l} - X[t_a^{u-1}, \omega_{k_l}$
Scaled Phase Locking Peak Phase Equation	$\angle Y[t_s^u, \omega_{k+1}] =$
	$\angle Y[t_s^{u-1}, \omega_k] + \angle X[t_a^u, \omega_{k+1}] - \angle X[t_a^{u-1}, \omega_k]$
Gabor Elementary Signal	$s(t) = e^{-\alpha^2(t-t_0)^2}e^{i2\pi f_0 t}$
Fourier Transform of Elementary Signal	$S(f) = e^{-(\frac{\pi}{\alpha})^2(f-f_0)^2}e^{i2\pi t_0 f}$
Power Spectrum	$F_{xx}(f) = \lvert\mathcal{F}[f_x(t)]\rvert^2$
Cepstrum	$c(\tau) = \lvert\mathcal{F}\big[log\big(F_{xx}(f)\big)\big]\rvert^2$
Power Cepstrum	$c_p(\tau) = \mathcal{F}^{-1}\big[log\big(F_{xx}(f)\big)\big]$
Complex Cepstrum	$c_c(\tau) = \mathcal{F}^{-1}[log\big(F_x(f)\big)]$
Complex Logarithm	$log\big(F_x(f)\big) = log\big(\lvert F_x(f)\rvert\big) + i\angle F_x(f)$

Bibliography

[1] Julius O. III Smith, *Mathematics of the Discrete Fourier Transform(DFT)*, BookSurge Publishing, Stanford CA, 2007.

[2] ———, *Spectral Audio Signal Processing*, W3K Publishing, Stanford CA, 2011.

[3] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, San Diego CA, 1997.

[4] Jean Laroche and Mark Dolson, *Improved Phase Vocoder Time-Scale Modification of Audio*, IEEE Transactions on Speech and Audio Processing **7** (1999), 323 – 332.

[5] Miller Puckette, *book*, San Diego CA, 30 December 2006.

[6] ———, *Phase-locked Vocoder*, IEEE (1995).

[7] Cort Lippe and Richard Dudas, *The Phase Vocoder - Part I*, Cycling '74 (02 November 2006).

[8] ———, *The Phase Vocoder - Part II*, Cycling '74 (02 July 2007).

[9] Ross Bencina, *Audio Anecdotes III*, A K Peters Wellesley, 31 August 2001.

[10] Curtis Roads, *The Computer Music Tutorial*, MIT Press, London, England, 1996.

[11] Daniel W. Griffin and Jae W. Lin, *Signal Estimation from Modified Short-Time Fourier Transform*, IEEE Trans. Acoust. Speech Signal Processing (April 1984).

[12] Thorsten Karrer, Eric Lee, and Jan Borchers, *PhaVoRIT: A Phase Vocoder for Real-Time Interactive Time-Stretching* (January 2006).

[13] But Why? Intuitive Mathematics, *The Fourier Transform*, `https://sites.google.com/site/butwhymath/fourier-analysis/the-fourier-transform`.

[14] James L. Flanagan and R. M. Golden, *Phase Vocoder*, IEEE Trans. Acoust. Speech Signal Processing (18 July 1966).

[15] Jonathan Driedger and Meinard Mller, *A Review of Time-Scale Modification of Music Signals*, Proceedings of the International Conference on Digital Audio Effects (February 2016).

[16] Dennis Gabor, *Theory of Communication* (24 September 1945).

[17] ———, *Acoustical Quanta and the Theory of Hearing*, Nature (May 3, 1947).

[18] R.B. Randall and J. Hee, *Cepstrum Analysis*, Brel & Kjaer Technical Review (August 1981).

[19] B.P. Bogert, M.J.R. Healy, and J.W. Tukey, *The Quefrency Alanysis of Time Series for Echoes: Cepstrum, Pseudo-Autocovariance, Cross-cepstrum and Saphe Cracking*, Proceedings of Symposium on Time Series Analysis (1963).