

Developing spectral processing applications

Victor Lazzarini,
MTL, NUI Maynooth, Ireland
1. May 2004

Abstract

Spectral processing techniques deal with frequency-domain representations of signals. This text will explore different methods and approaches of frequency-domain processing from basic principles. The discussion will be mostly non-mathematical, focusing on the practical aspects of each technique. However, wherever necessary, we will demonstrate the mathematical concepts and formulations that underline the process. This article is completed with an overview of the spectral processing classes in the Sound Object Library. Finally, A simple example is given to provide some insight into programming using the library.

1. The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is an analysis tool that is used to convert a time-domain digital signal into its frequency-domain representation. A complementary tool, the IDFT, does the inverse operation. In the process of transforming the spectrum, we start with a real-valued signal, composed of the waveform samples and we obtain a complex-valued signal, composed of the spectrum samples. Each pair of values (that make up a complex number) generated by the transform is representing a particular *frequency* point in the spectrum. Similarly, each single (real) number that composes the input signal represents a particular *time* point. The DFT is said to represent a signal at a particular time, as if it was a ‘snapshot’ of its frequency components.

One way of understanding how the DFT works its magic is by looking at its formula and trying to work out what it does:

$$DFT(x(n),k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) \times e^{-j2\pi kn/N} \quad k = 0,1,2,\dots,N-1 \quad (1)$$

The whole process is one of multiplying an input signal by complex exponentials and adding up the results to obtain a series of complex numbers that make up the spectral signal. The complex exponentials are nothing more than a series of

complex sinusoids, made up of cosine and sine parts:

$$e^{-j2\pi kn/N} = \cos(2\pi kn/N) - j \sin(2\pi kn/N) \quad (2)$$

The exponent $j2\pi kn/N$ determines the phase angle of the sinusoids, which in turn is related to its frequency. When $k=1$, we have a sinusoid with its phase angle varying as $2\pi n/N$. This will of course complete a whole cycle in N samples, so we can say its frequency is $1/N$ (to obtain a value in Hz, we just have to multiply it by the sampling rate). All other sinusoids are going to be whole-number multiples of that frequency, for $1 < k < N-1$. The number N is the number of points in the analysis, or the number of spectral samples (each one a complex number), also known as the *transform size*. Now we can see what is happening: for each particular frequency point k , we multiply the input signal by a sinusoid and then we sum all the values obtained (and scale the result by $1/N$).

Consider the simple case where the signal $x(n)$ is a sine wave with a frequency $1/N$, defined by the expression $\sin(2\pi n/N)$. The result of the DFT operation for the frequency point 1 is shown on fig.1. The complex sinusoid has detected a signal at that frequency and the DFT has output a complex value $[0, -0.5]$ for that spectral sample (the meaning of -0.5 will be explored later). This complex value is also called the *spectral coefficient* for frequency $1/N$. The real part of this number corresponds to the detected cosine phase component and its imaginary part relates to the sine phase component. If we slide the sinusoid to the next frequency point ($k=2$) we will obtain the spectral sample $[0, 0]$, which means that the DFT has not detected a sinusoid signal at the frequency $(2n/N)$.

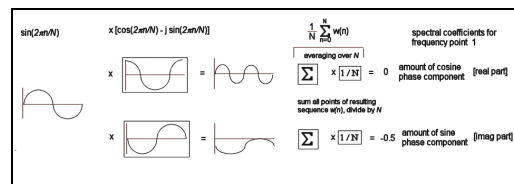


Figure 1. The DFT operation on frequency point 1, showing how a complex sinusoid is used to detect the sine and cosine phase components of a signal.

This shows that the DFT uses the ‘sliding’ complex sinusoid as a detector of spectral components. When a frequency component in the signal matches the frequency of the sinusoid, we obtain a non-zero output. This is, in a nutshell, how the DFT. Nevertheless, this example shows

only the simplest analysis case. In any case, the frequency $1/N$ is a special one, known as the *fundamental frequency of analysis*. As mentioned above, the DFT will analyse a signal as composed of sinusoids at multiples of this frequency.

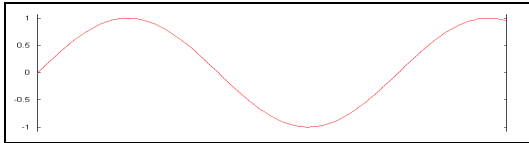


Figure 2. Plot of $\sin(2\pi 1.3n/N)$

Consider now a signal that does not contain components at any of these multiple frequencies. In this case, the DFT will simply analyse it in terms of the components it has at hand, namely the multiples of the fundamental frequency of analysis. For instance, take the case of a sine wave at $1.3/N$, $\sin(2\pi 1.3n/N)$ (fig.2). We can check the result of the DFT on table 1. The transform was performed using the C++ code above with $N=16$.

point (k)	real part (re[X(k)])	imaginary part (im[X(k)])
0	0.127	0.000
1	0.359	0.221
2	-0.151	0.127
3	-0.071	0.056
4	-0.053	0.034
5	-0.046	0.022
6	-0.042	0.013
7	-0.041	0.006
8	-0.040	0.000
9	-0.041	-0.006
10	-0.042	-0.013
11	-0.046	-0.022
12	-0.053	-0.034
13	-0.071	-0.056
14	-0.151	-0.127
15	0.359	0.221

Table 1. Spectral coefficients for a 16-point DFT of $\sin(2\pi 1.3n/N)$

Although confusing at first, this result is what we would expect, since we have tried to analyse a sine wave, which is 1.3 cycles long. We can, however, observe that one of the two largest pairs of absolute values is found on points 1. From what we saw in the first example, we might guess that the spectral peak is close to the frequency $1/N$, as in fact it is ($1.3/N$). Nevertheless, the result shows a large amount of spectral spread, contaminating all frequency points (see also fig.3). This has to do with the discontinuity between the last and first points of the waveform, something clearly seen on fig.2.

1.1. Reconstructing the time-domain signal

The result in the table above can be used to reconstruct the original waveform, by applying the inverse operation to the DFT, the Inverse Discrete Fourier Transform, defined as:

$$IDFT(X(k), n) = \sum_{k=0}^{N-1} X(k) \times e^{j2\pi kn/N} \quad n = 0, 1, 2, \dots, N-1 \quad (3)$$

In other words, the values of $X(k)$ are [complex] coefficients, which are used to multiply a set of complex sinusoids. These will be added together, point by point, to reconstruct the signal. This is, basically, a form of additive synthesis that uses complex signals. The coefficients are the amplitudes of the sinusoids (cosine and sine) and their frequencies are just multiples of the fundamental frequency of analysis. If we use the coefficients in the table above as input, we will obtain the original 1.3-cycle sine wave.

We saw above that point 1 refers to the frequency $1/N$, and point 2 to $2/N$ and so on. As mentioned before, the fundamental frequency of analysis in Hz will depend on how many samples are representing our signal in a second, namely, the sampling rate (SR). So our frequency points will be referring to kSR/N Hz, with $k=0, 1, 2, \dots, N-1$. So we will be able to quickly determine the frequencies for points 0 to $N/2$, ranging from the 0 Hz to $SR/2$, the Nyquist frequency, which is the highest possible frequency for a digital signal.

We can see in table 1 that points 9 to 15 basically have the same complex values as 7 to 1 (except for the sign of the imaginary part). It is reasonable to assume that they refer to the same frequencies. The sign of the imaginary parts indicates that they might refer to negative frequencies. This is because a negative frequency sine wave is the same as positive one with negative amplitude (or out-of-phase): $\sin(-x) = -\sin(x)$. In addition, $\cos(-x) = \cos(x)$, so the real parts are the same for negative and positive frequencies.

The conclusion is simple, the second half of the points refer to negative frequencies, from $-SR/2$ to $-SR/N$. It is essential to point out that the point $N/2$ refers to both $SR/2$ and $-SR/2$ (these two frequencies are indistinguishable). Also, it is important to note that the coefficients for 0 Hz and the Nyquist are always purely real (no imaginary part). We can see then that the output of the DFT then, splits the spectrum of a digital waveform in equally-spaced frequency points, or bands. The negative and positive spectral coefficients only differ in their imaginary part. For real signals, we can see that the negative side of the spectrum can

always be inferred from the positive side, so it is, in a way, redundant.

1.2. Rectangular and polar formats

In order to understand further the information provided by the DFT, we can convert the representation of the complex coefficients, from real/imaginary pairs to one that is more useful to us. The amplitude of a component is given by the magnitude of each complex spectral coefficient. The magnitude (or modulus) of a complex number z is:

$$|z| = \sqrt{re[z]^2 + im[z]^2} \quad (4)$$

As a real signal is always split into positive and negative frequencies, the amplitude of a point will be $\frac{1}{2}$ the 'true' value. The values obtained by the magnitude conversion are known as the *amplitude spectrum* of a signal. The amplitude spectrum of a real signal is always mirrored at 0 Hz.

The other conversion that complements the magnitude provides the phase angle (or offset) of the coefficient, in relation to a cosine wave. This yields the phase offset of a particular component, and it is obtained by the following relationship:

$$\theta(z) = \arctan \frac{im[z]}{re[z]} \quad (5)$$

The result of converting the DFT result in this way is called the phase spectrum. For real signals, it is always anti-symmetrical around 0 Hz. The process of obtaining the magnitude and phase spectrum of the DFT is called *cartesian-to-polar* conversion.

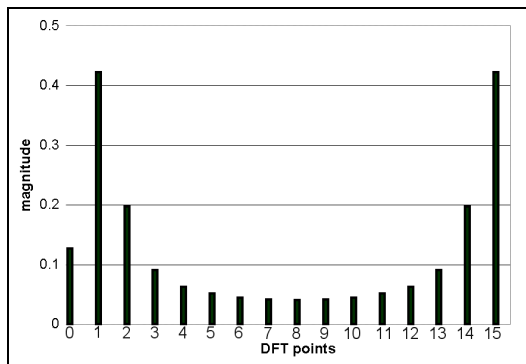


Figure 3. Magnitude spectrum from a 16-point DFT of $\sin(2\pi 1.3n/N)$.

We can see from fig.3 that the DFT results are not always clear. In fact unless the signal has all its components at multiples of the fundamental frequency of analysis, there will be a spectral

spread over all frequency points. In addition to these problems, the DFT in the present form, as a one-shot, *single-frame* transform, will not be able to track spectral changes. This is because it takes a single 'picture' of the spectrum of a waveform at a certain time. For a more thorough view of the DFT theory, please refer to (Jaffe, 1987a) and (Oppenheimer and Schaffer, 1975).

2. Applications of the DFT: Convolution

The single-frame DFT analysis as explored above has one important application, the convolution of time-domain signals through spectral multiplication. Before we proceed to explore this technique, it is important to note that the DFT is very seldom implemented in the direct form shown above. More usually, we will find optimised algorithms that will calculate the DFT much more efficiently. These are called the Fast Fourier Transform (FFT). Their result is in all aspects, equivalent to the DFT as described above. The only difference is in the way the calculation is performed. Also, because the FFT is based on specialised algorithms, they will only work with a certain number of points (N , the transform size). For instance, the standard FFT algorithm uses only power-of-two (2,4,..., 512, 1024...) sizes. From now on, when we refer to the DFT, we will imply the use of a fast algorithm for its computation.

Convolution is an operation with signals, just like multiplication or addition, defined as:

$$w(n) = y(n) * x(n) = \sum_{m=0}^n y(m)x(n-m) \quad (6)$$

One important aspect of time-domain convolution is that it is equivalent to the multiplication of spectra (and vice-versa). In other words, if $y(n)$ and $h(n)$ are two waveforms whose fourier transforms are $Y(k)$ and $H(k)$, then:

$$\begin{aligned} DFT[y(n) * h(n)] &= Y(k)H(k) \\ DFT[y(n)h(n)] &= Y(k) * H(k) \end{aligned} \quad (7)$$

This means that if the DFT is used to transform two signals into their spectral domain and the two spectra can be multiplied together, the result can be transformed back to the time-domain as the convolution of the two inputs. In this type of operation, we generally have an arbitrary sound that is convoluted with a shorter signal, called the

impulse response. The latter can be thought of as a mix of scaled and delayed unit sample functions and also as the list of the gain values in a tapped delay-line. The convolution operation will impose the spectral characteristics of this impulse signal into the other input signal. There are three basic applications for this technique:

- (1) Early reverberation: the impulse response is a train of pulses, which can be obtained by recording room reflections in reaction to a short sound.
- (2) Filtering: the impulse response is a series of FIR filter coefficients. Its amplitude spectrum determines the shape of the filter.
- (3) Cross-synthesis: the impulse response is an arbitrary sound, whose spectrum will be multiplied with the other sound. Their common features will be emphasized and the overall effect will be one of cross-synthesis.

Depending on the application, we might use a time-domain impulse response, whose transform is then used in the process. On other situations, we might start with a particular spectrum, which is directly used in the process. The advantage of this is that we can define the frequency-domain characteristics that we want to impose on the other sound.

2.1. A DFT-based convolution application

We can now look at the nuts and bolts of the application of the DFT in convolution. The first thing to consider is that, since we are using real signals, there is no reason to use a DFT that outputs both the positive and negative sides of the spectrum. We know that the negative side can be extracted from the positive, so we can use FFT algorithms that are optimised for the real signals. The discussion of specific aspects of these algorithms is beyond the scope of this text, but whenever we refer to the DFT, we will imply the use of a real input transform.

Basically, the central point of the implementation of convolution with the DFT is the use of the *overlap-add* method after the inverse transform. Since our impulse response will be of a certain length, this will determine the transform size (we will capture the whole signal in one DFT). The other input signal can be of arbitrary length, all we will need to do is to keep taking time slices of it that are the size of the impulse response. Now, because we know that the resulting length of the convolution of two signals is the sum of their lengths minus one, this will determine the minimum size of the transform (because the IDFT

output signal, as a result of the convolution, will have to be of that length).

The need for an *overlap-add* arises because, as the length of the convolution is larger than the original time-slice, we will need to make sure the tail part of it is mixed with the next output block. This will align the start of each output block with the original start of each time-slice in the input signal. So, if the impulse response size is S , we will slice the input signal in blocks of S samples. The convolution output size will be $2S - 1$, and the size of the transform will be the first power-of-two not less than that value.

The inputs to the transform will be padded to the required size. After the multiplication and the IDFT operation, we will have a block of $2S - 1$ samples containing the signal output (the zero padding will be discarded). All we need to do is to time-align it with the original signal, by overlapping the first $S - 1$ samples of this block with the last $S - 1$ samples of the previous output. The overlapping samples then are mixed together to form the final output. Fig.4 shows the input signal block sizes and the overlap-add operation.

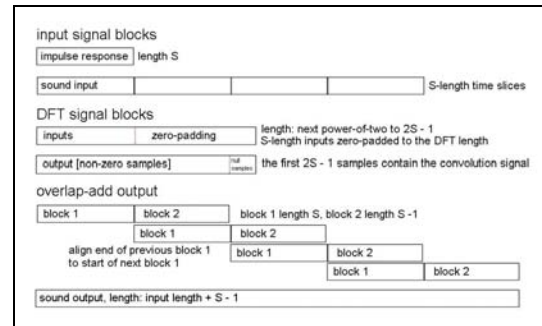


Figure 4. Convolution input and output block sizes and the overlap-add operation.

In terms of realtime applications, it is important to remark that there is an implicit delay in the DFT-based convolution. This is determined by the length of the impulse response. So with longer responses, a possible realtime use is somewhat compromised.

3. The Short-Time Fourier Transform

So far we have been using what we described as a single-frame DFT, one 'snapshot' of the spectrum at a specific time point. In order to track spectral changes, we will have to find a method of taking a sequence of transforms, at different points in time. In a way, this is similar to the process we used in the convolution application: we will be looking at extracting blocks of samples from a time-domain

signal and transform them with the DFT. This is known as the Short-Time Fourier Transform. The process of extracting the samples from a portion of the waveform is called *windowing*. In other words, we are applying a time window to the signal, outside which all samples are ignored.

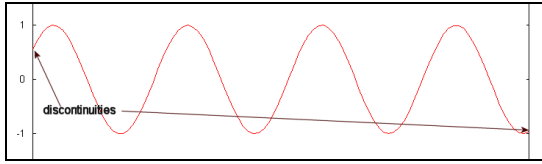


Figure 5. Rectangular window and discontinuities in the signal.

Time windows can have all sorts of shapes. The one we used in the convolution example is equivalent to a *rectangular* window, where all window contents are multiplied by one. This shape is not very useful in STFT analysis, because it can create discontinuities at the edges of the window (fig.5). This is the case when the analysed signal contains components that are not integer multiples of the fundamental frequency of analysis. These discontinuities are responsible for analysis artifacts, such as the ones observed in the above discussion of the DFT, which limit its usefulness.

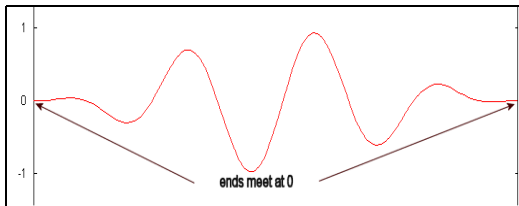


Figure 6. Windowed signal, where the ends tend towards 0.

Other shapes that tend towards 0 at the edges will be preferred. As the ends of the analysed segment meet, they eliminate any discontinuity (fig.6).

The effect of a window shape can be explained by remembering that, as seen before in (7), when we multiply two time-domain functions, the resulting spectrum will be the convolution of the two spectra. This is of course, the converse case of the convolution of two time-domain functions as seen in the section above. The effect of convolution in the amplitude spectrum can be better understood graphically. It is the shifting and scaling of the samples of one function by every sample of the other. When we use a window function in the DFT, we are multiplying the series of complex sinusoids that compose it by that function. Since this process results in spectral convolution, the resulting amplitude spectrum of the DFT after

windowing will be imposition of the spectral shape of a window function on every frequency point of the DFT of the original signal (fig.7). A similar effect will also be introduced in the phase spectrum.

When we choose a window that has an amplitude spectrum that have a peak at 0 Hz and dies away quickly to zero as the frequency rises, we will have a reasonable analysis result. This is case of the ideal shape in fig.7, where each analysis point will capture components around it and ignore spurious ones away from it.

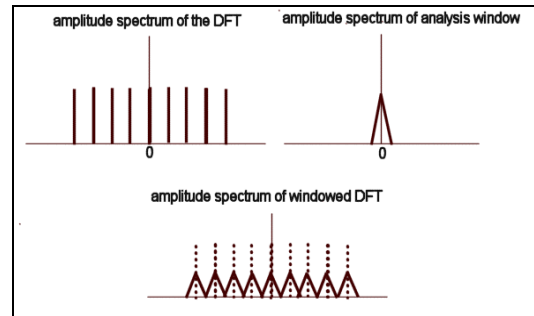


Figure 7. A simplified view of the amplitude spectrum of a windowed DFT as a convolution of the DFT sinusoids and the window function. The dotted lines mark the position of each DFT frequency point.

In practice, several windows with such low-pass characteristics exist. The simplest and more widely-used are the ones based on raised inverted cosine shapes, the Hamming and Hanning windows defined as:

$$w(n) = \alpha - (1 - \alpha) \cos\left(2\pi \frac{n}{N-1}\right) \quad 0 \leq n < N$$

where $\alpha = 0.5$ for Hanning and $\alpha = 0.54$ for Hamming (8)

In order to perform the STFT, we will apply a time-dependent window to the signal and take the DFT of the result. This will mean also that we are making the whole operation a function of time, as well as frequency. Here is a simple definition for the discrete STFT of an arbitrary-length waveform $x(n)$ at a time point t :

$$STFT(x(n), k, t) = \frac{1}{N} \sum_{n=-\infty}^{\infty} w(n-t)x(n)e^{-j2\pi kn/N} \quad k = 0, 1, 2, \dots, N-1 \quad (9)$$

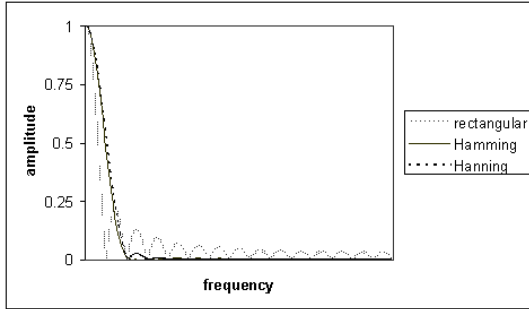


Figure 8. Spectral plots for rectangular, Hamming and Hanning windows (positive frequencies only).

The STFT provides a full spectral frame for each time point. In general, it is possible to take as little as four overlapped transforms (hopsize = $N/4$) to have a good resolution. Effectively, when using the STFT output for signal processing, the smallest hopsize will be determined by the type of window used ($N/4$ is the actual value for Hamming and Hanning windows).

The overlapped spectral frames can be transformed back into the time-domain by performing an inverse DFT on each signal frame. In order to smooth any possible discontinuities, we will also apply a window to each transformed signal block. The waveform is reconstituted by applying an overlap-add method that is very similar to the one employed in the convolution example.

4. Spectral Transformations: Manipulating STFT data

Each spectral frame can be considered as a collection of complex pairs relative to the information found on equally-spaced frequency bands at a particular time. They will contain information on the amplitude and frequency contents detected at that band. The rectangular, or cartesian, format that is the output of the transform packs these two aspects of the spectrum in the real and imaginary parts of each complex coefficient. In order to separate them, all we need to do is to convert the output into a polar representation. The magnitudes will give us the amplitude of each bin and the phases will be indicative of the detected frequencies. A single STFT frame can give us the amplitudes for each band, but we will not be able to obtain proper frequency values for them. This would imply extracting the *instantaneous frequency*, which is not really possible with one STFT measurement. Instead, the phases will contain the frequency information in a different form.

4.1. Cross-synthesis of frequencies and amplitudes

The first basic transformation that can be achieved in this way is *cross-synthesis*. There are different ways of crossing aspects of two spectra. The spectral multiplication made in the convolution example is one. By splitting amplitude and frequency aspects of spectra, we can also make that type of operation separately on each aspect. Another typical cross-synthesis technique is to combine the amplitudes of one sound with the frequencies of another. This is a spectral version of the time-domain *channel vocoder*. Once we have the STFT spectra of two sounds, there could not be an easier process to implement:

1. Convert the rectangular spectral samples into magnitudes and phases.
2. Take the magnitudes of one input and the phases of the other and make a new spectral frame, on a frame-by-frame basis.
3. Convert the magnitudes and phases to rectangular format. This is done with the following relationships:

$$\begin{aligned} re[z] &= Mag_z \times \cos(Pha_z) \text{ and} \\ im[z] &= Mag_z \times \sin(Pha_z) \end{aligned} \quad (10)$$

The resulting spectral frames in rectangular format can then be transformed back to the time-domain using the ISTFT and the overlap-add method.

4.2. Spectral-domain filtering

If we manipulate the magnitudes separately, we might also be able to create some filtering effects by applying a certain contour to the spectrum. For instance, to generate a simple low-pass filter we can use the $1/4$ of the shape of the cosine wave and apply it to all points from 0 to $N/2$. The function used for shaping the magnitude will look like this:

$$mag[k] = \cos\left(\frac{\pi k}{2N}\right) \quad 0 \leq k \leq N/2 \quad (11)$$

A high-pass filter could be designed by using a sine function instead of cosine in the example above. In fact, we can define any filter in spectral terms and use it by multiplying its spectrum with the STFT of any input sound. This leads us back into the convolution territory. Consider the typical 2-pole resonator filter design, whose transfer function is:

$$H(z) = \frac{A_0}{1 - 2R \cos \theta z^{-1} + R^2 z^{-2}} \quad (12)$$

Here, θ is the pole angle and R its radius (or magnitude), parameters that are related to the filter centre frequency and bandwidth, respectively. The scaling constant A_0 is used to scale the filter output so that it does not run wildly out-of-range. Now if we evaluate this function for evenly-spaced frequency points $z = e^{j2\pi k/N}$, we will reveal the discrete spectrum of that filter. All we need to do is to do a complex multiplication of the result with the STFT of an input sound.

The mathematical steps used to obtain the spectrum of the filter are based on Euler's relationship, which splits the complex sinusoidal $e^{j\omega}$ into its real and imaginary parts, $\cos(\omega)$ and $j\sin(\omega)$. Once we obtained the spectral points in the rectangular form $A_0(a + ib)^{-1}$, all we need is to multiply them with the STFT points of the original signal. This will in reality turn out to be a complex division:

$$Y[k] = (re[X[k]] + im[X[k]] \times A_0(re[F[k]] + im[F[k]])^{-1} = A_0 \left(\frac{re[X[k]] + im[X[k]]}{re[F[k]] + im[F[k]]} \right)$$

where $X[k]$, $A_0F[k]^{-1}$ and $Y[k]$ are the spectra of the input signal, filter and output, respectively

(13)

There are many more processes that can be devised for transforming the output of the STFT. The examples given here are only the start. They represent some classic approaches, but several other, more radical techniques can be explored.

5. Tracking the Frequency: the Phase Vocoder.

As we observed, although we can manipulate the frequency content of spectra, through their phases, the STFT does not have enough resolution to tell us what frequencies are present in a sound. We will have to find a way of tracking the instantaneous frequencies in each spectral band. A well-known technique known as the Phase Vocoder (PV) (Flanagan and Golden, 1966) can be employed to do just that.

The STFT followed by a polar conversion can also be seen as a bank of parallel filters. Its output is composed of the values for the magnitudes and phases at every time-point or hop period for each bin. The first step in transforming the STFT into a Phase Vocoder is to generate values that are proportional to the frequencies present in a sound. This is done ideally by the taking the time derivative of the phase, but we can approximate it by computing the difference between the phase

value of consecutive frames, for each spectral band. This simple operation, although not yielding the *right* value for the frequency at a spectral band, will output one that is proportional to it.

By keeping track of the phase differences, we can time-stretch or compress a sound, without altering its frequency content (in other words, its pitch). We can perform this by repeating or skipping spectral blocks, to stretch or compress the data. Because we are keeping the phase differences between the frames, when we accumulate them before resynthesis, we will reconstruct the signal back with the correct original phase values. We are keeping the same hop period between frames, but because we use the phase difference to calculate the next phase value, the phases will be kept intact, regardless of the frame readout speed.

One small programming point needs to be made in relation to the phase calculation. The inverse tangent function outputs the phase in the range of $-\pi$ to π . When the phase differences are calculated, they might exceed this range. In this case, we have to bring them down to the expected interval (known as *principal values*). This process is sometimes called phase unwrapping.

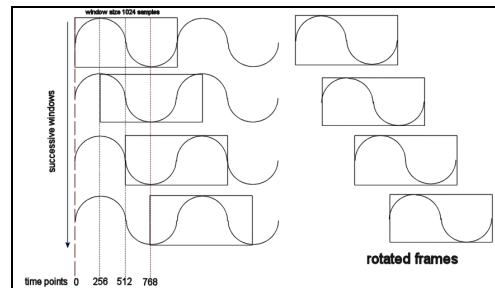


Figure 9. Signal frame rotation, according to input time point

5.1. Frequency estimation

So far we have been working with values that are proportional to the frequencies at each analysis band. In order to obtain the proper values in Hz, we will have to first modify the input to the STFT slightly. We will rotate the windowed samples inside the analysis frame, relative to the time point n (in samples and taken modulus N) of the input window. If our window has 1024 samples and we are hopping it every 256 samples, the moduli of the successive time-points n will be 0, 256, 512, 768, 0, 256.... The rotation will imply that for time point 256, we will move samples from positions 0 – 767 into positions 256 to 1023. The last 256 samples will be moved to the first locations of the

block. A similar process is applied to the other time points (fig. 9).

The mathematical reasons for this input rotation are somewhat complex, but the graphic representation shown on fig. 9 goes some way on helping us understand the process intuitively. As we can see the rotation process has the effect of aligning the phase of the signal in successive frames. This will help us obtain the right frequency values, but we will better understand it after seeing the rest of the process. In fact, the input rotation renders the STFT formulation mathematically correct, as we have been using a non-rigorous and simpler approach (which so far has worked for us).

After the rotation, we can take the DFT of the frame as usual and convert the result into polar form. The phase differences for each band are then calculated. This now tells us how much each detected frequency deviates from the centre frequency of its analysis band. The centre frequencies are basically the DFT analysis frequencies $2\pi k/N$, in radians. So, to obtain the proper detected frequency value, we only need add the phase differences to the centre frequency for each analysis band, scaled by the hopsize. The values in Hz can be obtained by multiplying the result, which is given in *radians per hopsize samples*, by $SR/[2\pi \times \text{hopsize}]$ (SR is, of course, the sampling rate in samples/sec).

Here is a summary of the steps involved in phase vocoder analysis:

1. Extract N samples from a signal and apply an analysis window. Rotate the samples in the signal frame according to input time $n \bmod N$.
2. Take the DFT of the signal.
3. Convert rectangular coefficients to polar format.
4. Compute the phase difference and bring the value to the $-\pi$ to $+\pi$ range.
5. Add the difference values to $2\pi kD/N$, and multiply the result by $SR/2\pi D$, where D is the hopsize in samples. For each spectral band, this result yields its frequency in Hz, and the magnitude value, its peak amplitude.

5.2. Phase vocoder resynthesis

Phase Vocoder data can be resynthesised using a variety of methods. Since we have blocks of amplitude and frequency data, we can use some sort of additive synthesis to playback the spectral frames. However, a more efficient way of converting to time-domain data for arbitrary sounds with many components is to use an

overlap-add method similar to the one in the ISTFT. All we need to do is retrace the steps taken in the forward transformation:

1. Convert the frequencies back to phase differences in radians per I samples by subtracting them from the centre frequencies of each channel, in Hz, kSR/N , and multiplying the result by $2\pi I/SR$, where I is the synthesis hopsize.
2. Accumulate them to compute the current phase values.
3. Perform a polar to rectangular conversion.
4. Take the IDFT of the signal frame.
5. Unrotate the samples and apply a window to the resulting sample block.
6. Overlap-add consecutive frames.

As a word of caution, it is important to point out that all DFT-based algorithms will have some limits in terms of partial tracking. The analysis will be able to resolve a maximum of one sinusoidal component per frequency band. If two or more partials fall within one band, the phase vocoder will fail to output the right values for the amplitudes and frequencies of each of them. Instead, we will have an amplitude-modulated composite output, in many ways similar to beat frequencies. In addition, because the DFT splits the spectrum in equal-sized bands, this problem will mostly affect lower frequencies, where bands are perceptually larger. However, we can say that in general, the phase vocoder is a powerful tool for transformation of arbitrary signals.

5.3. Spectral morphing

A typical transformation of PV data is spectral interpolation, or morphing. It is a more general version of the spectral cross-synthesis example discussed before. Here, we interpolate between the frequencies and amplitudes of two spectra, on a frame-by-frame basis.

Spectral morphing can produce very interesting results. However, its effectiveness depends very much on the spectral qualities of the two input sounds. When the spectral data does not overlap much, interpolating will sound more or less like cross-fading, which can be achieved in the time-domain for much less trouble. There are many more transformations that can be devised for modifying PV data. In fact, any number manipulation procedure that generates a spectral frame in the right format can be seen as a valid spectral process. Whether it will produce a musically useful output is another question. Understanding how the spectral data is generated

in analysis is the first step in designing transformations that work.

For a more detailed look into the theory of the Phase Vocoder, please refer to the James Flanagan's original article on the technique. Other descriptions of the technique are also found in (Dolson, 1986) and (Moore, 1990).

6. The Instantaneous Frequency Distribution

An alternative method of frequency estimation is given by the instantaneous frequency distribution (IFD) algorithm proposed by Toshihiko Abe (Abe et al, 1997). It uses some of the principles already seen in the phase vocoder, but its mathematical formulation is more complex. The basic idea, which is also present in the PV algorithm, is that the frequency, or more precisely, the instantaneous frequency detected at a certain band is the time derivative of the phase. Using Euler's relationship, we can define the output of the STFT in polar form. This is shown below, using $\omega = 2\pi k/N$:

$$STFT(x(n), k, t) = R(\omega, t) \times e^{j\theta(\omega, t)} \quad (14)$$

The phase detected by band k at time-point t is $\theta(2\pi kn/N, t)$ and the magnitude is $R(2\pi kn/N, t)$. The Instantaneous Frequency Distribution of $x(n)$ at time t is then the time derivative of the STFT phase output:

$$IFD(x(n), k, t) = \frac{\partial}{\partial t} \theta(\omega, t) \quad (15)$$

This can be intuitively understood as the measurement of the *rate of rotation* of the phase of a sinusoidal signal. In the phase vocoder, we estimated it by crudely taking the difference between phase values in successive frames. The IFD actually calculates the time derivative of the phase directly, from data corresponding to a single time-point:

$$IFD(x(n), k, t) = \omega + \text{imag} \left\{ \frac{DFT(x'_t(m), k)}{DFT(x_t(m), k)} \right\} \quad (16)$$

The mathematical steps involved in the derivation of the IFD are quite involved. However, if we want to implement the IFD, all we need is to employ its definition in terms of DFTs, as given

above. We can use the straight DFT of a windowed frame to obtain the amplitudes and use the IFD to estimate the frequencies. Also, because we are using the straight transform of a windowed signal, there is no need to rotate the input, as in the phase vocoder. If we look back at the DFT as defined in (19), we see that it, in fact, does not include the multiplication by a complex exponential (as does the STFT). Finally, the derivative of the analysis window is generated by computing the differences between its consecutive samples..

7. Tracking spectral components: Sinusoidal Modelling

Sinusoidal modelling techniques are based on the principles that we have held all along as the background to what we have done so far: that time-domain signals are composed of the sum of sinusoidal waves of different amplitudes, frequencies and phases. The number of sinusoidal components present in the spectrum will vary from sound to sound, and also can vary dynamically during the evolution of a single sound. As we have pointed out before, since we are still using STFT-based spectral analysis, at any point in time, the maximum resolution of components will depend on the size of each analysis band. Partial tracking will, therefore, not be suitable for spectrally dense sounds. However, there will be many musical signals that can be manipulated through this method.

7.1. Sinusoidal Analysis

The principle behind sinusoidal analysis is very simple, although its implementation is somewhat involved. Using the magnitudes from STFT analysis, we will identify the spectral peaks at the integral frequency points (STFT bins or bands). The identified peaks will have to be above a certain threshold, which will help separate the detected sinusoid components from transient spectral features. The exact peak position and amplitude can be estimated by using an interpolation procedure based on the magnitudes of the bins around the peaks. With the interpolated bin positions we can then find the exact values for the frequencies and phases obtained originally from the IFD/STFT input, again through interpolation. These will then, together with the amplitude, form a 'track', linked to each detected peak (fig. 11). However the track will only exist as such if there is some consistency in consecutive frames, ie. if there is some matching between peaks found at each time-point. When peaks are short-lived, they will not make a track.

Conversely, when a peak disappears, we will have to wait a few frames to declare the track as finished. So most of the process becomes one of track management, which accounts for the more involved aspects of the algorithm.

As seen in fig.10, the sinusoidal analysis will output tracks made up of frequencies, amplitudes and phases. This in turn can be used for additive re-synthesis of the signal or of portions of its spectrum. One typical method involves the use of the phase and frequency parameters to calculate the varying phase used to drive an oscillator. This uses the two parameters in a cubic interpolation, which is not only mathematically involved, but also computationally intensive. A simpler version can be formulated that would employ only the frequency and amplitudes interpolated linearly. This is simpler, more efficient and for many applications sufficiently precise. Also, since we do not require the phase parameter, we can simplify the analysis algorithm to calculate only frequencies and amplitudes for each track. This version could employ either the IFD (as shown in fig.12) or the Phase Vocoder, as discussed in the previous sections, to provide the magnitude and frequency inputs.

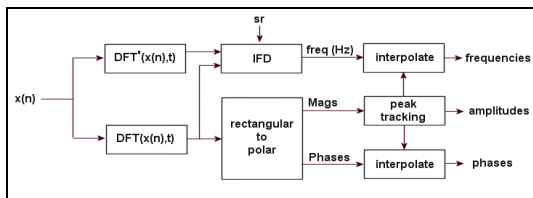


Figure 10. Sinusoidal analysis and track generation from a time-domain signal $x(n)$.

7.2. Additive resynthesis

The additive resynthesis procedure will take the track frames and use a frame-by-frame interpolation of the amplitudes and frequencies of each track to drive a bank of sinewave oscillators (fig.12). We will only need to be careful about using the track IDs to perform the interpolation between the frames. Also, when a track is created/destroyed, we will create an amplitude onset/decay so that we do not have discontinuities in the output signal. We will be using interpolating lookup oscillators, with a table size of 1024 points to generate the signal for each track. Each sine wave component will be then mixed into the output.

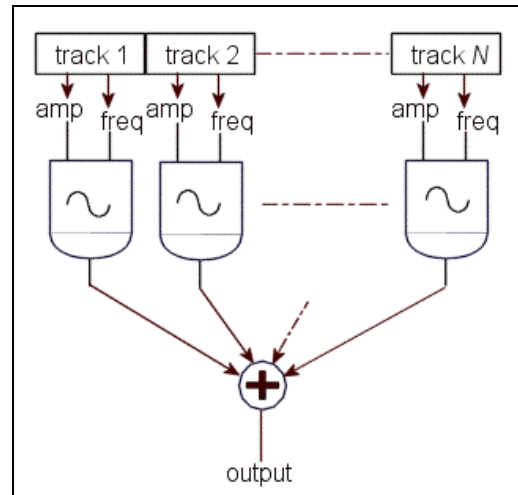


Figure 11. The additive synthesis process.

The main point about this type of analysis is that it is designed to track the sinusoidal components of a signal. Some sounds will be more suitable for this analysis than others. Distributed spectra will not be tracked very effectively, since its complexity will not suit the process. However, for certain sounds with both sinusoidal content and some more noise-like/transient elements, we could in theory obtain these 'residual' aspects of the sound by subtracting the resynthesised sound from the original. That way, we would be able to separate these two elements and perhaps process them individually. A more precise method of resynthesis, using the original phases is required for the residual extraction to work. This idea is developed in the Spectral Modelling Synthesis (SMS) technique (Serra, 1997).

8. Spectral processing with the Sound Object Library

The Sound Object (SndObj) library (Lazzarini, 2000) is a multi-platform music and audio processing C++ class library. Its 100-plus classes feature support for the most important time and frequency-domain processing algorithms, as well as basic soundfile, audio and MIDI IO services. The library is available for Linux, Irix, OS X and Windows; its core classes are fully portable to any system with ANSI C/C++ compilers.

The spectral processing suite of classes of the SndObj version 2.5.1 include the following:

(a) Analysis/Resynthesis:

- FFT** STFT analysis
- IFFT** ISTFT resynthesis
- PVA** Phase Vocoder Analysis
- PVS** Phase Vocoder Synthesis
- IFGram** IFD + Amps (and phases) analysis

SinAnal Sinusoidal track analysis
SinSyn Sinusoidal synthesis (cubic interpolation)
AdSyn Sinusoidal synthesis (linear interpolation)
Convolve FFT-based convolution

(b) Spectral Modifications

PVMorph PV data interpolation
SpecMult spectral product
SpecCart polar-rectangular conversion
SpecSplit/SpecCombine
 split/combine amps & phases
SpecInterp spectral interpolation
SpecPolar rectangular-polar conversion
SpecThresh thresholding
SpecVoc cross-synthesis

(c) Input/Output

PVRead variable-rate PVOCEX file readout
SpecIn spectral input
SndPVOCEX PVOCEX file IO
SndSinIO sinusoidal analysis file IO

In addition, the library provides a development framework for the addition of further processes. The processing capabilities can be fully extended by user-defined classes.

8.1. A programming example

The following example shows the use of the library for the development of a PD class for spectral morphing (Fig.12). Here we see how SndObj objects are set-up in the PD class constructor code:

```
void *morph_tilde_new(t_symbol *s, int
                    argc, t_atom *argv)
{
  (...)
  x->window = new HammingTable(1024, 0.5);
  x->inobj1 = new SndObj(0, DEF_VECSIZE,
                        sr);
  x->inobj2 = new SndObj(0, DEF_VECSIZE,
                        sr);
  x->spec1 = new PVA(x->window, x->inobj1,
                    1.f, DEF_FFTSIZE,
                    DEF_VECSIZE, sr);
  x->spec2 = new PVA(x->window, x->inobj2,
                    1.f, DEF_FFTSIZE,
                    DEF_VECSIZE, sr);
  x->morph = new PVMorph(morphfr, morpha,
                        x->spec1, x->spec2,
                        0,0,DEF_FFTSIZE,
                        sr);
  x->synth = new PVS(x->window, x->morph,
                    DEF_FFTSIZE,
                    DEF_VECSIZE, sr);
  (...)
}
```

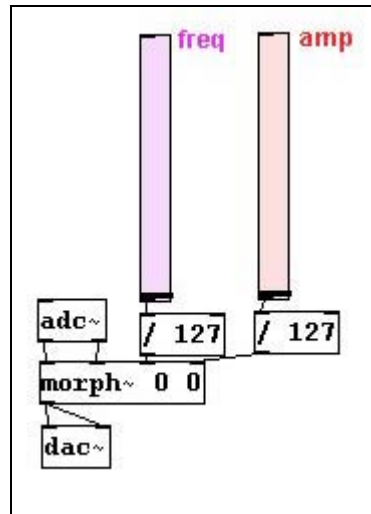


Figure 12. The morph~ object in a PD patch.

The class perform method will then contain the calls to SndObj::DoProcess() methods of each processing object. The methods SndObj::PushIn() and SndObj::PopOut() are used to send the signal into the SndObj chain and to get the processed output, respectively:

```
t_int *morph_tilde_perform(int *w){
  t_sample *in1 = (t_sample*) w[1];
  t_sample *in2 = (t_sample*) w[2];
  t_sample *out = (t_sample*) w[3];
  t_int size = (t_int) w[4];
  t_morph_tilde *x =
    (t_morph_tilde*)w[5];

  int pos =
    x->inobj1->PushIn(in1, size);
  x->inobj2->PushIn(in2, size);
  x->synth->PopOut(out, size);

  if(pos == DEF_VECSIZE){
    x->spec1->DoProcess();
    x->spec2->DoProcess();
    x->morph->DoProcess();
    x->synth->DoProcess();
  }
  return (w+6);
}
```

9. Conclusion

The techniques of spectral processing are very powerful. We have seen that they in fact have a multitude of applications, of which we saw the classic and most important ones. The standard DFT is generally a very practical spectral analysis tool, mostly because of its simplicity and elegance, as well as the existence of fast computation algorithms for it. There are adaptations and variations of it, which try to overcome some of its shortcomings, with important applications in signal analysis.

Nevertheless their use in sound processing and transformation is still somewhat limited. In addition to Fourier-based processes, which have so far been the most practical and useful ones to implement, there are other methods of spectral analysis. The most important of these is the Wavelet Transform (Meyer, 1991), which so far has had limited use in audio signal processing.

10. Bibliography

- Abe T, et al (1997). "The IF spectrogram: a new spectral representation," *Proc. ASVA 97*: 423-430.
- Dolson, M (1986). "The Phase Vocoder Tutorial". *Computer Music Journal*, 10(4): 14-27. MIT Press, Cambridge, Mass.
- Flanagan, JL, Golden RM (1966). "Phase Vocoder". *Bell System Technical Journal* 45: 1493-1509.
- Jaffe, D (1987a). "Spectrum Analysis Tutorial, Part 1: The Discrete Fourier Transform". *Computer Music Journal*, 11(2): 9-24. MIT Press, Cambridge, Mass.
- Jaffe, D (1987b). "Spectrum Analysis Tutorial, Part 2: Properties and Applications of the Discrete Fourier Transform". *Computer Music Journal*, 11(2): 17-35. MIT Press, Cambridge, Mass.
- Lazzarini, V (2000). "The Sound Object Library". *Organised Sound* 5 (1). Cambridge University Press, Cambridge.
- McCaulay, RJ, Quatieri, TF (1986). "Speech Analysis/Synthesis Based on a Sinusoidal Representation". *IEEE Trans. On Acoustics, Speech, and Signal Processing*, ASSP-34 (4).
- Meyer, Y (ed.)(1991). *Wavelets and applications*. Springer-Verlag, Berlin, 1991
- Moore, FR (1990). *Elements of Computer Music*. Prentice-Hall, Englewood Cliffs, N.J.
- Openheim, AV, Schaffer, RW (1975). *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, N.J.
- Serra, X. (1997). "Musical Sound Modelling with Sinusoids plus Noise". in: G.D. Poli et al (eds.), *Musical Signal Processing*, Swets & Zeitlinger Publishers, Amsterdam
- Steiglitz, K (1995). *A Signal Processing Primer*. Addison-Wesley Publ., Menlo Park, Ca.