ENGIN 100: Music Signal Processing

Synthesis of Simple Music Signals

Professor Andrew E. Yagle

Dept. of EECS, The University of Michigan, Ann Arbor, MI 48109-2122

I. ABSTRACT

These notes are a quick introduction to synthesis of musical signals. They focus on digital turntable synthesis, since this can be implemented easily on a recording of an actual instrument using upsampling, downsampling, and the Circle of Fifths. They also discuss additive synthesis, which allows synthesis of artificial musical instruments and provides direct control over frequency. Subtractive and FM synthesis are not discussed; they are primarily analog techniques, and so they lie beyond the scope of Engin 100. The application of this to the second project (simple musical synthesis and transcription) should be evident.

II. BACKGROUND

There are at least four major approaches to synthesis of musical signals. Two (FM and subtractive synthesis) apply primarily to analog signal processing, and will not be discussed. Additive and turntable synthesis were developed for analog signal processing, but can also be used in digital signal processing.

III. TURNTABLE SYNTHESIS

The most obvious way to synthesize music is to record on tape short snippets of actual musical instruments playing a note. The pitch of the note played can be varied by speeding up or slowing down the tape during playback. Various instruments playing various notes can then be cut and pasted together (literally, using tape!) or dubbed together to create music. This works by using a variable speed motor to drive the tape.

The same idea can be used in digital signal processing. Obviously it is easier to add and concatenate (string together) signals digitally, since no physical processes are involved. Altering the pitch of a recorded instrument can be accomplished by multirate filtering (downsampling and upsampling), discussed next.

A. Multirate Filtering I: Downsampling–Increasing Frequency by an Integer Factor

Doubling or tripling the pitch of a digital recording is simple, *provided all of the increased frequencies are less than half of the sampling rate.* Simply omit every other sample to double the pitch, and omit two out of three samples to triple pitch. A moment's thought shows that this is equivalent to reducing the sampling rate by a factor of two or three. The effect of this is to double or triple the frequency of each harmonic in the Fourier series expansion of the signal, which doubles or triples the pitch of the musical signal. This also reduces the duration of the signal, so the duration of the original signal must be increased by concatenation.

Matlab commands for downsampling a digital signal `X` (row vector) with length `L=length(X)`:
>> `Z=[X X];Y=Z(1:2:2*L);` to double the pitch; >> `Z=[X X X];Y=Z(1:3:3*L);` to triple it. OR:
>> `F=fft(X);Y=(1/2)*real(ifft([F(1:L/4+1) F(3*L/4+2:L)]));Y=[Y Y];` using $f=(K-1)\frac{F}{N}$.

To see why downsampling works, consider its effect on an f-Hertz sinusoid sampled at $F\frac{\text{SAMPLE}}{\text{SECOND}}$:

$$x(t)=A\cos(2\pi ft)\rightarrow x[n]=x(t=\frac{n}{F})=A\cos(2\pi\frac{f}{F}n)\,\text{for}\,n=\ldots-2,-1,0,1,2\ldots \tag{1}$$

The downsampled signal (omit $x[n]$ for n odd) is then

$$y[n]=x[2n]=A\cos(2\pi\frac{f}{F}(2n))=A\cos(2\pi\frac{2f}{F}n)\,\text{for}\,n=\ldots-2,-1,0,1,2\ldots \tag{2}$$

which immediately shows that downsampling doubles frequency.

B. Multirate Filtering II: Upsampling–Reducing Frequencies by an Integer Factor

To halve the pitch of a digital recording is slightly more complicated. Instead of discarding samples, we must include additional samples, which *interpolate* the given samples. Simply averaging two samples to get the interpolated sample between them gives an approximate solution, but the *exact* solution is as follows:

- Insert a zero between each pair of given samples, giving $\{\ldots x[-2],0,x[-1],0,x[0],0,x[1],0,x[2]\ldots\}$;
- Digitally lowpass filter the resulting signal (which is twice as long as the original signal) as follows:
- (1) Compute its DFT; (2) Set the middle half of the DFT values to zero; (3) Compute the inverse DFT;
- The resulting signal has its pitch and amplitude halved, but its duration is twice as long.

Matlab commands for upsampling by 2 a digital signal `X` (row vector) of even length `L=length(X)`:
>> `Z=[X;zeros(1,L)];F=fft(Z(:));F(L/2+1:3*L/2+1)=zeros(1,L+1);Y=real(ifft(F));` OR:
>> `F=fft(X);Y=2*real(ifft([F(1:L/2) zeros(1,L+1) F(L/2+2:L)]));Y=Y(1:L);`

To see why upsampling works, note that the signal with zeros inserted at odd-numbered times is

$$z[n]=\begin{cases}x[n/2] & \text{for } n \text{ even}\\ 0 & \text{for } n \text{ odd}\end{cases}=x[n/2](1+(-1)^n)/2=A\cos(2\pi\frac{f}{F}\frac{n}{2})(1+\cos(n\pi))/2\,\text{for}\,n=\ldots-2,-1,0,1,2\ldots \tag{3}$$

Using our old friend $2\cos(x)\cos(y)=\cos(x+y)+\cos(x-y)$ with $x=2\pi\frac{f/2}{F}n$ and $y=n\pi=2\pi\frac{F/2}{F}n$ gives

$$z[n]=\frac{A}{2}\cos\left[2\pi\frac{f/2}{F}n\right]+\frac{A}{4}\cos\left[2\pi\frac{f+F}{2F}n\right]+\frac{A}{4}\cos\left[2\pi\frac{f-F}{2F}n\right]=\frac{A}{2}\cos\left[2\pi\frac{f/2}{F}n\right]+\frac{A}{2}\cos\left[2\pi\frac{F-f}{2F}n\right]. \tag{4}$$

Note that $\cos(2\pi\frac{f+F}{2F}n)$=$\cos(2\pi\frac{f-F}{2F}n)$=$\cos(2\pi\frac{F-f}{2F}n)$ since $\cos(x)=\cos(x-2\pi n)=\cos(-x)$.

This shows filtering out the sinusoid with frequency above $F/4$ leaves a sinusoid with halved frequency $f/2$. Repeating this argument for each harmonic in a Fourier series shows upsampling halves frequency.

C. Circle of Fifths

While upsampling and downsampling can be applied to alter pitch by a factor of any rational number, it is preferable to keep the factors small. Fortunately, we can alter any of the 12 semitones to any other semitone, by repeatedly downsampling by 2 and upsampling by 3, and by using the *Circle of Fifths*.

The Circle of Fifths is due to two numerical facts:
(1) $3^{12} = 531,441 \approx 524,288 = 2^{19} \rightarrow \frac{3}{2} \approx 2^{7/12}$; (2) 7 and 12 are relatively prime (no common factor).

As a result of these two facts, we have the following table, starting with note "A" at the **right** edge:

Frequency / 440 Hz	$\frac{(\frac{3}{2})^0}{2^0}$	$\frac{(\frac{3}{2})^1}{2^0}$	$\frac{(\frac{3}{2})^2}{2^1}$	$\frac{(\frac{3}{2})^3}{2^1}$	$\frac{(\frac{3}{2})^4}{2^2}$	$\frac{(\frac{3}{2})^5}{2^2}$	$\frac{(\frac{3}{2})^6}{2^3}$	$\frac{(\frac{3}{2})^7}{2^4}$	$\frac{(\frac{3}{2})^8}{2^4}$	$\frac{(\frac{3}{2})^9}{2^5}$	$\frac{(\frac{3}{2})^{10}}{2^5}$	$\frac{(\frac{3}{2})^{11}}{2^6}$	$\frac{(\frac{3}{2})^{12}}{2^7}$
Hertz	440	660	495	742	557	835	626	470	705	529	793	595	446
Note	A	E	B	F#	C#	G#	D#	A#	F	C	G	D	A

This shows that by downsampling by 2 (sometimes twice) and upsampling by 3 repeatedly, we can obtain close approximations to all 12 semitones from any single semitone. So given a recording of an instrument playing any semitone, we can alter its pitch to any of the other semitones. Octave changes are handled by separate upsampling or downsampling by 2. Since we *reduce* frequencies each time, no aliasing occurs.

We can downsample by 2 and upsample by 3 (reduce frequencies by 2/3) in a single step as follows:

```
>> F=fft(X);Y=(3/2)*real(ifft([F(1:L/2) zeros(1,L/2+1) F(L/2+2:L)]));Y=Y(1:L);
```

This is called the Circle of Fifths because arranging the 12 semitones in a circle, like hours on a clock face, and taking repeated jumps of 7 "hours" each, we can attain all 12 "hours." A Fifth is the interval between 5 whole tones (such as A to E above) or 7 semitones. The Circle of Fifths is actually a relic of the era when pitches were related exactly by various ratios of small integers, instead of the constant ratios used today.

IV. ADDITIVE SYNTHESIS

A. Specification of Fourier Series Amplitudes c_k

Once we accept that musical notes can be represented by Fourier series of a fundamental and harmonics, also called overtones or partials (partials can vary over time), a way to synthesize music is to generate

$$x(t) = c_1 \cos(2\pi f t) + c_2 \cos(2\pi 2 f t) + \ldots \rightarrow x[n] = c_1 \cos(2\pi \frac{f}{F} n) + c_2 \cos(2\pi \frac{2f}{F} n) + c_3 \cos(2\pi \frac{3f}{F} n) + \ldots \quad (5)$$

- This generates a pitch at frequency f Hertz for a sampling rate $F \frac{\text{SAMPLE}}{\text{SECOND}}$;
- The amplitudes c_k determine the timbre of the sound (varies by instrument);
- Phases are omitted, since the human ear does not perceive them (a big break);
- There are only a finite number $\frac{F}{2f}$ of terms, since kf may not exceed $\frac{F}{2}$;
- Changing the frequency is merely a matter of using a different value of f.
- A Hammond or pipe organ does this *physically*: opening stops varies the c_k.

`>> soundsc(C*cos(2*pi*440*[1:length(C)]'*[1:F]/F),F)` plays "A" (440 Hertz) sampled at $S\frac{\text{SAMPLE}}{\text{SECOND}}$.

Try listening to this for the following two row vectors of Fourier series amplitudes c_k:

Factor	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}
1/60	24	9	6	10	1.8	4	2.5	.9	.9	.55	0	0	0	0	0
1/400	34	27	60	67	30	42	6	5.1	4.5	3.5	5	16	48	42	12.5

B. Generation of Sinusoids

When implementing in Matlab, the Fourier series can easily be generated directly. However, if a DSP chip is used, computation of all of the cosines can be a problem. Fortunately, we can use our old friend $2\cos(x)\cos(y) = \cos(x+y) + \cos(x-y)$ yet again to eliminate repeated computation of cosines.

We want to generate $x[n] = A\cos(2\pi\frac{f}{F}n)$, a cosine of frequency f Hertz sampled at $F\frac{\text{SAMPLE}}{\text{SECOND}}$. Setting $x = 2\pi\frac{f}{F}n$ and $y = 2\pi\frac{f}{F}$ in the trig identity and multiplying by A gives

$$x[n+1] + x[n-1] = 2\cos(2\pi\frac{f}{F})x[n] \rightarrow x[n+1] = 2\cos(2\pi\frac{f}{F})x[n] - x[n-1] \tag{6}$$

which we recognize as a rearrangement of the frequency estimation algorithm from Lab #2.

But we can also use this equation to recursively compute $x[n+1]$ from $x[n]$ and $x[n-1]$. Starting with $x[0] = A$ and $x[1] = A\cos(2\pi\frac{f}{F})$, we can plug into this equation and compute $x[2], x[3] \ldots$. Each step requires only a single multiplication and subtraction; any chip can perform this action very quickly. This avoids computation of cosines, except for the single constant $2\cos(2\pi\frac{f}{F})$. This also works with nonzero θ.

V. Subtractive and FM Synthesis

These are primarily analog techniques, but we include them for completeness.

A. Subtractive Synthesis

The idea here is to use analog circuitry to generate a simple periodic signal, such as a square wave or sawtooth wave, and then use an analog filter to remove most of its harmonics. Moog synthesizers do this.

B. FM Synthesis

The idea here is to use an FM (Frequency Modulation) signal that looks like

$$x(t) = A\cos(2\pi(f + I\cos(2\pi g t))) \tag{7}$$

where I is the modulation index and f and g are both frequencies. This is a simple way to generate complicated periodic signals rich in harmonics. Another way is to use a nonlinear function $y(t) = \frac{|x(t)|}{1+|x(t)|}$.