Eng. 100: Music Signal Processing
DSP Lecture 10
Music synthesis: Advanced methods

Curiosity:
http://www.sonicvisualiser.org/

Announcements:
• Final Exam: Thu. Dec. 17, 4-6 PM, 1500 EECS
Outline

- Part 1. Beyond Fourier series: More music synthesis methods
- Part 2. Advanced music synthesis methods
  - Amplitude variations
    - Envelope
    - Attack, Decay, Sustain, Release (ADSR)
    - Tremolo
  - Frequency / spectrum variations
    - Vibrato
    - Glissando
- Part 2. Project 3 logistics and Q/A
- Part 3. Project 3 tips
  - `reshape`, `function`
  - Transcriber hints: note durations
- Part 4. DSP application: Beats per minute
Part 1. Beyond Fourier series: More music synthesis methods
Additive synthesis review

Mathematical formula (Fourier series) for additive synthesis [wiki]:

\[ x(t) = \sum_{k=1}^{K} c_k \cos \left( 2\pi \frac{k}{T} t \right) = \sum_{k=1}^{K} c_k \cos(2\pi kf t) \]

- Parameters that control timbre: \( c_1, \ldots, c_K \)
- Parameter that controls pitch: ??
**Frequency Modulation (FM) synthesis**

In 1973, John Chowning of Stanford invented the use of frequency modulation (FM) as a technique for musical sound synthesis.

The mathematical formula for *FM synthesis* is [wiki]:

\[ x(t) = A \sin(2\pi ft + I \sin(2\pi gt)), \]

where \( I \) is the *modulation index* and \( f \) and \( g \) are both frequencies. (Yamaha licensed the patent for synthesizers and Stanford made out well.)

This is a simple way to generate periodic signals that are rich in harmonics. However, finding the value of \( I \) that gives a desired effect requires experimentation.
FM example 1: Traditional

\[ S = 44100; \]
\[ N = 1.0 \times S; \]
\[ t = [0:N-1]/S; \text{ % time samples: } t = n/S \]
\[ I = 7; \text{ % adjustable} \]
\[ x = 0.9 \times \sin(2\pi \times 400 \times t + I \times \sin(2\pi \times 400 \times t)); \]

Very simple implementation (both in analog and digital hardware), yet can produce harmonically very rich spectra:
**FM example 2: Time-varying**

Time-varying modulation index: \( x(t) = A \sin \left( 2\pi ft + I(t) \sin(2\pi gt) \right) \).

Simple formula / implementation can make remarkably intriguing sounds.

```python
S = 44100;
N = 1.0 * S;
t = [0:N-1]/S; % time samples: t = n/S
I = 0 + 9*t/max(t); % slowly increase modulation index
x = 0.9 * sin(2*pi*400*t + I .* sin(2*pi*400*t));
```

Besides making the modulation index \( I \) a vector, how else did the code change?

Previous code for reference:

```python
S = 44100;
N = 1.0 * S;
t = [0:N-1]/S; % time samples: t = n/S
I = 7; % adjustable
x = 0.9 * sin(2*pi*400*t + I * sin(2*pi*400*t));
```

What is the most informative graphical representation?
Illustrations of previous FM signal

Spectrogram of FM signal with time-varying modulation

First 0.05 seconds

Last 0.02 seconds
Another way to make signals that are rich in harmonics is to use a nonlinear function such as \( y(t) = x^9(t) \).

```matlab
S = 44100;
N = 1.0 * S;
t = [0:N-1]/S; % time samples: t = n/S
x = cos(2*pi*400*t);
y = x.^9;
```

![Graphs and plots](image)
Nonlinearities in amplifiers

- High quality audio amplifiers are designed to be very close to linear because any nonlinearity will introduce undesired harmonics (see previous slide).
- Quality amplifiers have a specified maximum total harmonic distortion (THD) that quantifies the relative power in the output harmonics for a pure sinusoidal input.

\[ \cos(2\pi ft) \rightarrow \text{Amplifier} \rightarrow c_1 \cos(2\pi ft) + c_2 \cos(2\pi 2ft) + c_3 \cos(2\pi 3ft) + \cdots \]

- A formula for THD is: [wiki]

\[
\text{THD} = \frac{c_2^2 + c_3^2 + c_4^2 + \cdots}{c_1^2} \cdot 100\%
\]

- What is the best possible value for THD? ??
- On the other hand, electric guitarists often deliberately operate their amplifiers nonlinearly to induce distortion, thereby introducing more harmonics than produced by a simple vibrating string.
  - pure: [play]
  - distorted: [play]
Part 2. Advanced music synthesis methods
Envelope of a musical sound
Envelope example: Train whistle

- **Train whistle signal**
- **Train whistle envelope**

- **Attack**
- **Release**

Play button
Envelope example: Plucked guitar

plucked guitar signal

plucked guitar envelope
Envelope implementation

\[ S = 44100; \]
\[ N = 1 * S; \]
\[ t = [0:N-1]/S; \]
\[ c = 1 ./ [1:2:15]; \text{ % amplitudes} \]
\[ f = [1:2:15] * 494; \text{ % frequencies} \]
\[ x = 0; \]
\[ \text{for } k=1:\text{length}(c) \]
\[ \quad x = x + c(k) * \sin(2 * \pi * f(k) * t); \]
\[ \text{end} \]
\[ \text{env} = (1 - \exp(-80*t)) .* \exp(-3*t); \text{ % fast attack; slow decay} \]
\[ y = \text{env} .* x; \]
Programmable music synthesizers usually allow the user to control separately the time durations of these 4 components of the envelope.

- For synthesizers with a keyboard, the “sustain” portion lasts as long as the key is pressed.
- The “release” portion occurs after the key is released.
- In synthesizers with “touch control” the properties of the “attack” and “decay” portions may depend on how hard/fast one presses the key.
- Does duration of release portion depend on how quickly one releases the key?
- For a pipe organ, how long is the attack and decay?
Example ADSR implementation in Matlab

S = 44100;
N = 1.5 * S;
t = [0:N-1]/S;
c = 1 ./ [1:2:15]; % amplitudes
f = [1:2:15] * 494; % frequencies
x = c * sin(2 * pi * f' * t); % fourier synthesis
env = interp1([0 0.1 0.3 1.1 1.5], [0 1 0.4 0.4 0], t); % !
subplot(211), plot(t, env), xlabel 't [sec]', ylabel 'Envelope'
y = env .* x;

What kind of synthesis was used here? ??
Tremolo
Tremolo implementation: LFO

\[
\begin{align*}
S &= 44100; \\
N &= 1 \times S; \\
t &= [0:N-1]/S; \\
c &= 1 ./ [1:2:15]; \text{ % amplitudes} \\
f &= [1:2:15] \times 494; \text{ % frequencies} \\
x &= c \times \sin(2 \times \pi \times f' \times t); \text{ % concise way} \\
lfo &= 0.5 - 0.4 \times \cos(2\pi6t); \text{ % what frequency?} \\
y &= lfo \times x;
\end{align*}
\]
Tremolo: Why LFO?

\[
\begin{align*}
S &= 44100; \\
N &= 1 * S; \\
t &= [0:N-1]/S; \\
c &= 1 ./ [1:2:15]; \quad \% \text{amplitudes} \\
f &= [1:2:15] * 494; \quad \% \text{frequencies} \\
x &= c * \sin(2 * \pi * f' * t); \quad \% \text{concise way} \\
lfo &= 0.5 - 0.4 * \cos(2*\pi*60*t); \quad \% \text{what frequency now?} \\
y &= lfo .* x;
\end{align*}
\]
Frequency variations: vibrato and glissando
Vibrato
A Leslie speaker in a Hammond organ has both:
Vibrato implementation: LFO

\[ S = 44100; \]
\[ N = 2 \times S; \]
\[ t = [0:N-1]/S; \]
\[ c = 1 ./ [1:2:15]; \% amplitudes \]
\[ f = [1:2:15] \times 494; \% frequencies \]
\[ x = 0; y = 0; \]
\[ lfo = 0.001 \times \cos(2\pi \times 4 \times t) / 4; \% about 0.1\% pitch variation \]
\[ \text{for } k=1: \text{length}(c) \]
\[ \quad \text{fnew} = f(k) \times lfo; \]
\[ \quad x = x + c(k) \times \sin(2 \times \pi \times f(k) \times t); \]
\[ \quad y = y + c(k) \times \sin(2 \times \pi \times f(k) \times t + f(k) \times lfo); \]
\[ \text{end} \]
Vibrato: Why LFO?

(ttry more; you may not like it)

Note: FM synthesis is like an extreme form of vibrato
Glissando
Glissando implementation

\[
\begin{align*}
S &= 44100; \\
N &= 1 \times S; \\
t &= [0:N-1]/S; \\
f &= [1 2^{(-3/12)}] \times 494; \text{ \% frequencies: B G}^\# \\
x &= 0.9 \times [\cos(2\pi f(1) t), \cos(2\pi f(2) t)]; \\
\text{tau} &= 0.25; \text{ \% length of glissando} \\
t2 &= [0:(\text{tau}S)-1]/S; \\
\text{gliss} &= \cos(2\pi f(1) t2 + t2^2/\text{tau}/2*(f(2) - f(1))); \\
y &= 0.9 \times [\cos(2\pi f(1) t), \text{gliss}, \cos(2\pi f(2) t)];
\end{align*}
\]
Frequency variations: Theory

(This page requires calculus and is entirely optional.)

For a (standard) sinusoid: \( x(t) = \cos(2\pi ft) \implies \)

\[
\frac{d}{dt}x(t) = -\sin(2\pi ft) \cdot 2\pi f
\]

\( f \) is frequency

For a sinusoid with time-varying phase: \( x(t) = \cos(\phi(t)) \implies \)

\[
\frac{d}{dt}x(t) = -\sin(\phi(t)) \cdot \left(2\pi \cdot \frac{1}{2\pi} \frac{d}{dt}\phi(t)\right)
\]

\( \phi(t) \) is instantaneous frequency
Example: Glissando

Example. If we want a signal with instantaneous frequency

\[ f(t) = f_1 + \frac{t}{\tau}(f_2 - f_1), \]

then we need

\[ \phi(t) = 2\pi \int_0^t f(t') \, dt' = 2\pi \int_0^t \left[ f_1 + \frac{t'}{\tau}(f_2 - f_1) \right] \, dt' \]

\[ = 2\pi f_1 t + 2\pi \frac{t^2}{2\tau}(f_2 - f_1), \]

so that

\[ \frac{1}{2\pi} \frac{d}{dt} \phi(t) = f_1 + \frac{t}{\tau}(f_2 - f_1). \]

So the desired signal is

\[ x(t) = \cos(\phi(t)) = \cos \left( 2\pi f_1 t + 2\pi \frac{t^2}{2\tau}(f_2 - f_1) \right). \]

See Matlab example on earlier slide.
Vibrato combined with glissando

% fig_gliss2.m glissando + vibrato

S = 44100;
N = 2^15;
t1 = [0:N-1]/S;
f = 2 * [1 2^-(-3/12)] * 494; % frequencies: B G#
x = 0.9 * [cos(2*pi * f(1) * t1), cos(2*pi * f(2) * t1)];
Ngliss = 2^16; tau = Ngliss / S; % length of glissando
t2 = [0:Ngliss-1]/S;
nvibe = 9;
phi = 2 * pi * (f(1) * t2 + t2.^2/tau/2*(f(2) - f(1)) ...
    + 5 * tau/nvibe/2/pi*cos(2*pi*nvibe/tau*t2));
x = 0.9 * [cos(2*pi * f(1) * t1), cos(phi), cos(2*pi * f(2) * t1)]
M = 2^12;
y = reshape(x, M, []);
imagesc(1:size(y,2), [0:M-1]/M*S, 2/M*abs(fft(y)))
axis xy, axis([0 size(y,2) 0 2*500])
Music synthesis summary

- There are numerous methods for musical sound synthesis
- Additive synthesis provides complete control of spectrum
- Other synthesis methods provide rich spectra with simple operations (FM, nonlinearities)
- Time-varying spectra can be particularly intriguing
- Signal envelope (time varying amplitude) also affects sound characteristics
- Other advanced synthesis methods:
  - sound reversal
  - physical modeling
  - sampling
  - ...
- Ample room for creativity and originality!
Part 3. Project 3 tips
Using reshape to simplify indexing

Given a vector $x$ with 3 instruments each playing 12 notes, with $N = 1000$ samples per note.

How do we access the 4th note of the 3rd instrument?

One way: $y = x(27001:28000)$;

Slightly better way: $y = x(27000+(1:N))$;

Still better way: $y = x((2*12+3)*N+(1:N))$;

Elegant way using 3D array slicing:
$z = \text{reshape}(x, N, 12, 3)$;
$y = z(:,4,3)$;
Anonymous function example

Old way to create GUI with three “push button” widgets:

```matlab
uicontrol('style', 'pushbutton', 'position', [100 250 40 80], ...  
    'string', 'A', 'callback', 'disp(1)', 'foregroundcolor', 'red');
uicontrol('style', 'pushbutton', 'position', [150 250 40 80], ...  
    'string', 'B', 'callback', 'disp(2)', 'foregroundcolor', 'red');
uicontrol('style', 'pushbutton', 'position', [200 250 40 80], ...  
    'string', 'C', 'callback', 'disp(3)', 'foregroundcolor', 'red');
```

New way using anonymous functions:

```matlab
fun = @(pos,str,com) uicontrol('style', 'pushbutton', ...
    'position', pos, 'string', str, ...
    'callback', com, 'foregroundcolor', 'red');

fun([100 250 40 80], 'A', 'disp(1)')
fun([150 250 40 80], 'B', 'disp(2)')
fun([200 250 40 80], 'C', 'disp(3)')
```

If we wanted to change the color from red to blue, which way is easier? ??
Transcriber hints: note durations

Project 3 classic synthesizer includes 100 zeros at end of each note to facilitate finding note duration.

<table>
<thead>
<tr>
<th>Note</th>
<th>Whole</th>
<th>Half</th>
<th>Quarter</th>
<th>1 second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>32668 + 100</td>
<td>16284 + 100</td>
<td>8092 + 100</td>
<td>$S = 44100$</td>
</tr>
<tr>
<td></td>
<td>$32768 = 4 \times 8192$</td>
<td>$16384 = 2 \times 8192$</td>
<td>8192</td>
<td></td>
</tr>
</tbody>
</table>

Transcriber must located those zeros. How?

```matlab
a = reshape(x, 8192, []);
b = a(end-99:end,:);
c = sum(abs(b));
d = find(c == 0);
e = [0 d(1:end-1)]
```

Sizes of each variable? ?? ?? ?? ?? ??
Part 4. Beats per minute
% bpm1_gen
% generate metronome tick signal to test bpm estimator

S = 8192;
bpm = 120;
bps = bpm / 60; % beats per second
spb = 60 / bpm; % seconds per beat
t0 = 0.01; % each "tick" is this long
tt = 0:1/S:9; % 9 seconds of ticking

f = 440;
%x = 0.9 * cos(2*pi*440*tt) .* (mod(tt, spb) < t0); % tone
clf, subplot(211), rng(0)
x = randn(1,numel(tt)) .* (mod(tt, spb) < t0) / 4.5; % click via "envelope"
%sound(x, S)
% audiowrite('bpm1a.wav', x, S, 8)
% bpm1_find
% first try at beats-per-minute (bpm) estimator

[x S] = audioread('bpm1a.wav');
x = x'; % row vector
N = numel(x);

% a = real(ifft(abs(fft(x,2*N)).^2)); % autocorrelation
a = real(ifft(abs(fft(abs(x),2*N)).^2)); % why abs?

spacing = [0:(2*N-1)]/S; % why?
good = (spacing > 60/300 & spacing < 60/25); % min and max reasonable bpm
clf, subplot(211)
plot(spacing, a, 'b.-', spacing, 10*good, 'g.-', spacing, a .* good, 'r.-')
xlabel 'shift [s]', ylabel 'autocorrelation', axis([0 4 0 60])
[~, index] = max(a .* good); % highest correlation for reasonable bpm range
disp(sprintf('estimated spb = %g, so bpm = %g', index/S, 60*S/index))
% ir_savefig -tight cw fig_bpm1b
BPM Summary

Is the preceding “metronome” signal periodic?

This small example illustrates several useful ideas.

- Noise blips
- Using modulo $\text{mod}$ for repeating patterns
- Using logical operations like $<$ to make binary signals.
- Constraining $\text{max}$ to reasonable search range
- Looking for correlation between bursts of noisy signals using $\text{abs}$
- A few lines of $\text{Matlab}$ code can do sophisticated DSP operations

Summary: (auto)correlation is quite widely useful