一、Background knowledge Introduction

背景知識介紹

Musical Analysis and Synthesis in Matlab

Mark R. Petersen (mark.petersen@colorado.edu), Applied Mathematics Department, University of Colorado, Boulder, CO 80309-0526

This article presents musical examples and computer tools that could be used in an introductory pde and Fourier Series class, where students work with the heat and wave equations. The study of Fourier series from a musical perspective offers great insight into basic mathematical concepts and the physics of musical instruments. Tools available in Matlab allow students to easily analyze the wave forms and harmonics of recorded sounds and to synthesize their own. These experiments are a thought provoking way to understand how a wave form is composed of a summation of Fourier Series basis functions and how this relates to the frequency domain. Many students I have worked with have special musical interests and go on to conduct experiments with their own instruments.

The plucked string. One of the standard problems in an introductory pde course is the wave equation with Dirichlet boundary conditions,

$$\begin{cases} u_{tt} = c^2 u_{xx}, & x \in (0, L), t \geq 0 \\ u(0,t) = u(L,t) = 0, & t \geq 0 \\ u(x,0) = \alpha(x),\ u_t(x,0) = \beta(x), & x \in (0, L). \end{cases} \tag{1}$$

Physically, we can think of $u(x,t)$ as the displacement of a plucked guitar string with initial displacement $\alpha(x)$ and initial velocity $\beta(x)$. Additional terms may be added to the wave equation to account for string stiffness and internal damping [**1**]. The solution, obtained by separation of variables, is

$$u(x,t) = \sum_{n=1}^{\infty} \sin\frac{n\pi}{L}x \left(a_n \cos\frac{n\pi}{L}ct + b_n \sin\frac{n\pi}{L}ct \right), \tag{2}$$

where

$$a_n = \frac{2}{\pi}\int_0^L \alpha(x)\ \sin\frac{n\pi}{L}x\,dx \tag{3}$$

$$b_n = \frac{2L}{cn\pi^2}\int_0^L \beta(x)\ \sin\frac{n\pi}{L}x\,dx \tag{4}$$

are the Fourier coefficients of $\alpha(x)$ and $\beta(x)$. Using the trig identity $\cos(x-y) = \cos x\ \cos y + \sin x\ \sin y$ we may rewrite the solution as

$$u(x,t) = \sum_{n=1}^{\infty} p_n\ \sin\frac{n\pi}{L}x\ \cos\frac{n\pi}{L}c\,(t-\gamma_n). \tag{5}$$

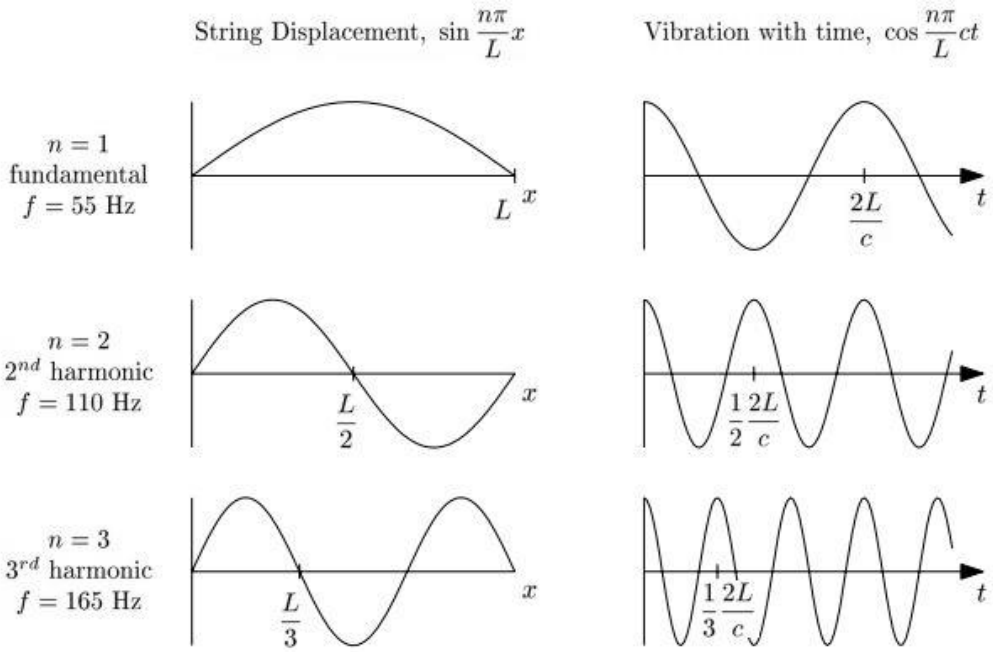


Figure 1. Modes of a vibrating string.

The solution is a superposition of many modes, where each mode oscillates at a different frequency as shown in Figure 1. The first factor in the sum, $\sin \frac{n\pi}{L}x$, is a snapshot of the vibrating string. The second factor, $\cos \frac{n\pi}{L}c(t - \gamma_n)$, shows how this mode vibrates with time. The nth mode has a frequency of $f_n = n\frac{c}{2L}$, a phase shift of γ_n, and the amplitude of each mode is $p_n = \sqrt{a_n^2 + b_n^2}$.

A musician would call f_1 the fundamental frequency and $f_2, f_3, \ldots$ its harmonics. The pitch of each note on the keyboard is associated with a specific fundamental frequency. For example, a low low low A has a fundamental frequency of 55 Hz, and harmonics occur at $110, 165, 220, 275, \ldots$. Note that the harmonics form an arithmetic sequence, $f_n = nf_1$. This is because a string with fixed endpoints can physically only vibrate in the modes shown in column 1 of Figure 1.

The wave speed c can be shown to be the square root of the string's tension over its line density [**1**, p. 36]. Thus the pitch of a string is described by three parameters,

$$f_1 = \frac{1}{2 \text{ length}} \sqrt{\frac{\text{tension}}{\text{line density}}}. \tag{6}$$

This makes sense, as tightening a guitar string increases its pitch while choosing longer strings and thicker strings lowers the pitch. A guitarist changes pitch while playing by shortening the string against the frets.

The solution (5) describes a vibrating string, but other instruments like woodwinds and brass are similar. When an instrument vibrates it sets the surrounding air in motion, producing oscillations in pressure. These are the sound waves detected by your ear. Thus a microphone near our vibrating string would record

$$y(t) = \sum_{n=1}^{\infty} p_n \cos 2\pi f_n (t - \gamma_n). \tag{7}$$

Physically the amplitudes p_n describe the energy associated with each harmonic. The power spectrum is the distribution of energy among the harmonics, in this case the vector $[p_1, p_2, p_3, \ldots]$. For real sounds the power is concentrated in the first 10 to 20 harmonics and the power spectrum is a continuous function of frequency.

Harmonic analysis. At this point there is a wealth of experiments that a student (or instructor) could conduct. Questions about how the spectrum of harmonics changes for different instruments and notes can be investigated using Matlab. The function `analyze.m` [**4**] uses the built-in Matlab functions `wavread` and `fft` to calculate the power spectrum of a Microsoft wave (.wav) sound file. A similar function named `auread` can be used for UNIX audio files.

The plots produced by `analyze.m` can be used to identify the pitch and volume of a sound sample. Figure 2 shows the results of low A, middle A, and high A played on a piano. The waveforms on the left are the pressure variations with time detected by the microphone. The amplitude of the wave is a measure of its pressure oscillations and corresponds to the volume of the sound. Volume is usually measured in decibels, the logarithm of pressure. One does not usually think of sound in terms of pressure, but feeling the pressure waves of sound at a concert or near loud equipment is a common experience.

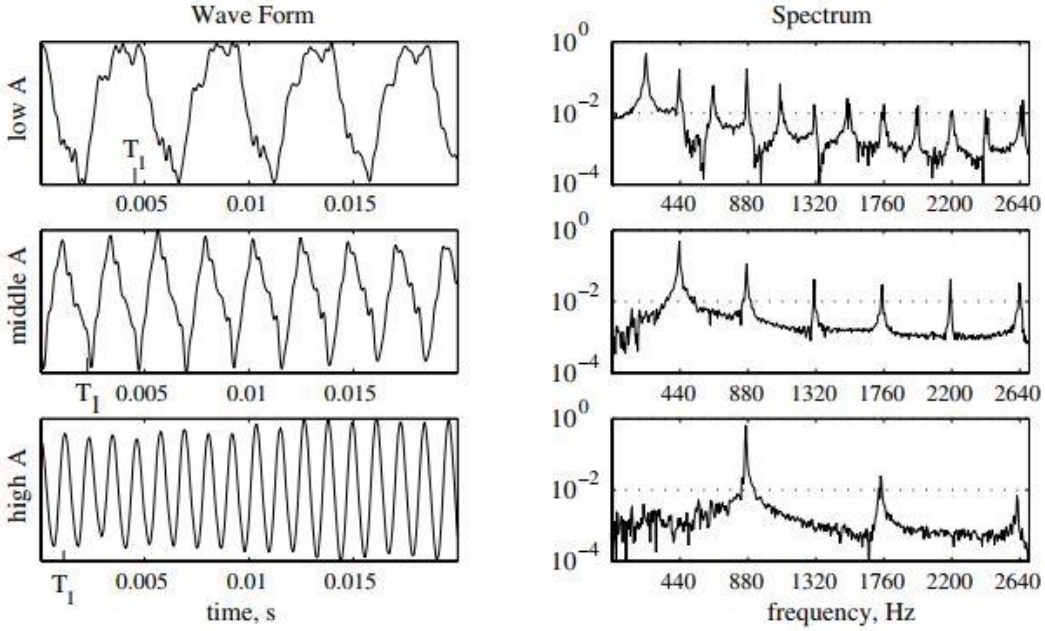


Figure 2. Analysis of piano notes using `analyze.m`

From the wave forms in Figure 2 one can immediately see the periodic nature of each sound and pick out its fundamental period T_1. The corresponding fundamental frequencies ($f_1 = 1/T_1$) are 220 Hz, 440 Hz, and 880 Hz for the three notes and can be seen as the first peaks on the power spectrums. The fundamental frequency f_1 doubles with each octave, and the harmonics are spaced in proportion to f_1 as expected. The spectrum is nonzero between harmonics because the waves are not perfectly periodic.

Notice that the difference in frequency between any two consecutive harmonics is the fundamental frequency. The human mind unconsciously uses this fact to identify a pitch even if the fundamental and lower harmonics are missing. This is how small

speakers make the low sounds of a bass guitar without actually producing low frequencies (Exercise 2).

Going up an octave always corresponds to a doubling in frequency. Thus the frequencies of octaves form a geometric sequence. For example, the frequencies of As are 55, 110, 220, 440, 880 . . . . According to (6) an octave's doubling in frequency can be accomplished by halving the length of the string, as can be verified on any stringed instrument.

The fact that octaves form a geometric sequence has many physical manifestations, such as:

- Low instruments must be much larger than high instruments. In general, an instrument which is an octave lower must be twice as large. For example, in the string family, as we progress from violin, viola, cello, to bass, the cello is large and the bass is very large.
- Organ pipes must also double in size to go down an octave. This is why the organ pipes at the front of a church, if arranged in descending order, approximate an exponential curve.
- Frets on a guitar are far apart at the neck and close together near the body, a pattern which also appears on log graphing paper. Frets and log paper both follow an inverse exponential pattern.

The human mind can identify a seemingly infinite number of instruments by their sound alone, even if they are playing the same pitch at the same volume. What is it that differentiates the pressure signal of a flute from that of a violin? Figure 3 shows the wave form and spectrum of several instruments playing middle A (440 Hz). The wave forms look completely different, but all have the same fundamental period of 0.0023 seconds. The timbre, or sound quality of an instrument, is due to the relative strengths of the harmonics. A pure tone, composed of the fundamental alone, is shrill and metallic, like a tuning fork. Power in the higher harmonics add warmth and color to the tone. Figure 3 shows that a flute is dominated by the lower harmonics, accounting for its simple whistle-like timbre. In contrast violins have power in the higher harmonics,

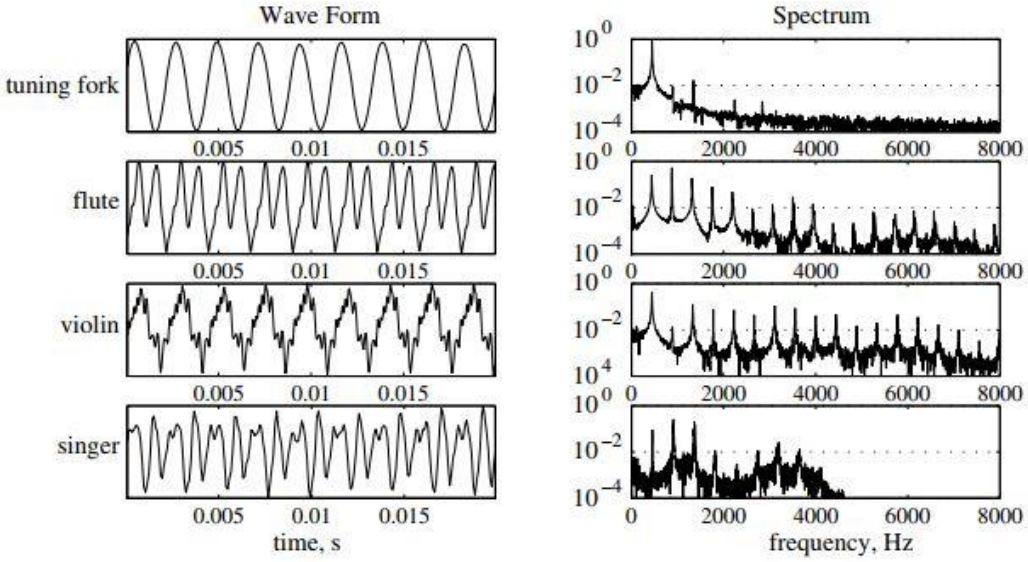


Figure 3. Analysis of several instruments using `analyze.m`

which gives them a warmer, more complex sound. The effect of higher harmonics is clearly seen in the waveform, where the violin has complex oscillations within each period.

Now consider a singer who sings different vowels at the same pitch and volume. Again, the difference between vowels is the distribution of power among the harmonics. In 1859 Helmholtz was able to detect individual harmonics using membranes and bulbs which vibrate sympathetically at specific frequencies [**2**, p. 42]. He then created a series of tuning forks tuned to the harmonics of $b^\flat$. Each fork was forced to vibrate electromagnetically while its volume was controlled by a cover. He astonished audiences by creating synthesized vowel sounds by simply adjusting the covers of these tuning forks (see Exercise 3).

Synthesis. At this point, I present the following role-play to my students:

> Imagine you are an engineer for Yamaha in 1958, and your boss comes in the office one day and says, "Can we make a keyboard that sounds like any instrument you choose? That could revolutionize the music industry!" And you say, "Yes, I know how to do that!"

We know that musical sounds can be analyzed to measure the distribution of power in the harmonics. For a particular instrument, this distribution is the key to synthesizing its sound. Given a fundamental frequency f_1 and power p_n associated with the nth harmonic, the synthesized waveform is

$$y(t) = \sum_{n=1}^{N} p_n \cos 2\pi n f_1 t. \tag{8}$$

The function `synthesize.m` [**4**] creates sound wave data in this way, and then converts the data into a wave sound file with the Matlab function `wavwrite`. Students can use this simple program to experiment with different combinations of harmonics. Higher frequencies sound louder to the human ear than lower frequencies of the same decibel level (amplitude). Specifically, when choosing the power spectrum to create a sound, frequencies between 3000 and 5000 Hz should have a tenth the amplitude of lower frequencies in order to produce the same apparent volume [**3**, p. 233].

Early synthesized music used a power spectrum which was constant in time, which gave it a false, drone sound. In reality the power spectrum changes as the musician varies volume, vibrato, and phrasing. Modern keyboards vary the power spectrum with time to account for the attack and decay of a hit piano string or plucked banjo. They also vary the spectrum for high, mid, and low ranges as real instruments do. Synthesized music has improved dramatically since the sixties, but it could never be as variable or expressive as a musician on a live instrument.

二、Research methods and steps

研究方法及步驟

Part 1: Matlab functions and loops

Matlab anonymous functions (1)

Tedious way to write a 4-note song in Matlab:

```
S = 8192;
sound(0.9*cos(2*pi*660*[0:1/S:0.5]), S); pause(0.5)
sound(0.9*cos(2*pi*880*[0:1/S:0.5]), S); pause(0.5)
sound(0.9*cos(2*pi*660*[0:1/S:0.5]), S); pause(0.5)
sound(0.9*cos(2*pi*440*[0:1/S:1.0]), S); pause(1.0)
```

Matlab anonymous functions (2)

Using an *anonymous function* can simplify and clarify:

```
S = 8192;
playnote = @(f,d) sound(0.9*cos(2*pi*f*[0:1/S:d]), S);
playnote(660, 0.5); pause(0.5)
playnote(880, 0.5); pause(0.5)
playnote(660, 0.5); pause(0.5)
playnote(440, 1.0); pause(1.0)
```

Matlab loops (1)

Using a for loop is the most concise and elegant: (fig/loop/s_loop1c.m)

```
S = 8192;
playnote = @(f,d) sound(0.9*cos(2*pi*f*[0:1/S:d]), S);
fs = [660 880 660 440]; % list of frequencies
ds = [0.5 0.5 0.5 1.0]; % list of durations
for index=1:numel(fs)
        playnote(fs(index), ds(index))
        pause(ds(index))
end
```

Matlab loops (2)

Here is another loop version that sounds better: play

```
S = 8192;
note = @(f,d) 0.9*cos(2*pi*f*[0:1/S:d]);
fs = [660 880 660 440];
ds = [0.5 0.5 0.5 1.0];
x = [];
for index=1:numel(fs)
        x = [x note(fs(index), ds(index))];
end
sound(x, S)
```

Loops (and functions) are ubiquitous in software.

Matlab functions

Two types of functions in Matlab:

- Anonymous functions
  defined "in line" using syntax like: `triple = @(x) 3 * x;`
- Functions defined in separate files.

Example. If the file `triple.m` is in your Matlab path and has the lines

```
function y = triple(x)
y = 3 * x;
```

Then you can use it like any other Matlab function, *e.g.*,

```
a = [10 15 20];
b = triple(a)
```

Part 2: Music synthesis via Fourier series

Additive Synthesis: Mathematical formula

Simplified version of Fourier series for monophonic audio:

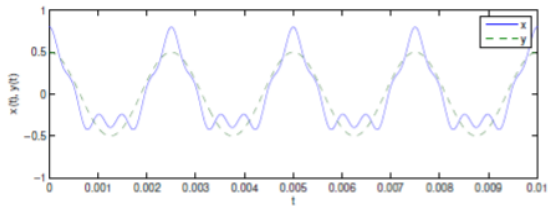
$$x(t) = \sum_{k=1}^{K} c_k \cos\left(2\pi \frac{k}{T} t\right)$$

- No DC term for audio: $c_0 = 0$.
- Phase unimportant for monophonic audio, so $\theta_k = 0$.
- Which version is this? ??
  (Sinusoidal form? trigonometric form? complex exponential form?)

Example:

$$x(t) = 0.5\cos(2\pi 400t) + 0.2\cos(2\pi 800t) + 0.1\cos(2\pi 2000t)$$

Example: Why we might want harmonics



play play

$$y(t) = 0.5\cos(2\pi 400t)$$
$$x(t) = 0.5\cos(2\pi 400t) + 0.2\cos(2\pi 800t) + 0.1\cos(2\pi 2000t)$$

```
% fig_why1: example of additive synthesis
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
y = 0.5 * cos(2*pi * 400 * t);
x = y + 0.2 * cos(2*pi * 800 * t) + 0.1 * cos(2*pi * 2000 * t);
plot(t, x, '-', t, y, '--'), legend('x', 'y')
xlabel 't', ylabel 'x(t), y(t)', axis([0 0.01 -1 1])
```

Same fundamental period, same pitch, but different timbre.

Matlab implementation: "Simple"

Example: $x(t) = 0.5\cos(2\pi 400t) + 0.2\cos(2\pi 800t) + 0.1\cos(2\pi 2000t)$

- A simple Matlab version looks a lot like the mathematical formula:

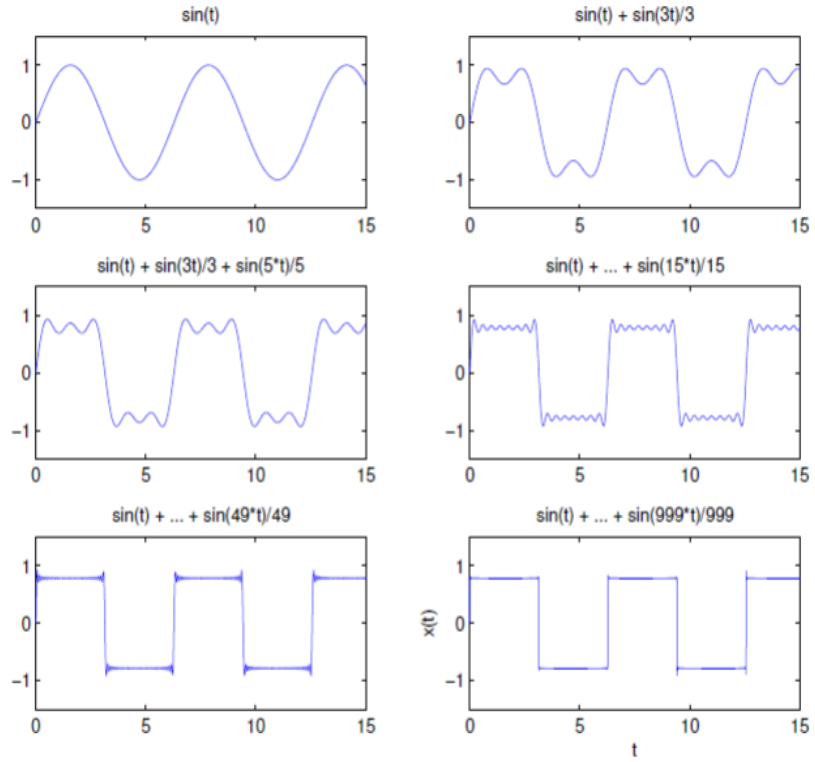
```
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
x = 0.5 * cos(2 * pi * 400 * t) ...
+ 0.2 * cos(2 * pi * 800 * t) ...
+ 0.1 * cos(2 * pi * 2000 * t);
```

- There are many "hidden" `for` loops above. Where? ??
- Matlab saves us from the tedium of writing out those loops, thereby making the syntax look more like the math.
- In "traditional" programming languages like C, one would have to code all those loops.
- This "simple" implementation is still somewhat tedious, particularly for signals having many harmonics.

C99 implementation

```
#include <math.h>
void my_signal(void)
{
        float S = 44100;
        int N = 0.5 * S; // 0.5 sec
        float x[N]; // signal samples
        for (int n=0; n < N; ++n)
        {
                float t = n / S;
                x[n] = 0.5 * cos(2 * M_PI * 400 * t)
                        + 0.2 * cos(2 * M_PI * 800 * t)
                        + 0.1 * cos(2 * M_PI * 2000 * t);
        }
}
```

Example revisited: Square wave



Many dozens of harmonics needed to get a "good" square wave approximation.

Matlab implementation: Loop over harmonics

Example: $x(t) = 0.5\cos(2\pi 400t) + 0.2\cos(2\pi 800t) + 0.1\cos(2\pi 2000t)$

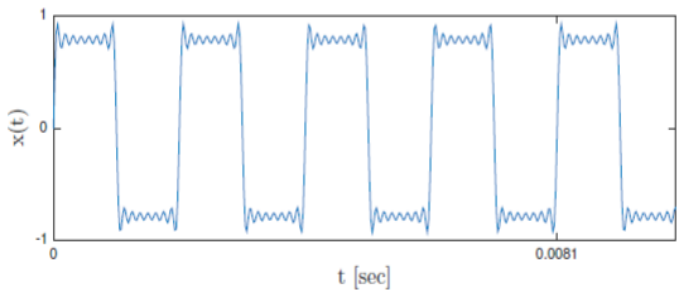
```
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
c = [0.5 0.2 0.1]; % amplitudes
f = [1 2 5] * 400; % frequencies
x = 0;
for k=1:length(c)
        x = x + c(k) * cos(2 * pi * f(k) * t);
end
```

- I think this version is the easiest to read and debug.
- It looks the most like the Fourier series formula: $x(t) = \sum_{k=1}^{K} c_k \cos\left(2\pi \frac{k}{T} t\right)$.
- In fact it is a slight generalization.
  - In Fourier series, the frequencies are multiples: k/T.
  - In this code, the frequencies can be any values we put in the `f` array.

Example: Square wave via loop, with sin

```
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
c = 1 ./ [1:2:15]; % amplitudes
f = [1:2:15] * 494; % frequencies
x = 0;
for k=1:length(c)
        x = x + c(k) * sin(2 * pi * f(k) * t);
end
```

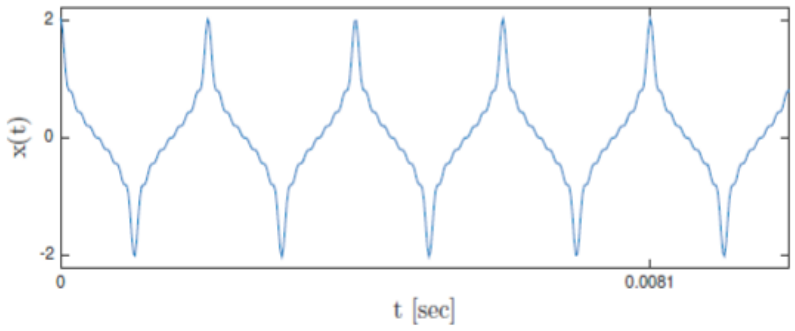
How many harmonics in this example? ??



play

Example: Square wave via loop, with cos

```
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
c = 1 ./ [1:2:15]; % amplitudes
f = [1:2:15] * 494; % frequencies
x = 0;
for k=1:length(c)
        x = x + c(k) * cos(2 * pi * f(k) * t);
end
```



Matlab implementation: Concise

We can avoid writing any explicit `for` loops (and reduce typing) by using the following more concise (*i.e.*, tricky) Matlab version:

```
S = 44100;
N = 0.5 * S; % 0.5 sec
t = [0:N-1]/S; % time samples: t = n/S
c = [0.5 0.2 0.1]; % amplitudes
f = [1 2 5] * 400; % frequencies
z = cos(2 * pi * f' * t);
x = c * z;
```

Part 3: Other music synthesis techniques

Frequency Modulation (FM) synthesis

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

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

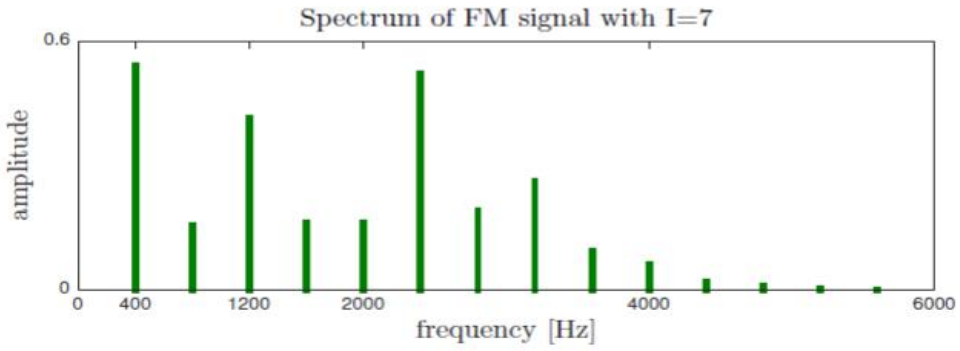
where I is the *modulation index* and f and g are both frequencies.
(Yamaha licensed the patent for synthesizers and Stanford made out well.)

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 * 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));
```

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: $x(t) = A\sin\left(2\pi ft + \underbrace{I(t)}\sin(2\pi gt)\right)$.

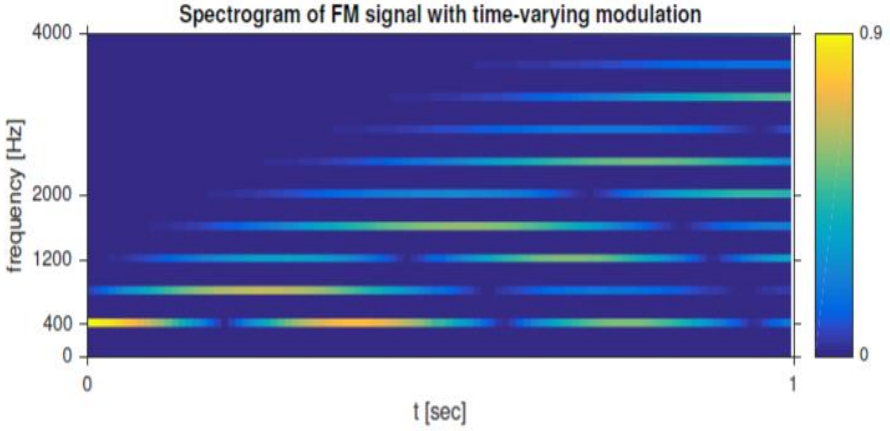
Simple formula / implementation can make remarkably intriguing sounds. play

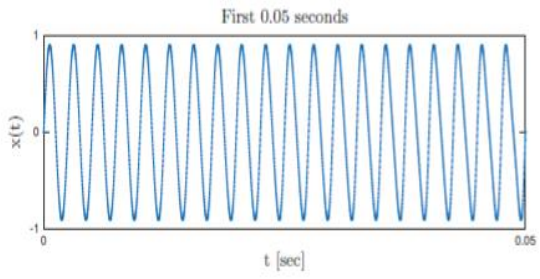
```
S = 44100;
N = 1.0 * S;
t = [0:N-1]/S; % time samples: t = n/S
I = 0 + 9*t/max(t); % slowly increase modulation index
x = 0.9 * sin(2*pi*400*t + I .* sin(2*pi*400*t));
```

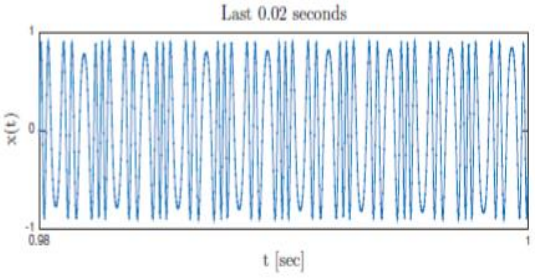
Besides making the modulation index I a vector, how else did the code change? ??
Previous code for reference:

```
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));
```

Illustrations of previous FM signal



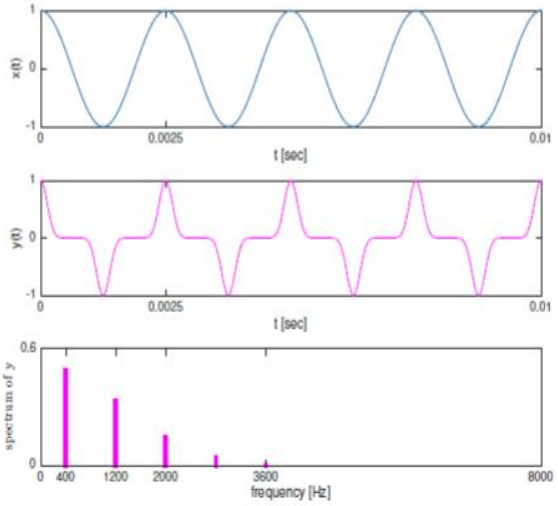




Nonlinearities

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

```
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;
```



play

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.

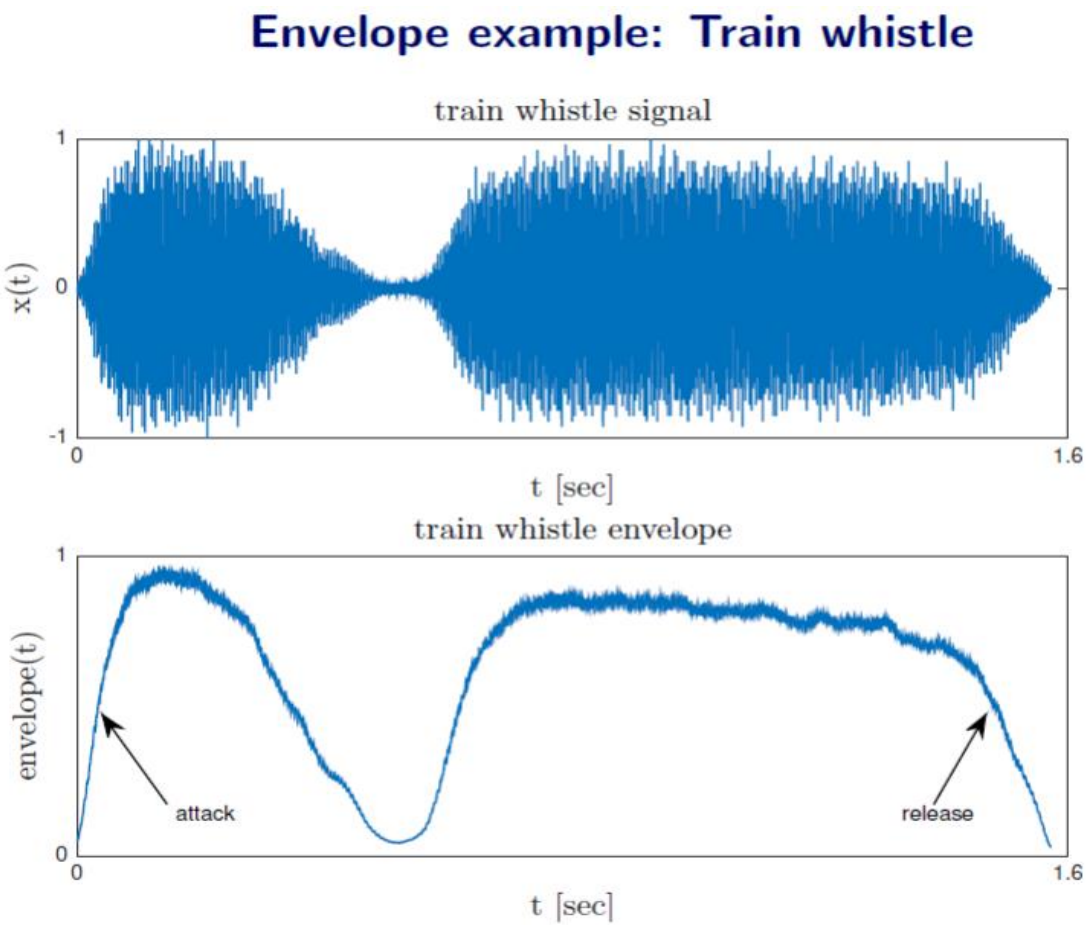
$$\underbrace{\cos(2\pi ft)}_{\text{input}} \rightarrow \boxed{\text{Amplifier}} \rightarrow c_1\cos(2\pi ft) + c_2\cos(2\pi 2ft) + c_3\cos(2\pi 3ft) + \cdots$$

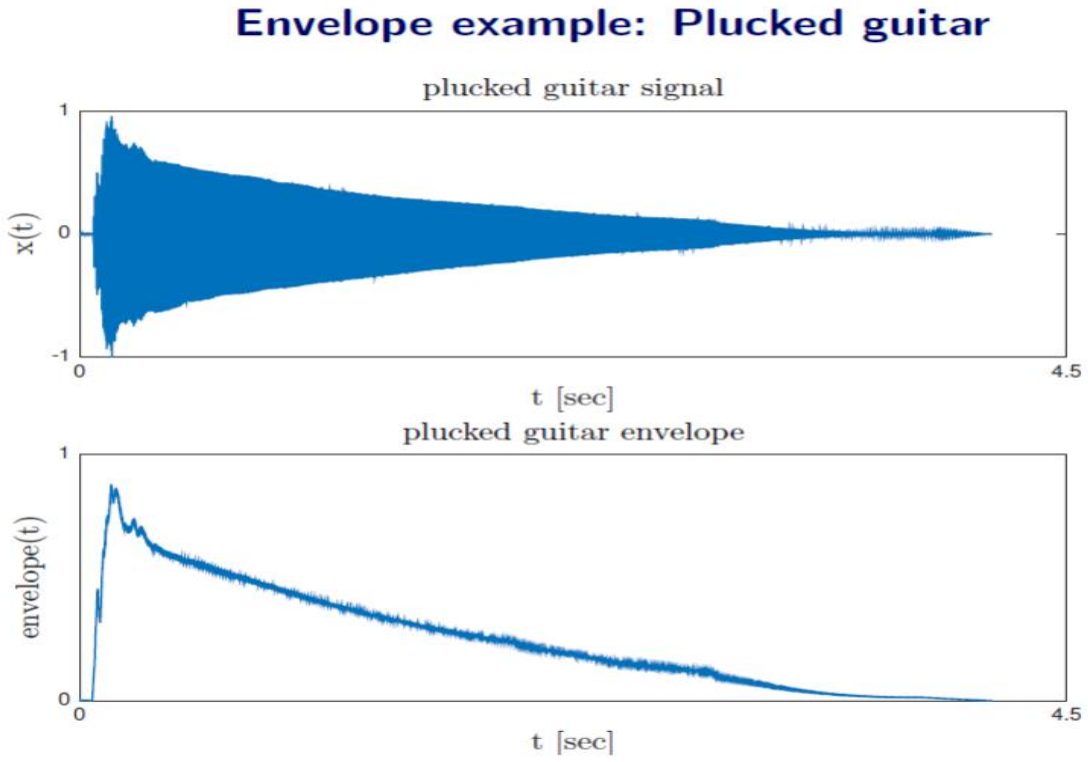
- A formula for THD is: [wiki]

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

- 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.

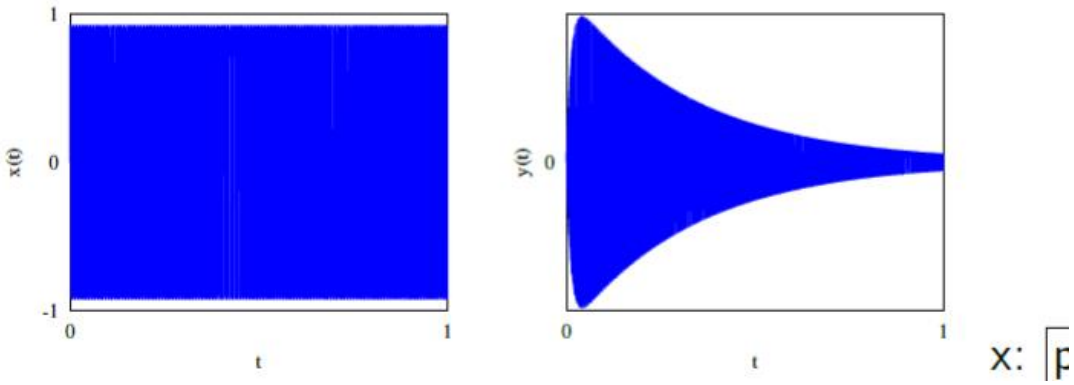
Envelope of a musical sound





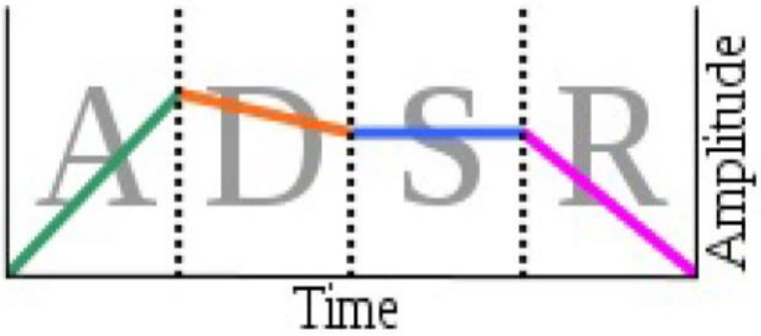
Envelope implementation

```
S = 44100;
N = 1 * S;
t = [0:N-1]/S;
c = 1 ./ [1:2:15]; % amplitudes
f = [1:2:15] * 494; % frequencies
x = 0;
for k=1:length(c)
        x = x + c(k) * sin(2 * pi * f(k) * t);
end
env = (1 - exp(-80*t)) .* exp(-3*t); % fast attack; slow decay
y = env .* x;
```



Attack, Decay, Sustain, Release (ADSR)

Programmable music synthesizers usually allow the user to control separately the time durations of these 4 components of the envelope.



[wiki]

- 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.

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
- Ample room for creativity and originality!

三、References

參考資料及網站

1. https://web.eecs.umich.edu/~fessler/course/100/l/l09-synth.pdf
2. https://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope
3. https://amath.colorado.edu/pub/matlab/music/