CS 583 – Computational Audio – Fall, 2021

Wayne Snyder
Computer Science Department
Boston University

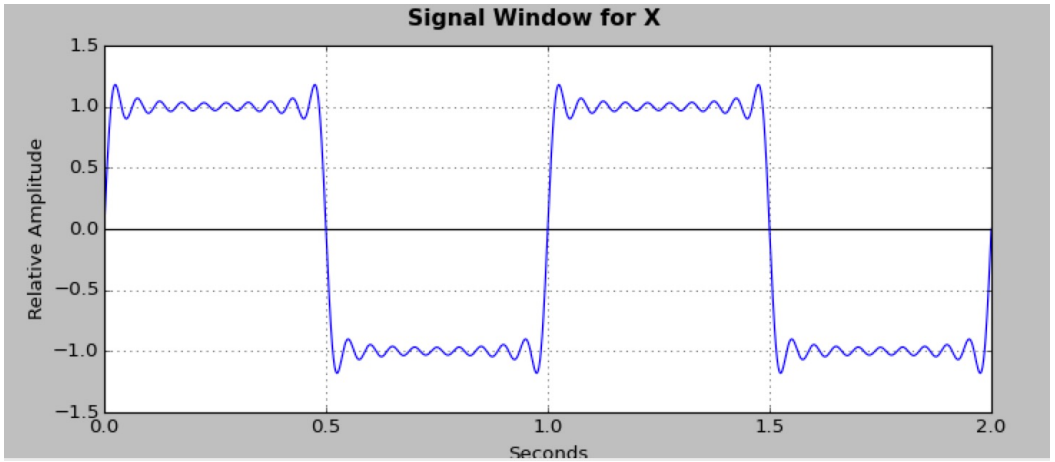
Lecture Three – Introduction to Audio Synthesis

Overview: Types of Audio Synthesis

Computer Science

Review: Fourier Analysis and Synthesis

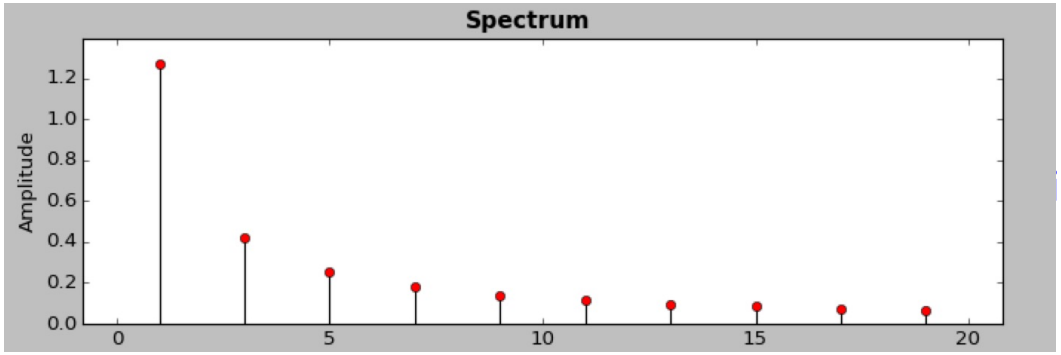
Fourier provided a way of looking at a signal in two equivalent ways, as a graph of amplitude of the signal vs time, the Time Domain, or as a graph of frequencies vs amplitudes (the Frequency Domain):



Time Domain

Fourier Synthesis using Inverse FFT

Fourier Analysis using FFT



Audio Synthesis

Computer Science

Audio Synthesis involves the modulation (control) of the various components of a music signal:

- Time
  - Duration; starting and stopping a note;
  - Rhythm
  - Tempo
- Pitch
  - Monophonic (one melody)
  - Homophonic (melody and chords)
  - Polyphonic (multiple simultaneous melodies)
- Amplitude
  - Amplitude envelope of a single note: onset, hold, decay
  - Dynamics: pp, pp, mf, ff, fff; crescendo, decrescendo, etc.
- Timbre (tone color, quality of sound)
  - Instantaneous spectrum
  - Spectral dynamics
    - Onset, duration, release
    - Spectral flux
    - Spectral rolloff
- Acoustics
  - Reverberation
  - Room acoustics
  - Stereo “sound stage”

Digital Audio: Synthesis

Computer Science

We can think of the **modular organization of a synthesis program** as a dataflow chart of the modules (e.g., functions or methods in Python or Java) that take certain parameters and produce certain outputs. **Data** used in these programs can be **artificial** (generated by the math in your code) or **real** (tables of measured samples), or combination of the two.

Roughly you can divide these modules into

Input Files: Instrument samples

Oscillators: Produce a waveform based on input parameters

Filters: Take an input waveform, modify it based on parameters, and output it to the next module

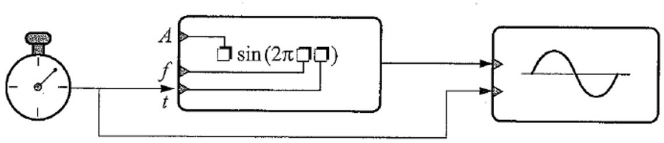
Miscellaneous: Timers, envelope generators, waveform "arithmetic" (**add**, invert, multiply, etc.)

Digital Audio: Synthesis -- Oscillators

Modular Organization of Synthesis Program

Your Python code can be viewed as a set of such modules, each module being a function.

Time is generated by a **for** loop and so it is easy to build an **oscillator for waveforms**:




```
X = np.zeros(N)

for k in range(len(X)):

    X[k] = A * np.sin( 2 * pi * f * k / SR + phi )
```

Notes:

- **f** is in Hz (cycles/second) and so must be converted to radians/sec by multiplying by 2π ;
- Time is in seconds, so **k / SR** gives the time stamp of the kth sample.

Digital Audio: Synthesis

Doing this for multiple frequencies results in a realistic additive synthesis program in the frequency domain:

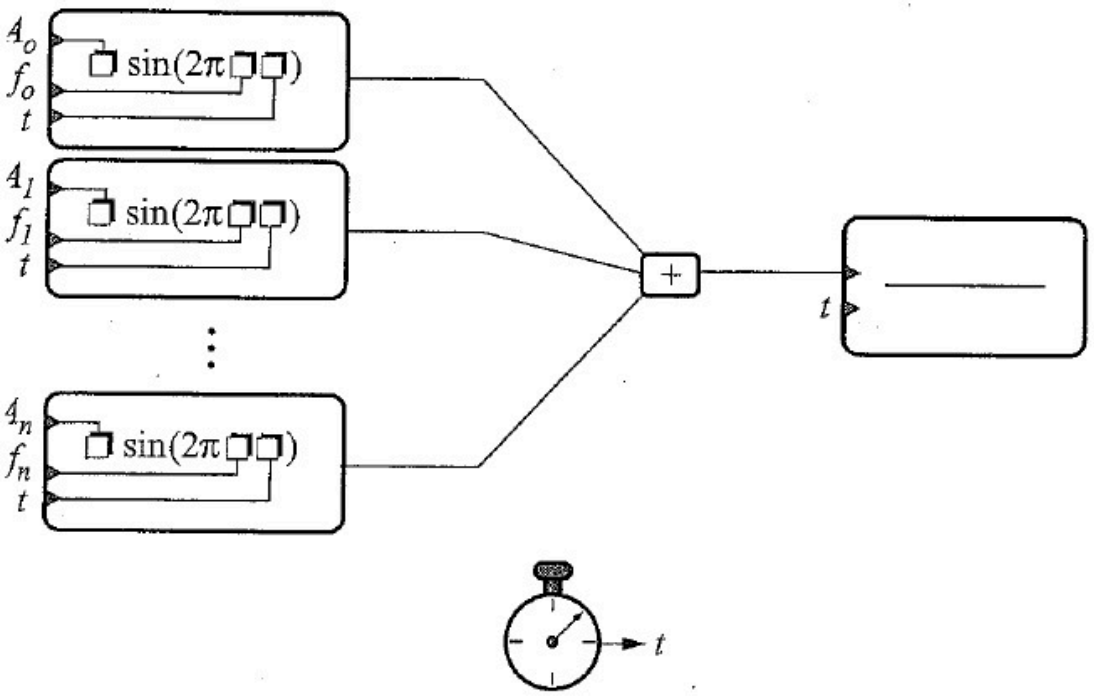


Figure 9.9
Additive synthesis.

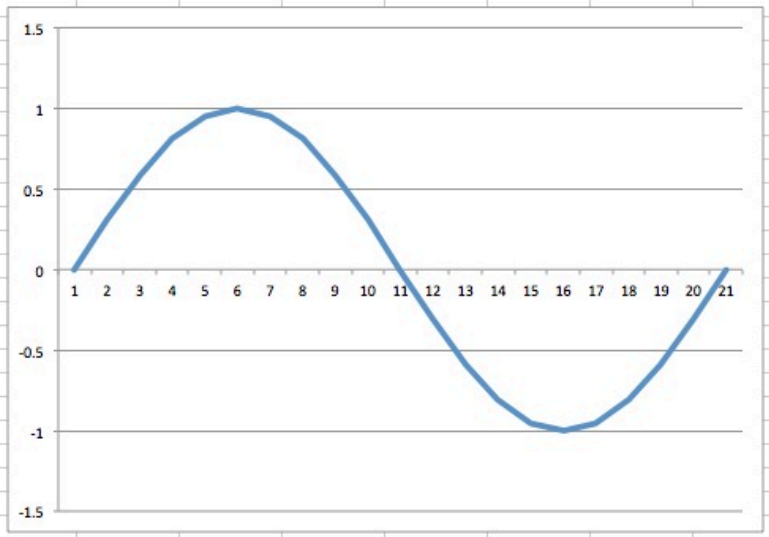
Digital Audio: Wavetables

Another source of inputs is real instrument recordings, which are manipulated to obtain the appropriate pitch and amplitude. These are simply signal files.

This is called by various names, including “wavetables.”

For example, you don’t need to call the sin() function, since you can simply store the values in an array and recall them when needed:

Time	Amplitude
0	0
0.05	0.30901699
0.1	0.58778525
0.15	0.80901699
0.2	0.95105652
0.25	1
0.3	0.95105652
0.35	0.80901699
0.4	0.58778525
0.45	0.30901699
0.5	5.666E-16
0.55	-0.309017
0.6	-0.5877853
0.65	-0.809017
0.7	-0.9510565
0.75	-1
0.8	-0.9510565
0.85	-0.809017
0.9	-0.5877853
0.95	-0.309017
1	1.5313E-15



Digital Audio: Wavetables

To obtain realistic instrument sounds, you can store a recording of a real instrument in a table, and lookup the values; you can change the frequency using interpolation to “upsample” or “downsample”:

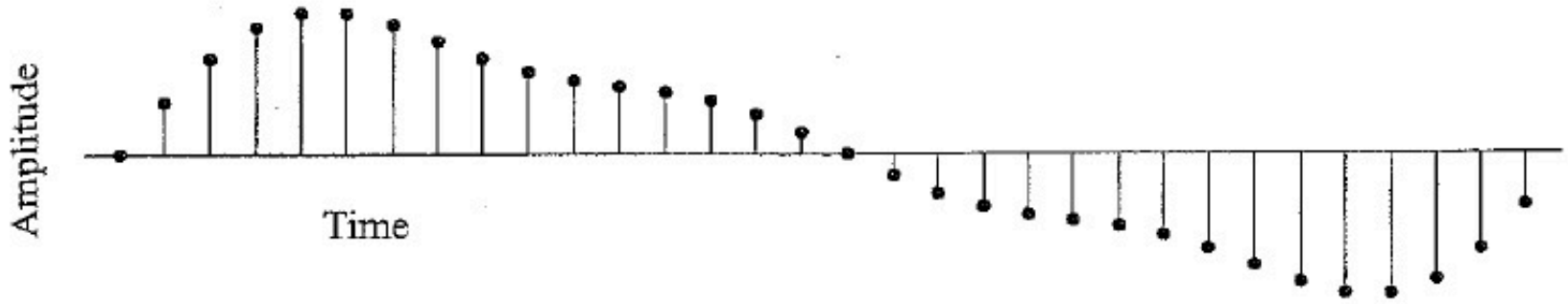


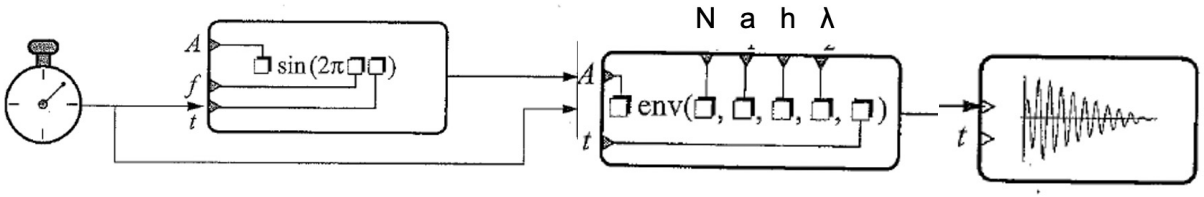
Figure 9.16
Sampled wave table.

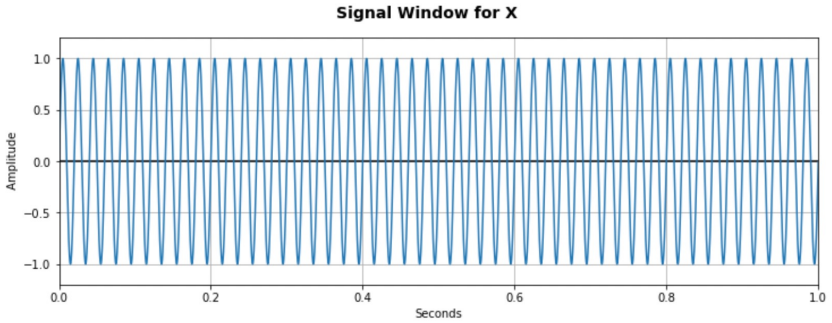
Time	Amplitude
0	0
0.05	0.30901699
0.1	0.58778525
0.15	0.80901699
0.2	0.95105652
0.25	1
0.3	0.95105652
0.35	0.80901699
0.4	0.58778525
0.45	0.30901699
0.5	5.666E-16
0.55	-0.309017
0.6	-0.5877853
0.65	-0.809017
0.7	-0.9510565
0.75	-1
0.8	-0.9510565
0.85	-0.809017
0.9	-0.5877853
0.95	-0.309017
1	1.5313E-15

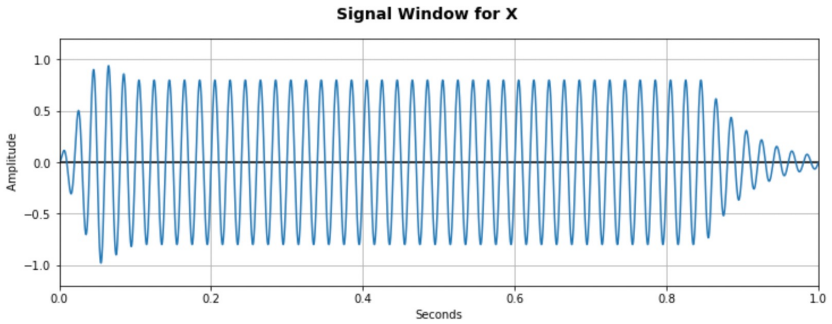
Digital Audio: Amplitude Modulation

Modular Organization of Synthesis Program

A very typical filter module is one that shapes the amplitude envelope of the signal:

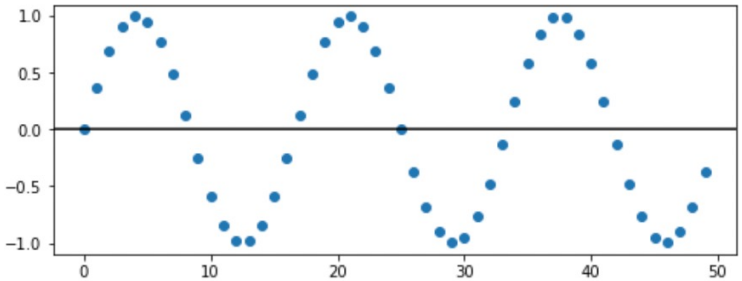




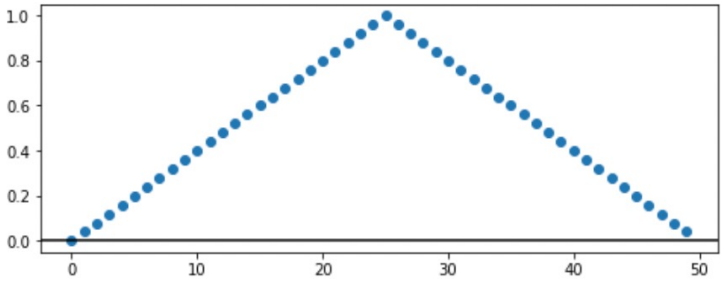


Digital Audio: Amplitude Modulation

In Python this is relatively simple: We create a “signal” consisting of the scaling factors at each sample, and then just multiply the two signals; here is an example of a “swell” envelope:

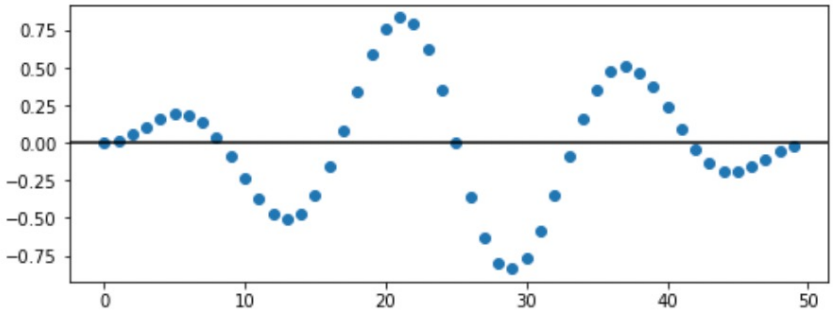


X



Y

*



X * Y

Digital Audio: Amplitude Modulation

Computer Science

Creating Amplitude Envelope Signals:

We want to model three aspects of amplitude envelopes for individual notes:

Attack: The inception of the tone from 0 amplitude to its maximum amplitude (say, 1.0), while the medium is absorbing energy.

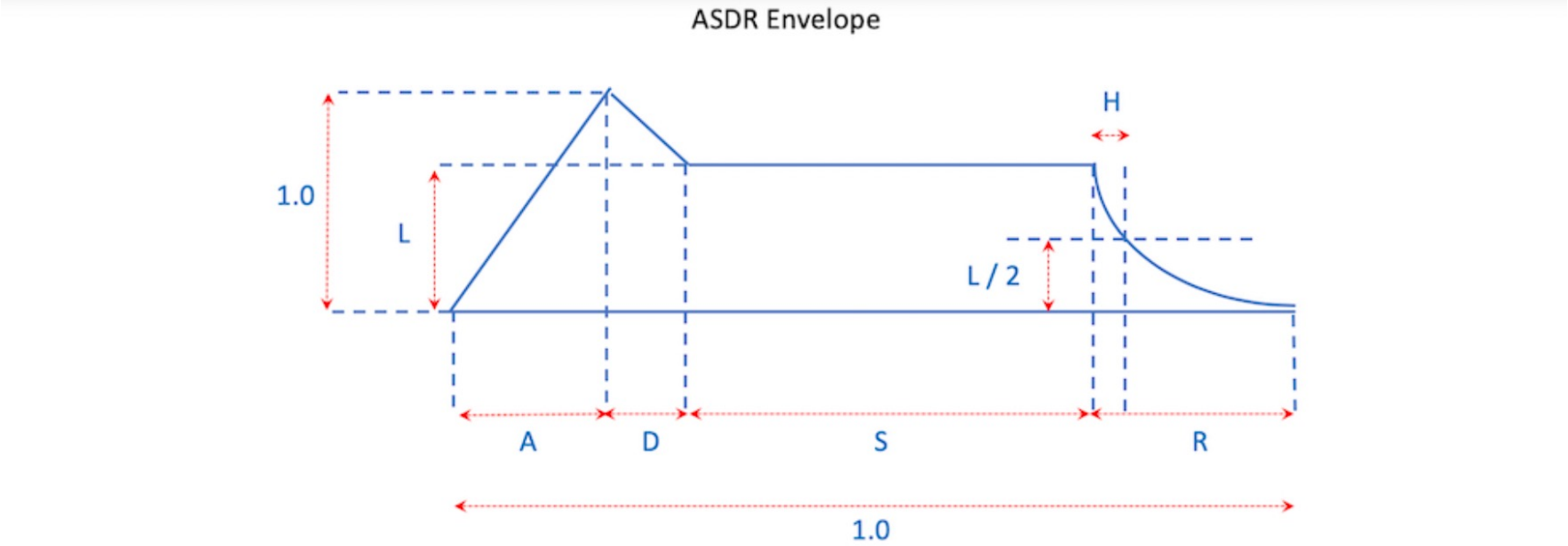
Hold: The period when the tone is held at maximum amplitude (may be 0)

Decay: The dying away of the tone as energy is dissipated

This does not model systems very well (such as wind or string instruments) where tone is produced by continuous effort, but is a good first approximation.

Digital Audio: Amplitude Modulation

ASDR Envelopes are a common way to specify amplitude envelopes:



- A, D, S, R = relative length of each phase as a percentage of the whole envelope;
- L = relative level during sustain phase, as percentage of maximum, e.g., 0.5 is sustain at half max value; and
- H = relative half-life of exponential decay during R phase (time for amplitude to be reduced by half), as a percentage of the whole envelope.

Note carefully that all the units are relative values in the range $[0..1]$ and that we must have $A + D + S + R = 1.0$.