



# NumPy as Synth Engine

MARCH 2026

6 min read • 1,409 words

---

There are zero audio files in [PyTheory](#). No samples. No recordings. Not one byte of pre-recorded sound anywhere in the repository.

|

You can see [the code that generated this song](#).

Every sound you hear — every plucked sitar string, every tabla stroke, every tambora drone — is computed at runtime from math. Sine waves, noise, filters, envelopes. NumPy arrays are the synth engine.

I still can't quite believe how well this works.

**A note:** I built the synthesis engine with help from Claude. I'm learning DSP as I go.

## Karplus-Strong

In 1983, Kevin Karplus and Alex Strong figured out how to generate plucked string sounds. The algorithm is almost offensively simple:

1. Fill a buffer with random noise, one pitch period long.
2. Loop through it. Average each sample with the next one.
3. That's it.

```

import numpy

SAMPLE_RATE = 44_100

def plucked_string(hz, n_samples=SAMPLE_RATE):
    """Karplus-Strong: a buffer of noise that averages itself into music."""
    period = int(SAMPLE_RATE / hz)
    buf = numpy.random.uniform(-1.0, 1.0, period)

    out = numpy.zeros(n_samples)
    for i in range(n_samples):
        out[i] = buf[i % period]
        next_idx = (i + 1) % period
        buf[i % period] = 0.5 * (buf[i % period] + buf[next_idx]) * 0.999

    return out

```

The averaging is a lowpass filter. High harmonics die off faster than low ones — exactly what a real vibrating string does. The `0.999` controls sustain.

That's a plucked string. From a loop and an average.

The fun part: the same algorithm with different parameters makes completely different instruments. An acoustic guitar adds body resonance — bandpass filters at the frequencies where a wooden body naturally vibrates (110 Hz air cavity, 250 Hz top plate, 500 Hz back):

```

for center, bw, gain in [(110, 60, 0.4), (250, 80, 0.3), (500, 120, 0.2)]:
    bp, ap = scipy.signal.butter(2, [center - bw, center + bw],
                                  btype="band", fs=SAMPLE_RATE)
    resonances += scipy.signal.lfilter(bp, ap, out) * gain

```

An electric guitar replaces body resonance with a pickup simulation — a comb filter at 1/4 of the string length that cancels the 4th harmonic and boosts the 2nd. That's the midrange honk.

A ukulele has a smaller body (resonances at 350, 700, 1200 Hz) and softer initial noise for nylon strings.

Same core algorithm. Different physics around it. Different instrument out the other end.

## The Tabla

This is where things got wild for me.

A tabla is two drums — the dayan (small, wooden, right hand) and the bayan (larger, copper, left hand). The syahi, that circular black paste on the drumhead, gives it those characteristic ringing harmonics. These sounds have been refined for centuries by Indian classical musicians, and they're nothing like Western percussion.

I modeled six strokes. Each one layers multiple physical components:

- **Na**: sharp rim strike on the dayan. Four elements — goatskin membrane thump (bandpass noise, 200-800 Hz), wooden shell resonance (800 Hz sine), syahi ring (three harmonics at 330, 680, 1050 Hz with independent decays), and finger attack transient:

```

def _synth_tabla_na(n_samples):
    t = numpy.arange(n_samples, dtype=numpy.float32) / SAMPLE_RATE

    # Goatskin membrane thump – bandpass filtered noise
    thump_raw = noise(int(SAMPLE_RATE * 0.05))
    bl, al = scipy.signal.butter(2, [200, 800], btype="band", fs=SAMPLE_RATE)
    thump = scipy.signal.lfilter(bl, al, thump_raw)
    thump *= exp_decay(len(thump), 40) * 0.8

    # Wooden shell resonance
    wood = numpy.sin(2 * numpy.pi * 800 * t[:len(thump)]) * exp_decay(len(thump), 50) *
    thump += wood

    # Syahi ring – three harmonics with independent decay
    ring = numpy.sin(2 * numpy.pi * 330 * t) * exp_decay(n_samples, 9) * 0.6
    ring2 = numpy.sin(2 * numpy.pi * 680 * t) * exp_decay(n_samples, 12) * 0.3
    ring3 = numpy.sin(2 * numpy.pi * 1050 * t) * exp_decay(n_samples, 16) * 0.15

    # Sharp finger attack
    click = noise(100) * exp_decay(100, 250) * 0.7

    # Layer everything
    result = ring + ring2 + ring3
    result[:len(thump)] += thump
    result[:100] += click
    return numpy.tanh(result * 1.4)

```

- **Tin:** open ring. Same architecture, different filter ranges, slower decay. The full stroke lets the membrane vibrate freely.
- **Ge:** the deep bayan. Five components — membrane thud (40-250 Hz), copper shell resonance (120 Hz), a pitch-sweeping body, sub boom (40 Hz), and palm attack. The pitch sweep is the wildest part:

```

# Hand modulates the membrane
freq = 55 + 100 * numpy.exp(-10 * t)
phase = 2 * numpy.pi * numpy.cumsum(freq) / SAMPLE_RATE
body = numpy.sin(phase) * exp_decay(n_samples, 5) * 0.7

```

Three lines modeling a palm pressing a goatskin membrane over a copper shell. The frequency starts at 55 Hz and exponentially rises to 155 Hz because the hand changes the effective pitch. That's a technique unique to tabla.

- **Dha**: both drums at once. Na + Ge summed through `tanh`.
- **Tit**: rapid-fire finger tap. Sixty milliseconds max. Mostly attack.
- **Ke**: muted bayan slap. Dead thud, no ring. Eighty milliseconds.

The membrane is bandpass-filtered noise because that's what a vibrating membrane sounds like. The wooden shell and the copper shell are different resonance frequencies. The syahi creates specific harmonics because the added mass changes the membrane's vibrational modes. These are approximations of physics, not approximations of recordings.

## Cross-Choking

When you hit a djembe slap, it dampens any ringing bass or tone. Your hand is on the skin — the previous vibration dies. This is how real drums work.

```
_CHOKE_GROUPS = {
    DJEMBE_BASS: (DJEMBE_TONE, DJEMBE_SLAP),
    DJEMBE_TONE: (DJEMBE_BASS, DJEMBE_SLAP),
    DJEMBE_SLAP: (DJEMBE_BASS, DJEMBE_TONE),
    CLOSED_HAT: (OPEN_HAT, ),
    PEDAL_HAT: (OPEN_HAT, ),
    CAJON_SLAP: (CAJON_BASS, ),
    DOUMBЕК_TEK: (DOUMBЕК_DUM, ),
}
```

When a new hit lands on a choke target, a 2-4ms fade-out kills whatever was ringing:

```
fade = numpy.linspace(1.0, 0.0, fade_len)
part_stereo[start - fade_len : start] *= fade
```

Without this, a fast bass-slap-bass pattern sounds like three separate events. With it, each hit silences the last. Same for hi-hats — closed chokes open. A few milliseconds of linear fade, and suddenly the drums sound like someone is playing them.

## The Hammond Organ

After all that physical modeling, this one is almost a palate cleanser. A Hammond B3 has nine drawbars, each adding a sine wave at a specific harmonic:

```
def hammond_wave(hz, peak, n_samples):
    t = numpy.arange(n_samples, dtype=numpy.float64) / SAMPLE_RATE

    wave = (
        numpy.sin(2 * numpy.pi * hz * t) * 1.0           # 16'
        + numpy.sin(2 * numpy.pi * hz * 2 * t) * 0.8     # 8'
        + numpy.sin(2 * numpy.pi * hz * 3 * t) * 0.6     # 5 1/3'
        + numpy.sin(2 * numpy.pi * hz * 4 * t) * 0.5     # 4'
        + numpy.sin(2 * numpy.pi * hz * 5 * t) * 0.3     # 2 2/3'
        + numpy.sin(2 * numpy.pi * hz * 6 * t) * 0.25    # 2'
        + numpy.sin(2 * numpy.pi * hz * 8 * t) * 0.15    # 1 3/5'
    )

    wave /= 3.5
    return (peak * wave).astype(numpy.int16)
```

Seven sine waves. That's an organ. Change the weighting and you get a different registration — gospel, jazz, rock. Play a chord through it and anyone over thirty will know what it is.

## Historical Tunings

Here's something you can't do with sample libraries: play a Karplus-Strong string in Pythagorean temperament. Or a tabla in meantone. Or an organ tuned to just intonation.

Soundfonts are recordings locked to equal temperament. If you want to hear what a harpsichord sounded like in 1685, you're out of luck — the samples were recorded at fixed frequencies. Pitch-shifting distorts the timbre.

When everything is computed from math, the frequency is just a parameter. A string at 440 Hz and a string at 438.07 Hz (Pythagorean A4) use the same algorithm — the delay line is just slightly longer. The body resonance still works. The envelope still works. You're generating a new sound, not stretching an old one.

I didn't plan this advantage. But now I can hear what a Renaissance lute sounded like in its original tuning, and that's really fun.

## The Absurdity

All of this is Python. The same language people use to parse CSV files is generating tabla strokes with physically modeled goatskin membranes.

`numpy.sin` is the oscillator. `scipy.signal.butter` is the filter. `numpy.tanh` is the soft clipper.

There's a version of this where I'd use a C extension or a proper DSP framework or just sample libraries like a sensible person. It would sound better and run faster. But the code wouldn't read like physics. The bandpass filter from 200 to 800 Hz is the goatskin membrane. The pitch sweep is the hand pressing the drumhead. The comb filter is the magnetic pickup.

I've written about [why PyTheory matters to me](#) and [how it grew into a mini DAW](#). The synthesis layer is really fun.

---

**Update:** The album has been released! [Interpretations](#) — 24 tracks, all written in Python.