Building a Basic Synthesizer

Goal:

Press keys

Make sounds

F7
F8
F9

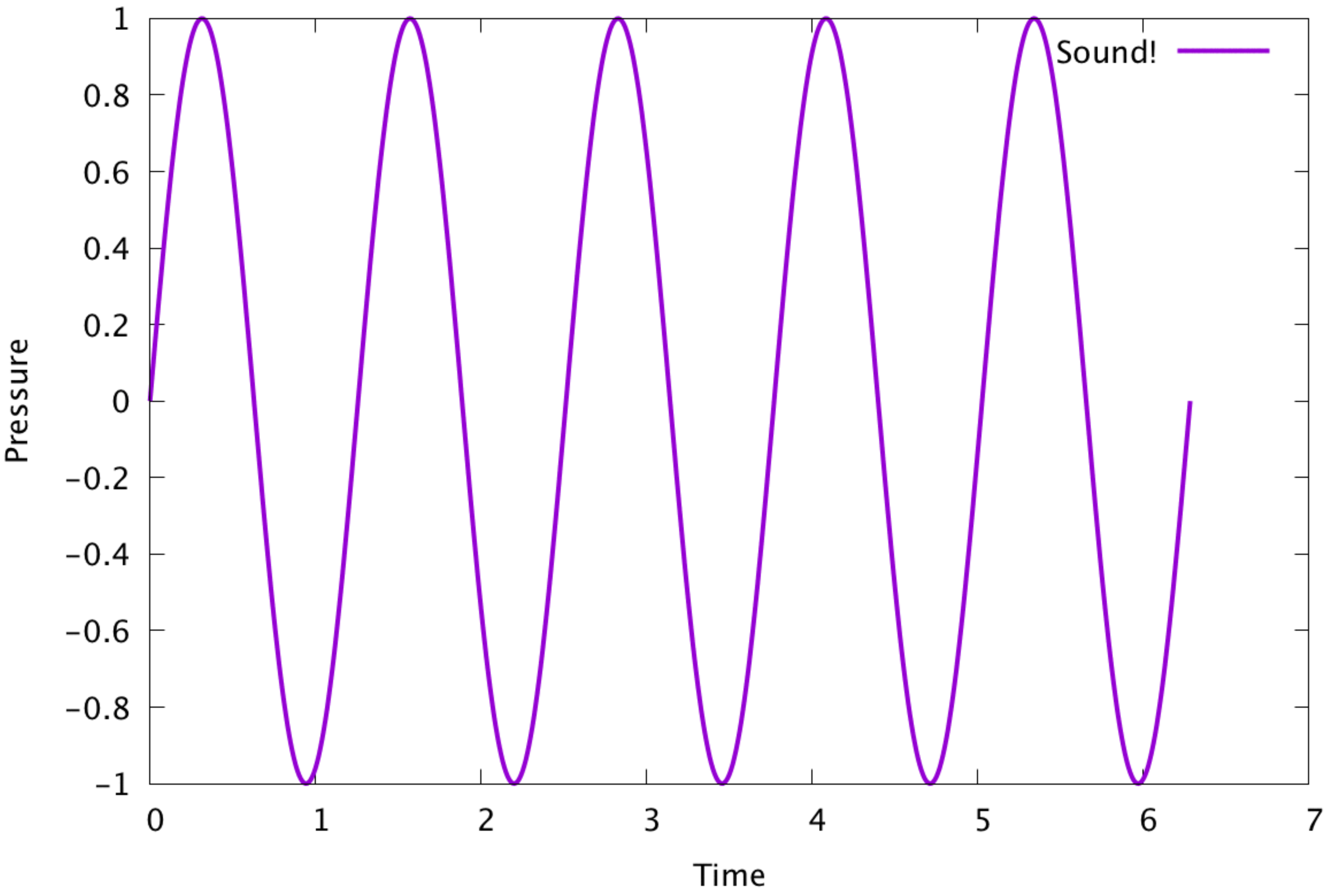
Goal:

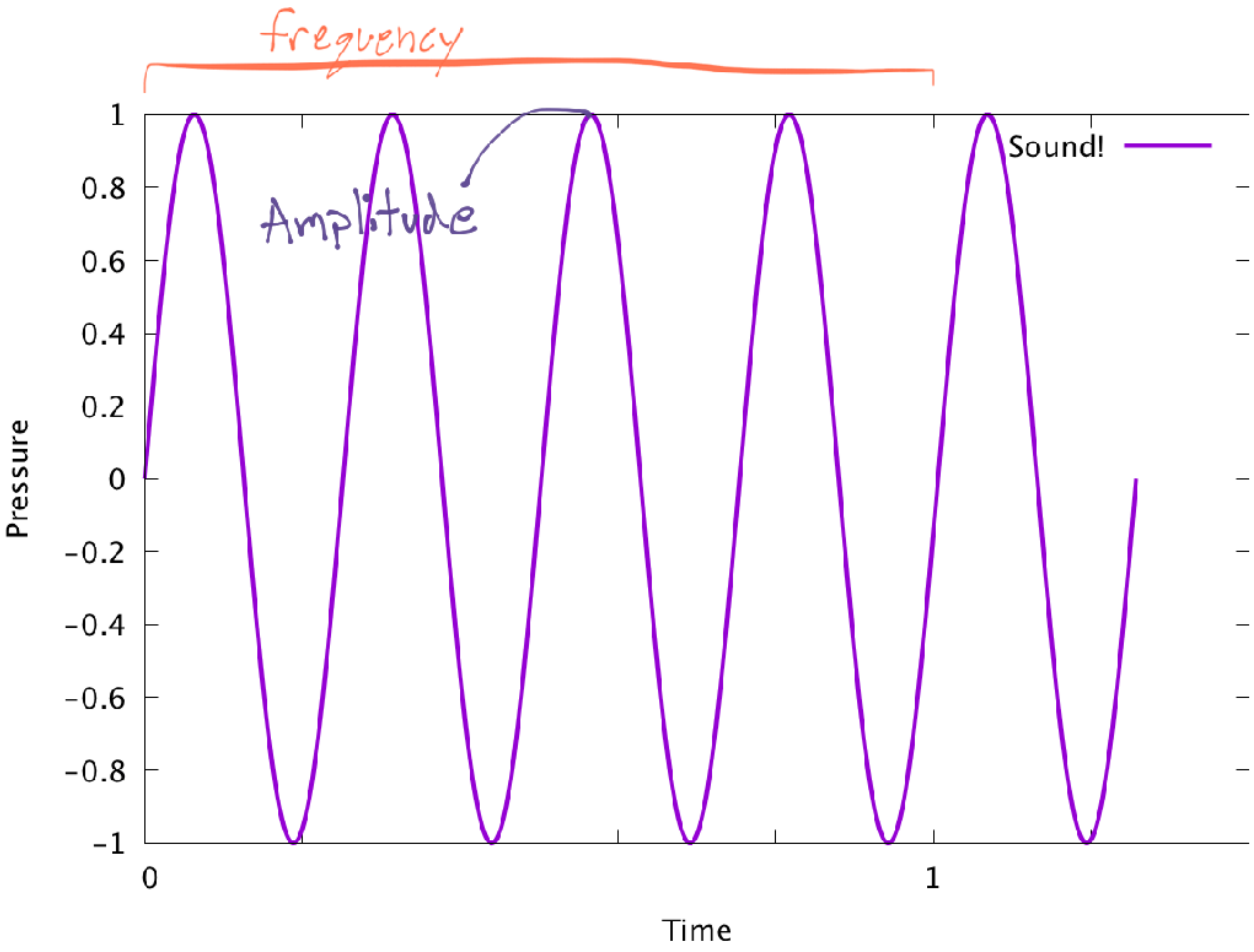
Press keys

~~Make~~ sounds

generate

Sound

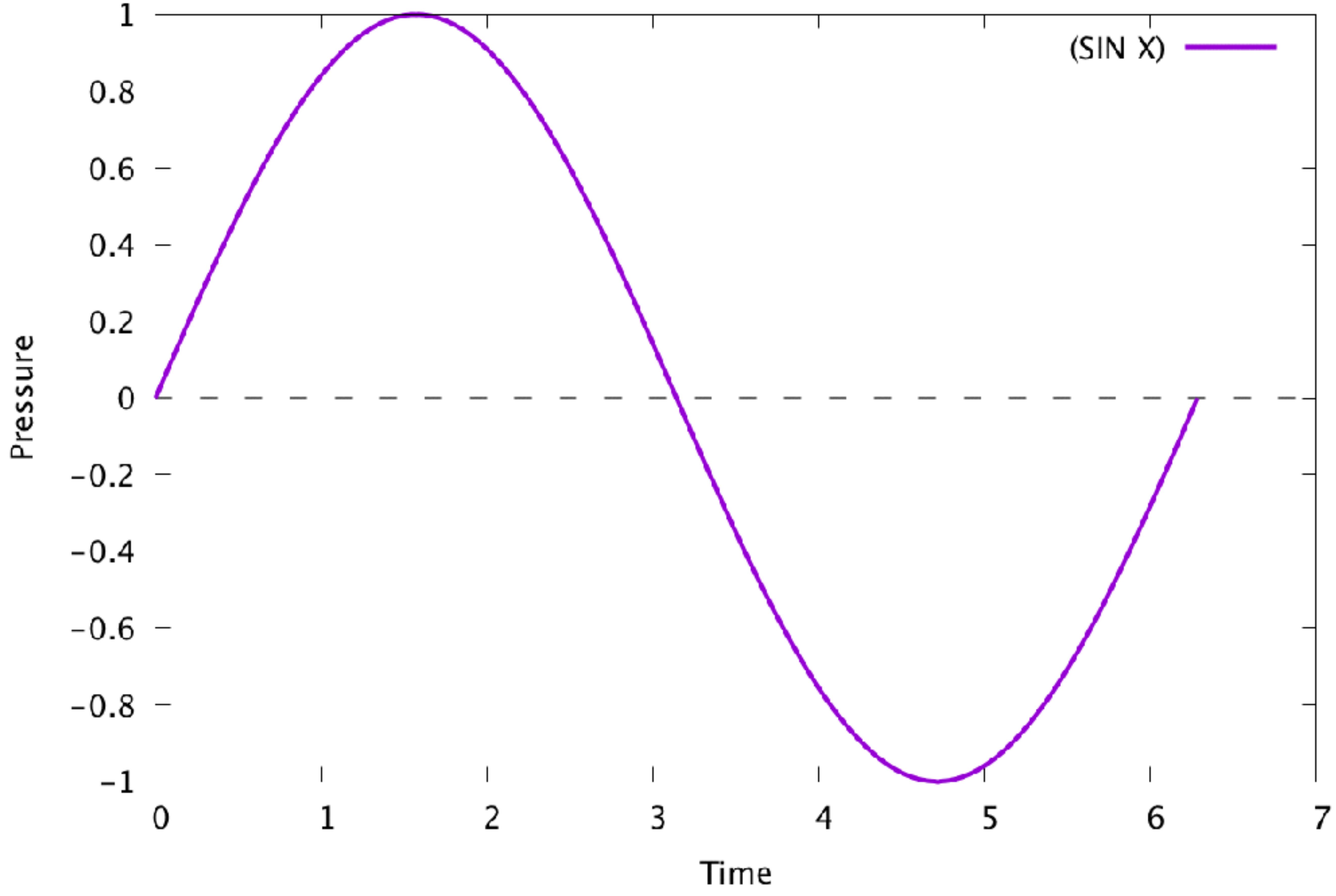

Sound!
Pressure
Time
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7

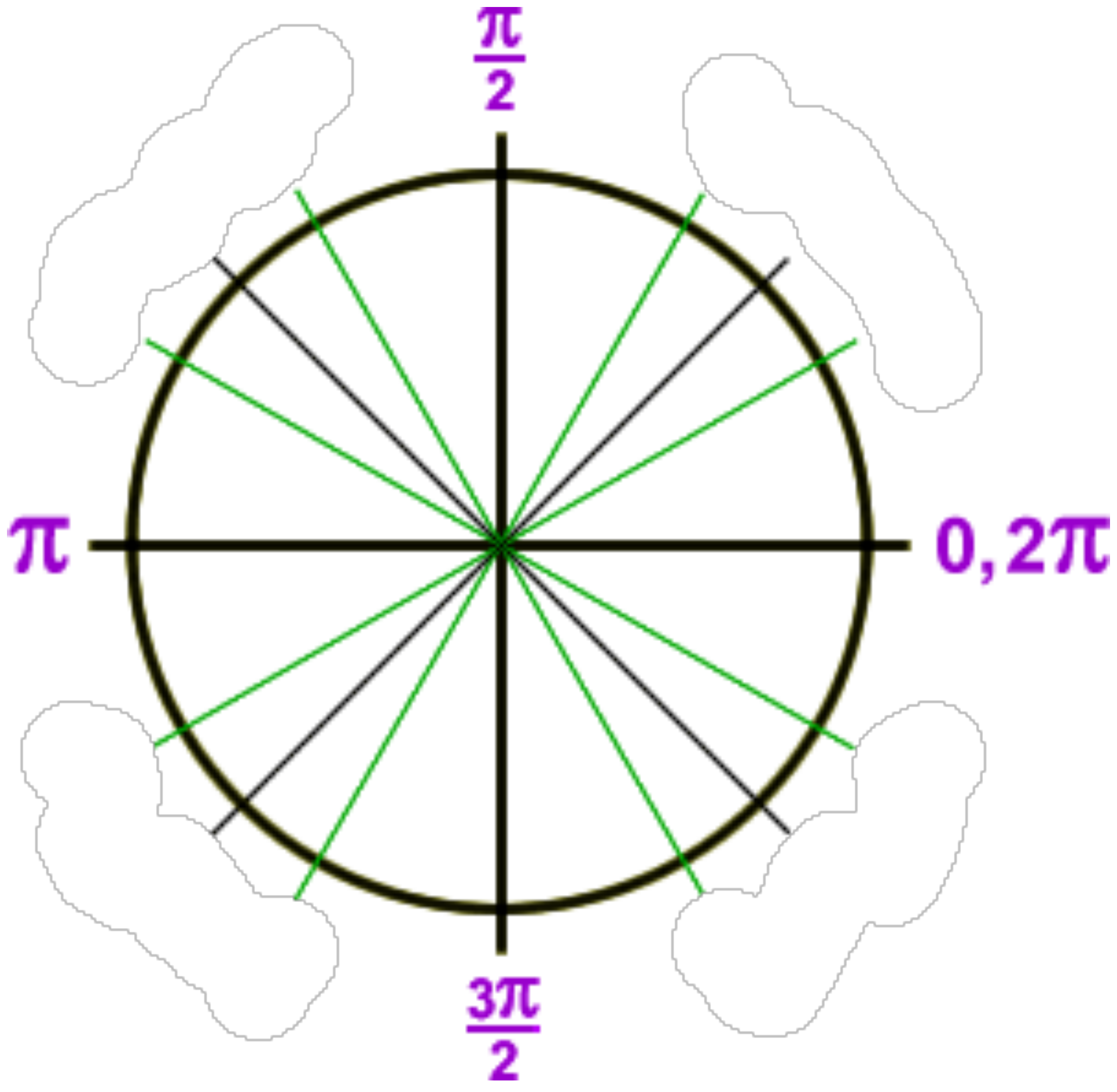


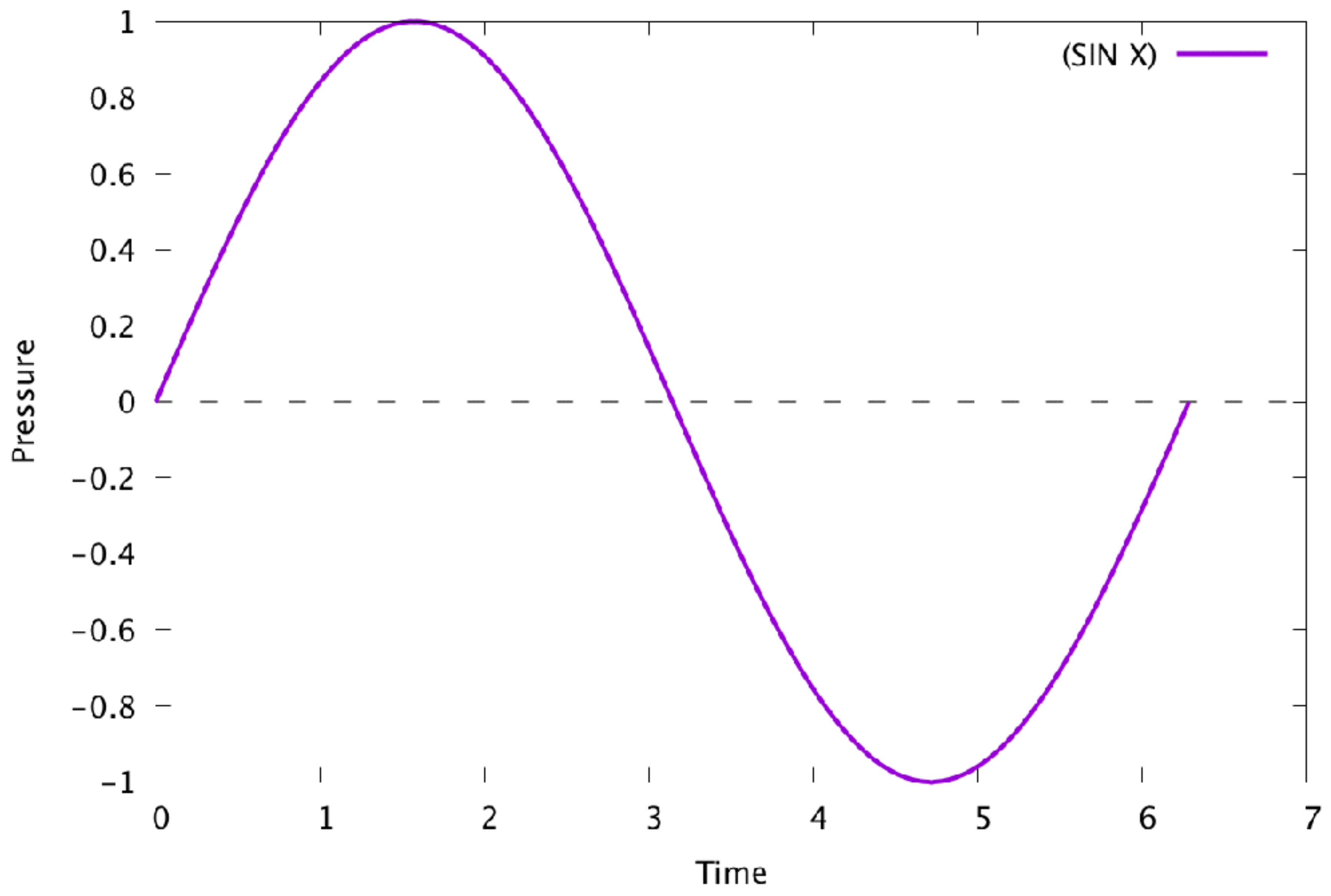
Frequency (Hz): The # of full cycles of a wave in a unit of time. In sound, the **pitch**.

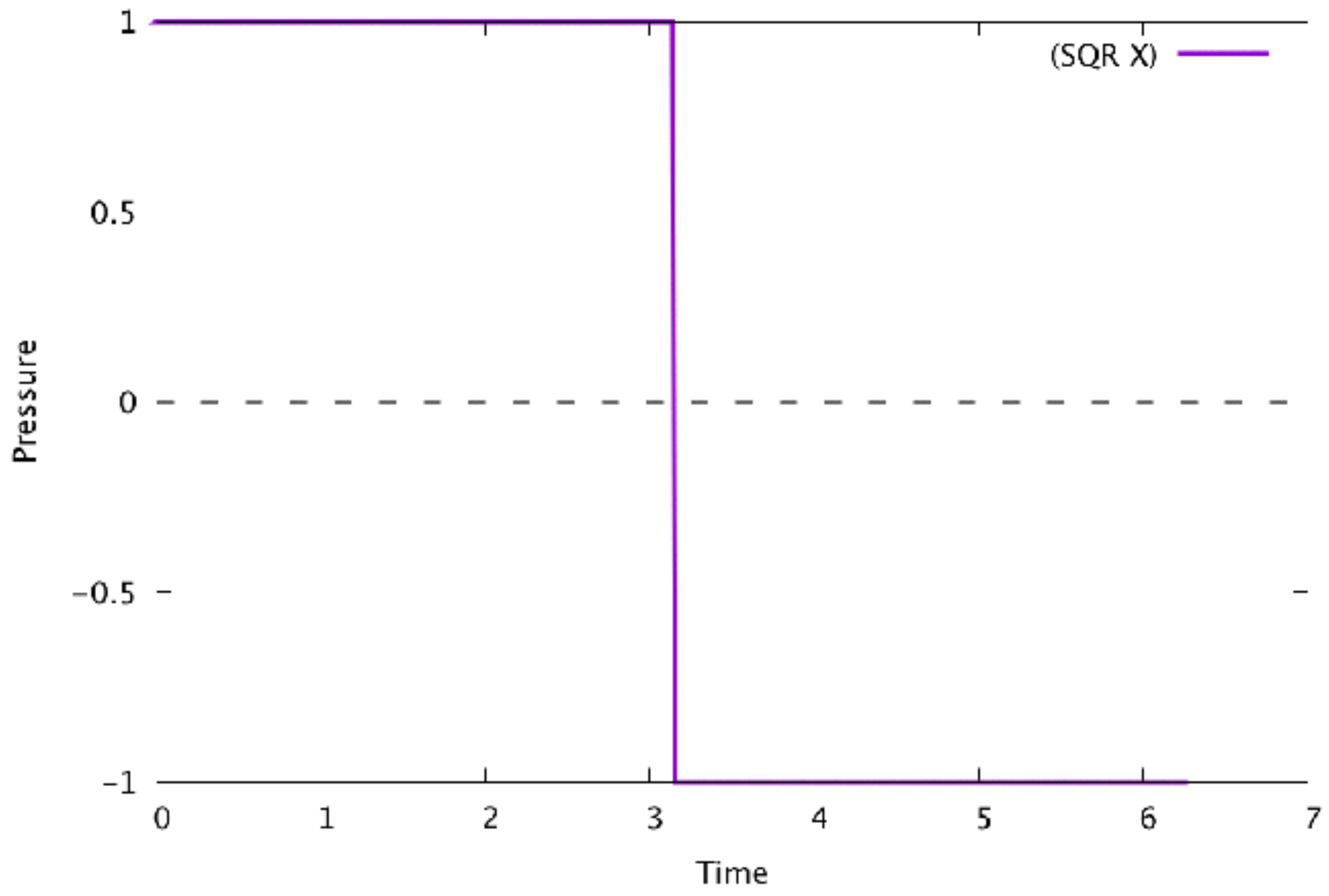
Amplitude: The maximum height of the wave. In sound, the **volume**.

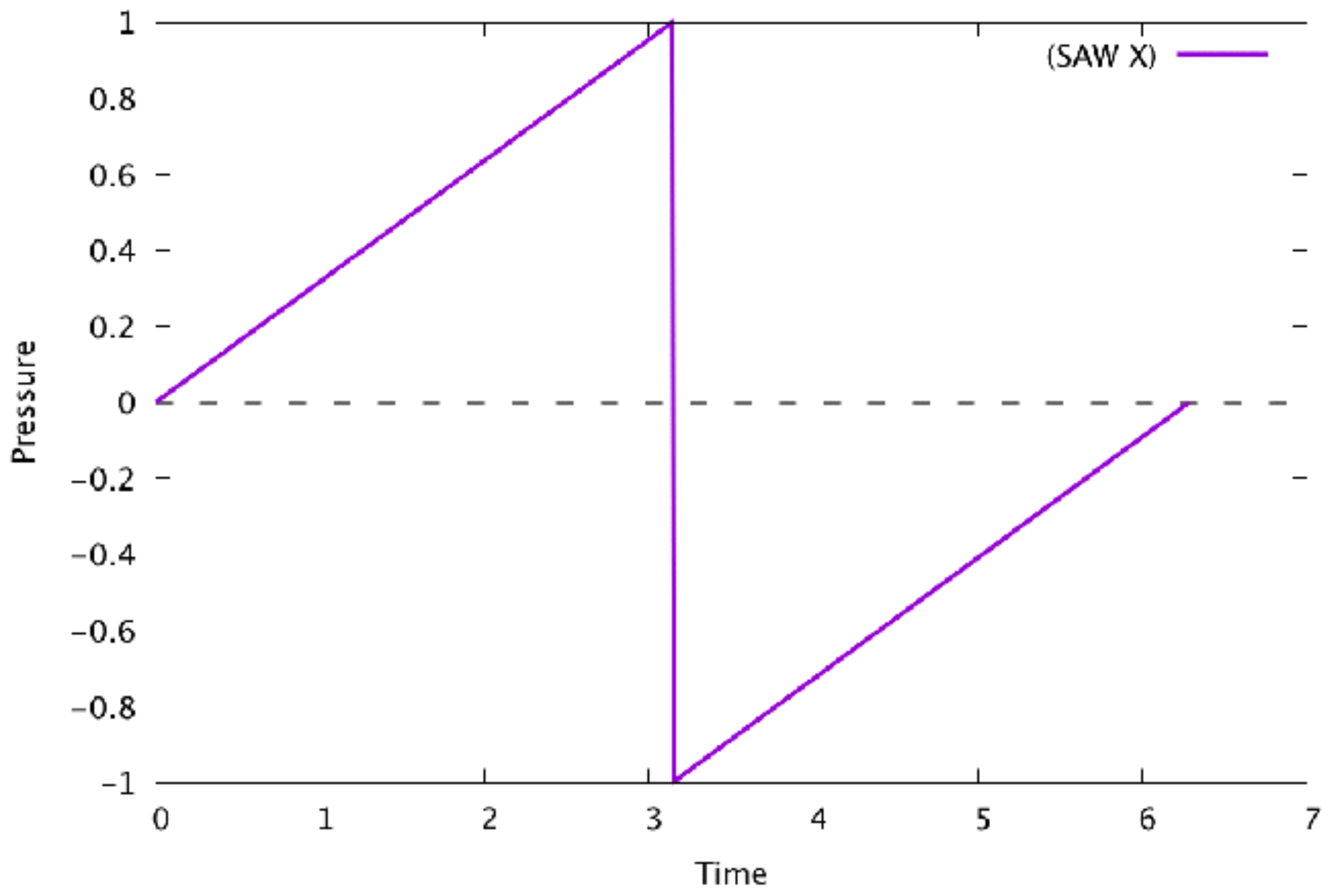
Trig Review

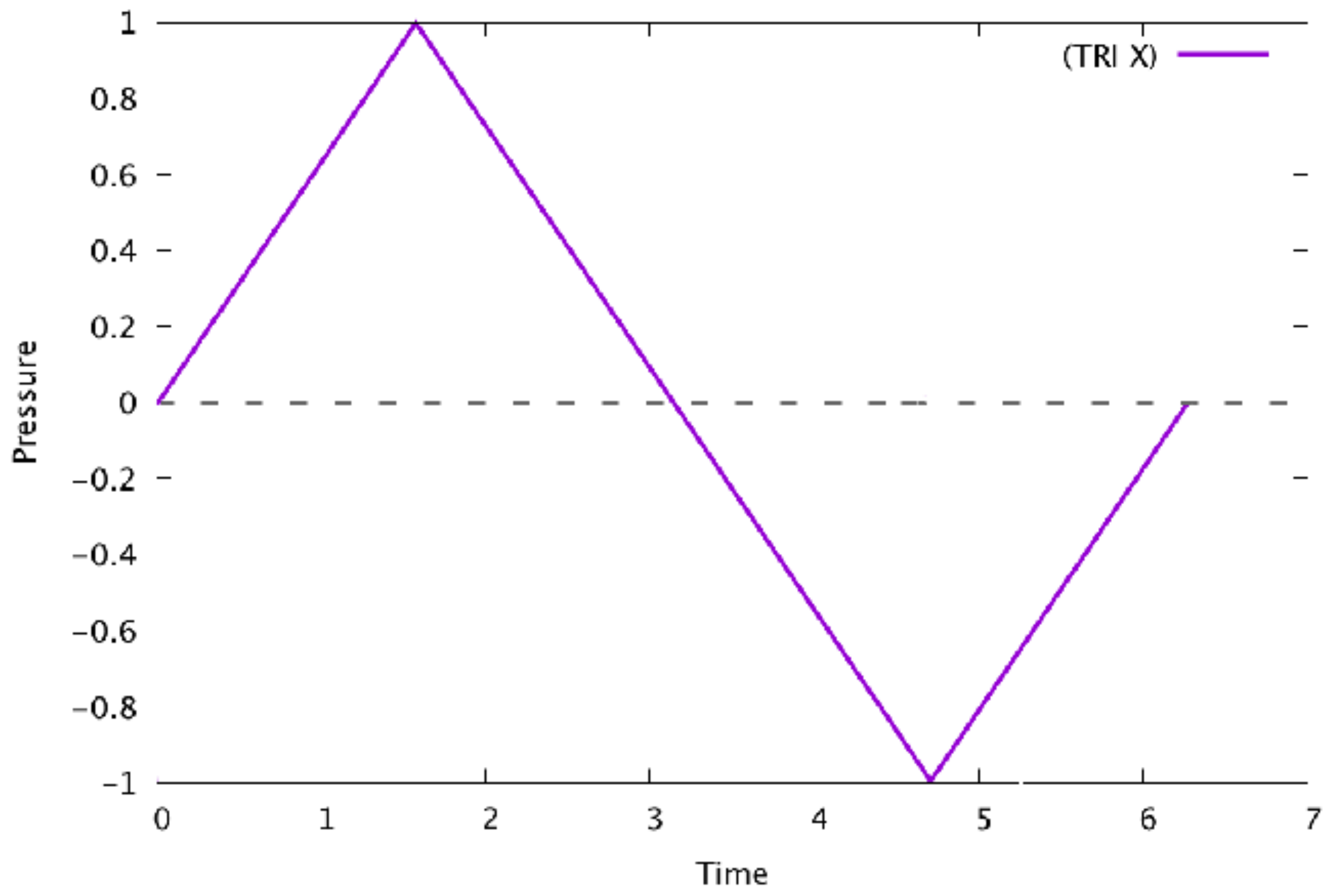

(SIN X)
Pressure
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7
Time


$\frac{\pi}{2}$
π
$0, 2\pi$
$\frac{3\pi}{2}$

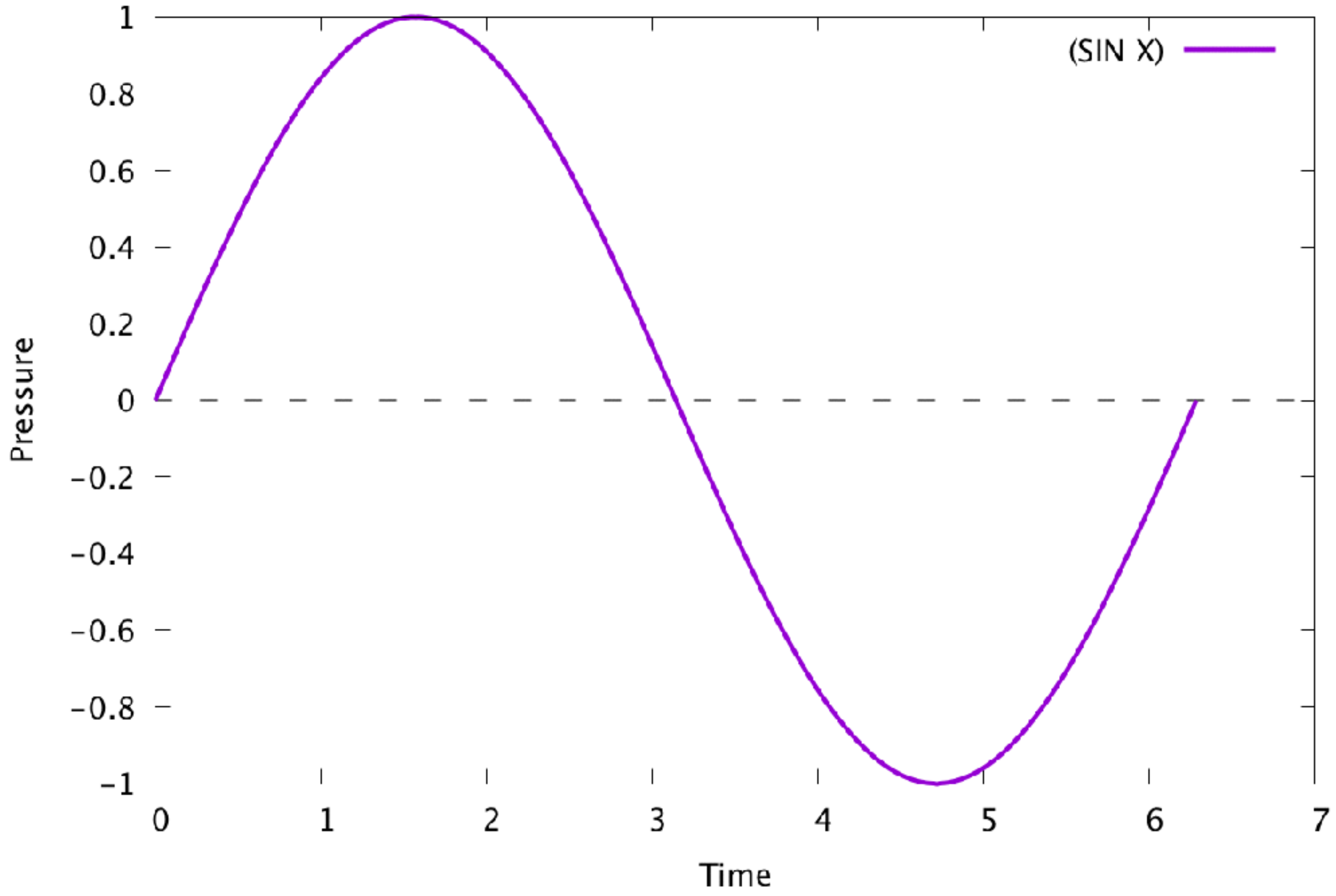

(SIN X)
Pressure
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7
Time


(SQR X)
Pressure
1
0.5
0
−0.5
−1
0
1
2
3
4
5
6
7
Time

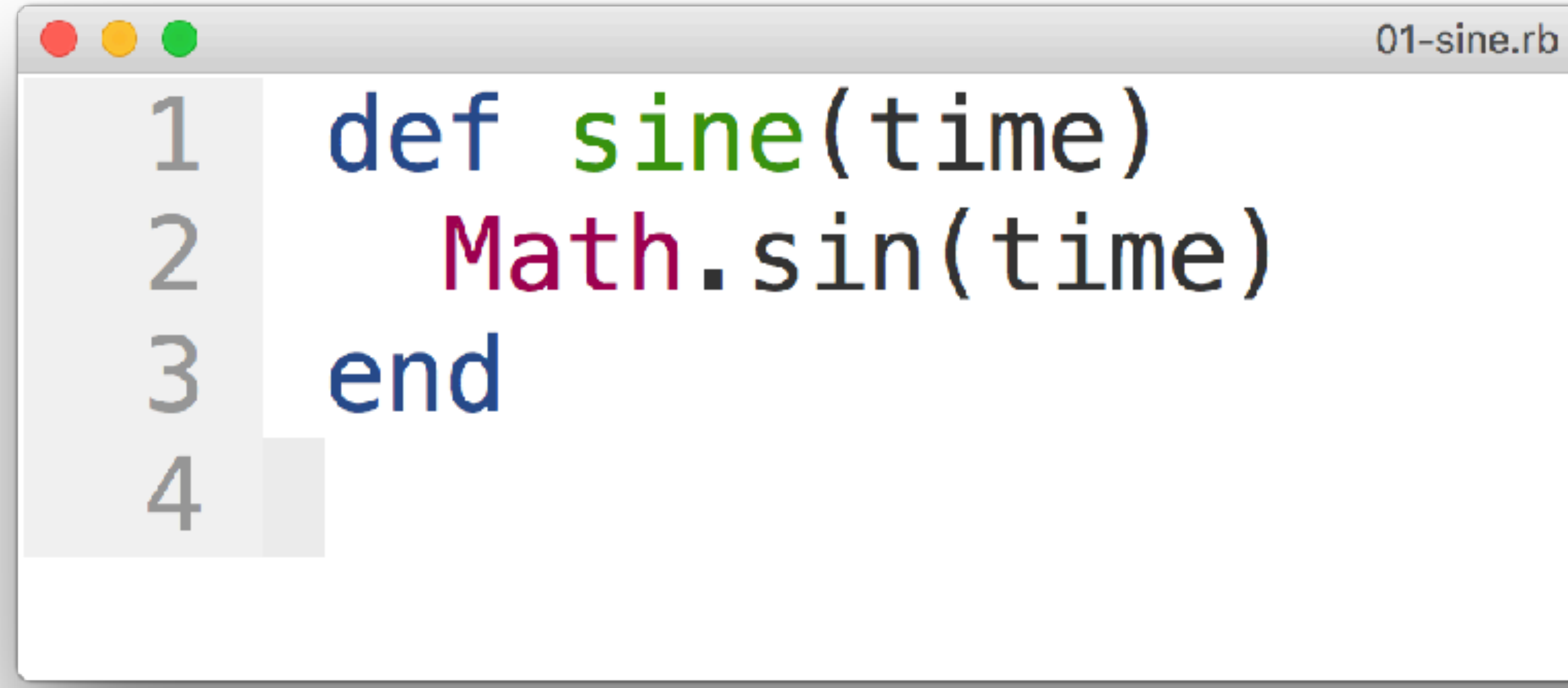

(SAW X)
Pressure
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7
Time


(TRI X)
Pressure
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
1
2
3
4
5
6
7
Time

Sine Wave

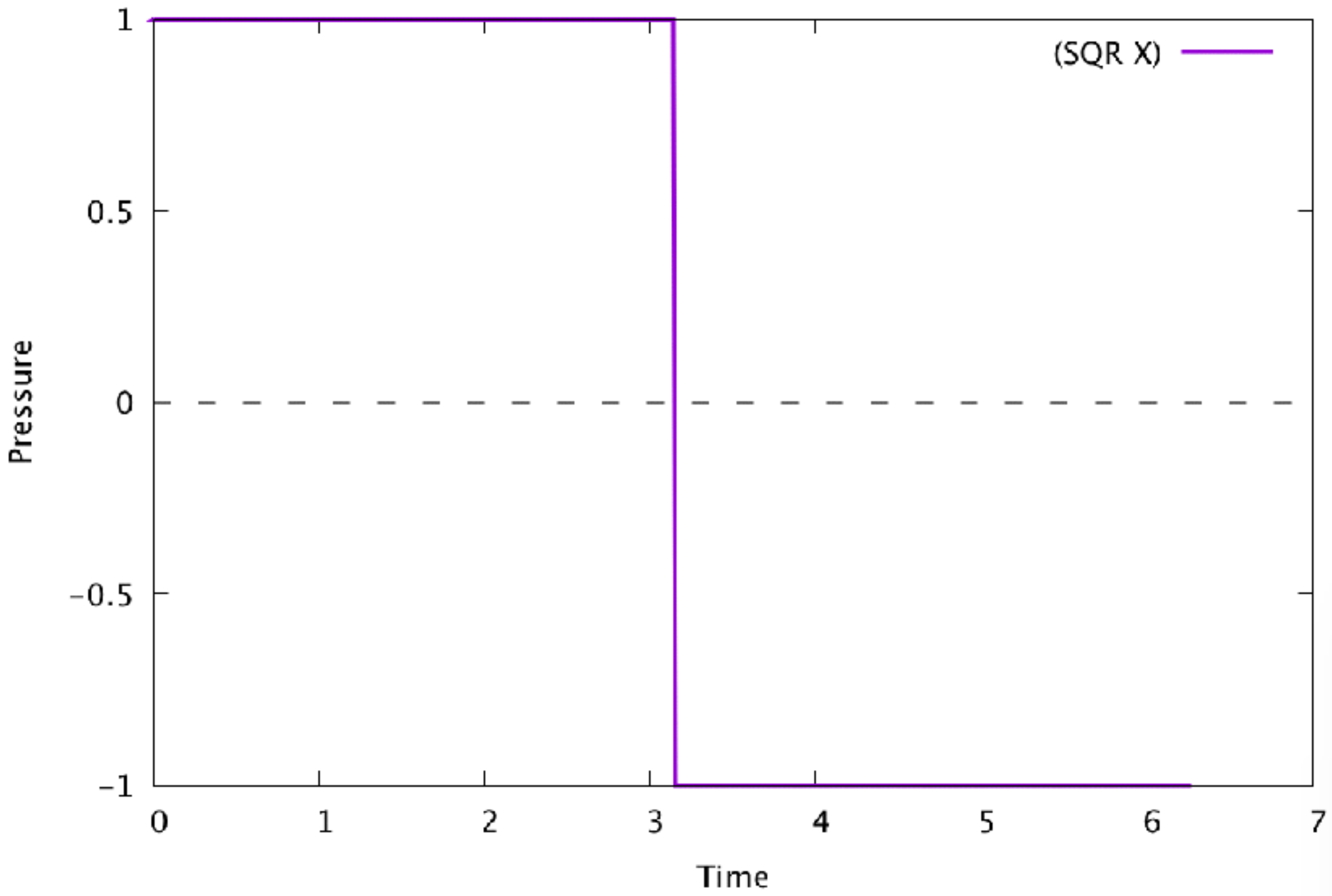


$$p = sin(t)$$

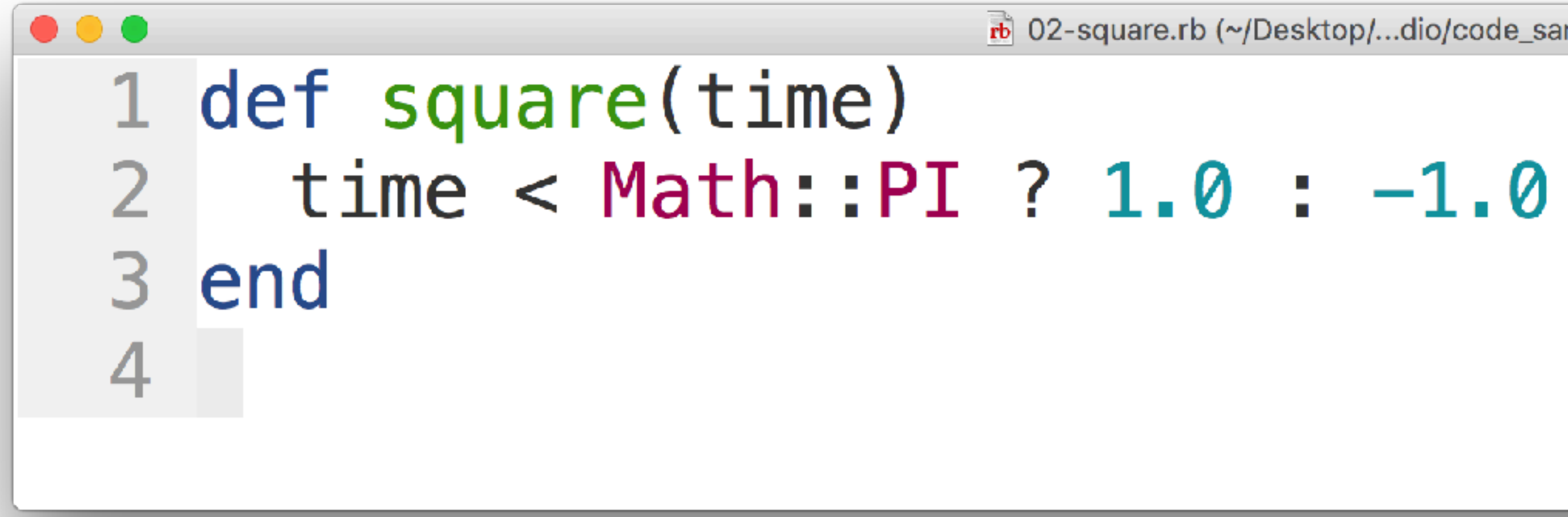


```ruby
def sine(time)
  Math.sin(time)
end

```

Square Wave

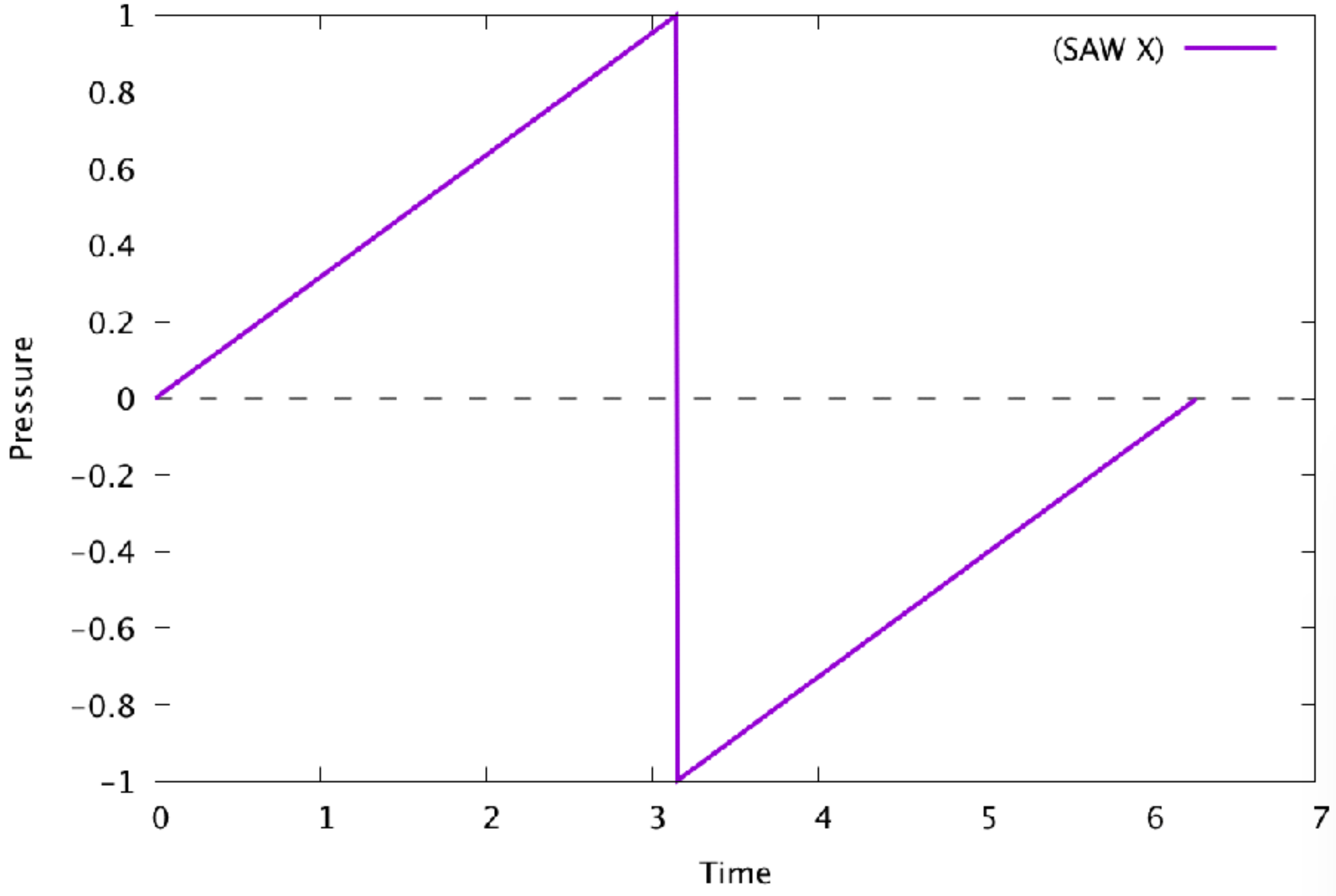


$$p = \begin{cases} 1 \text{ if } t < \pi \\ -1 \text{ if } t \geq \pi \end{cases}$$

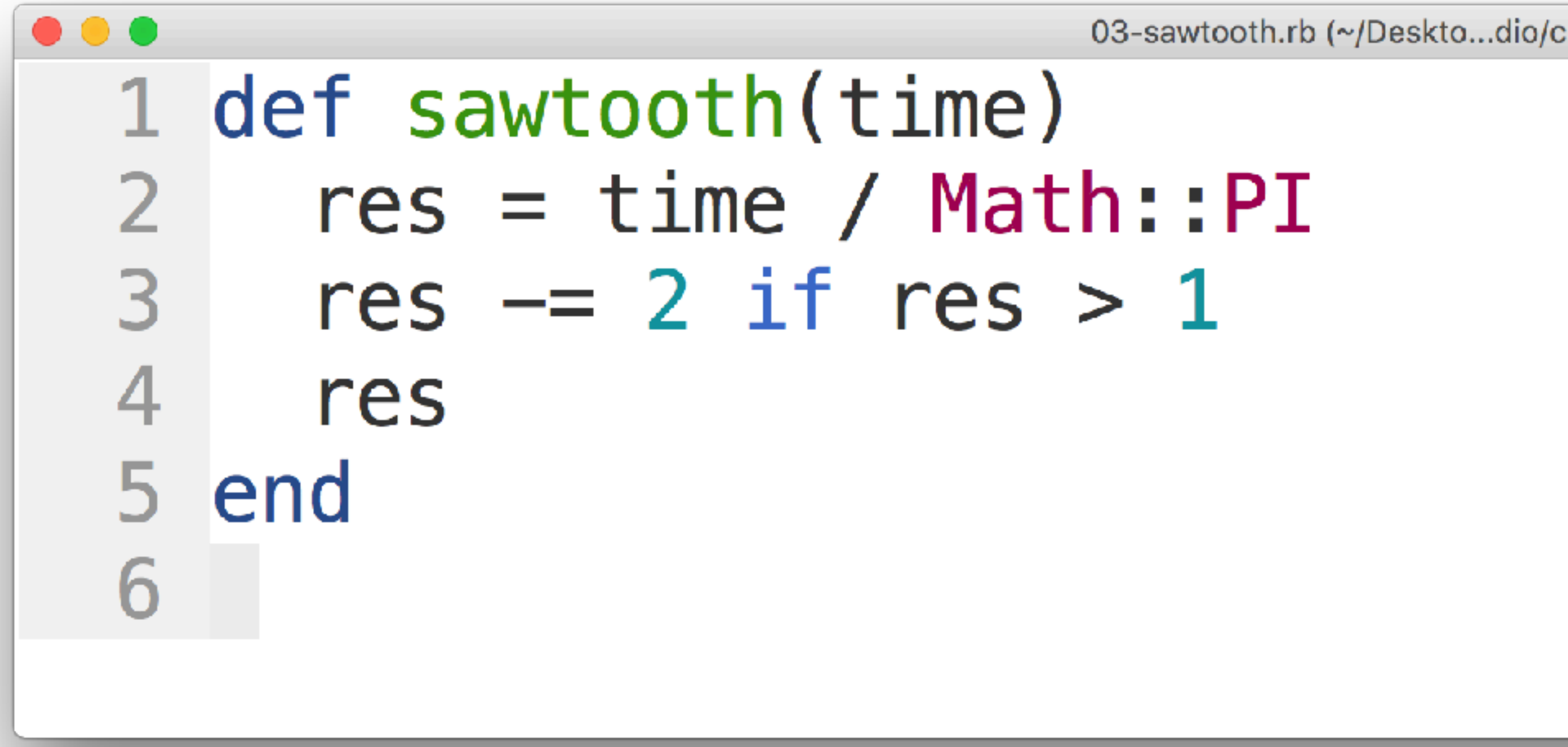


```
def square(time)
  time < Math::PI ? 1.0 : -1.0
end

```

Sawtooth Wave

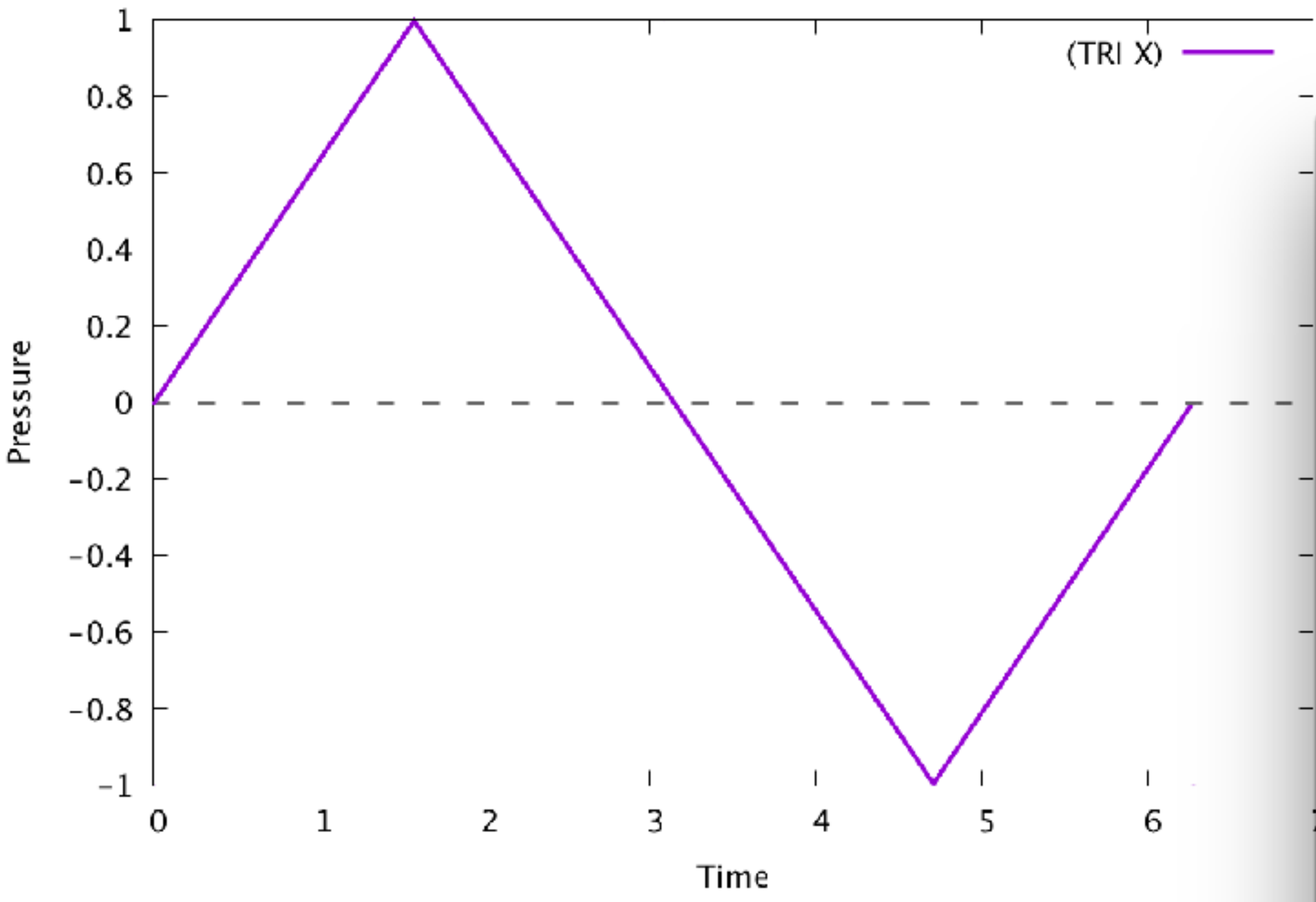


$$p = \begin{cases} \dfrac{t}{\pi} & \text{if } t < \pi \\ \dfrac{t}{\pi} - 2 & \text{if } t \geq \pi \end{cases}$$

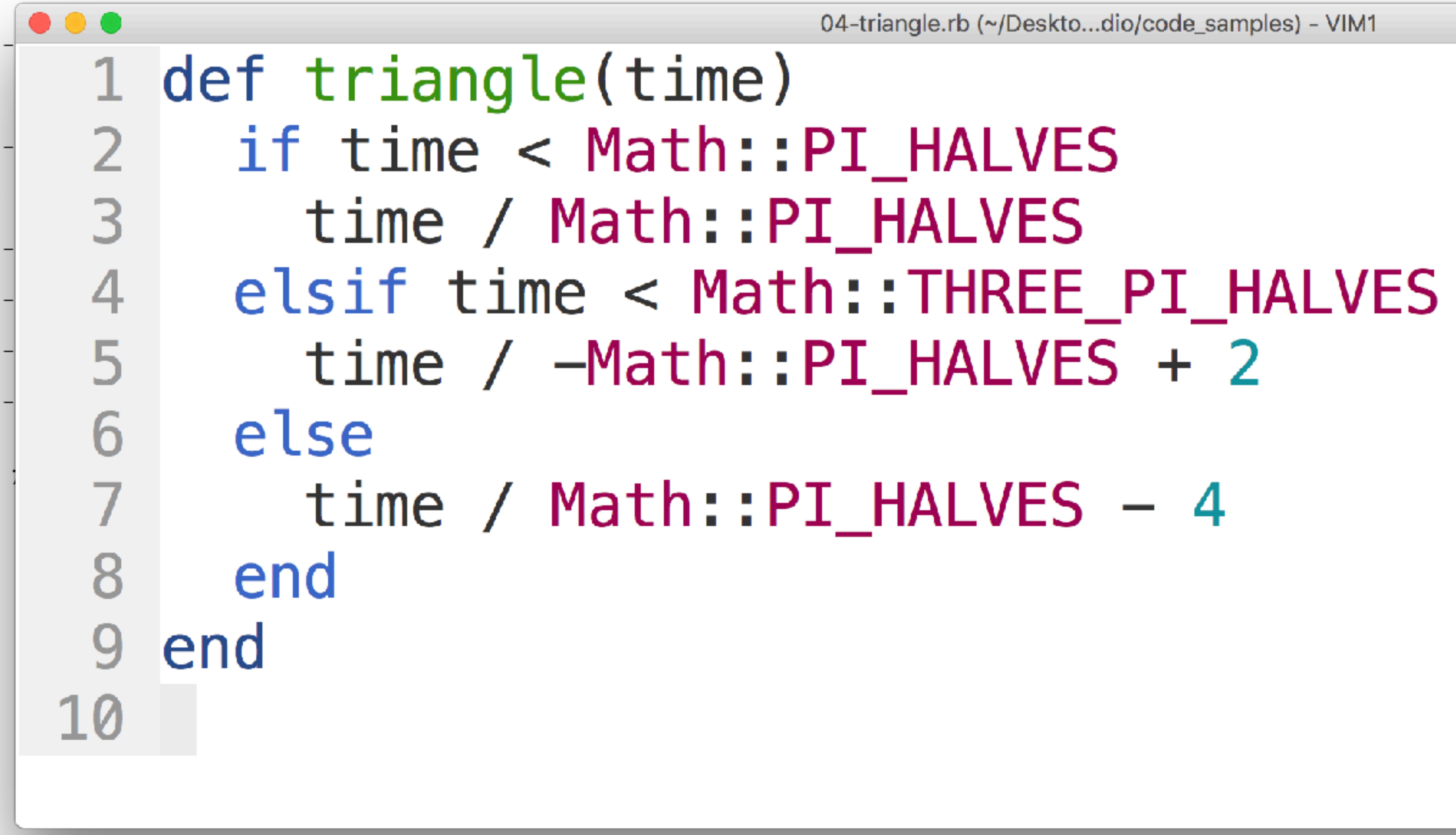


```ruby
def sawtooth(time)
  res = time / Math::PI
  res -= 2 if res > 1
  res
end

```

Triangle Wave

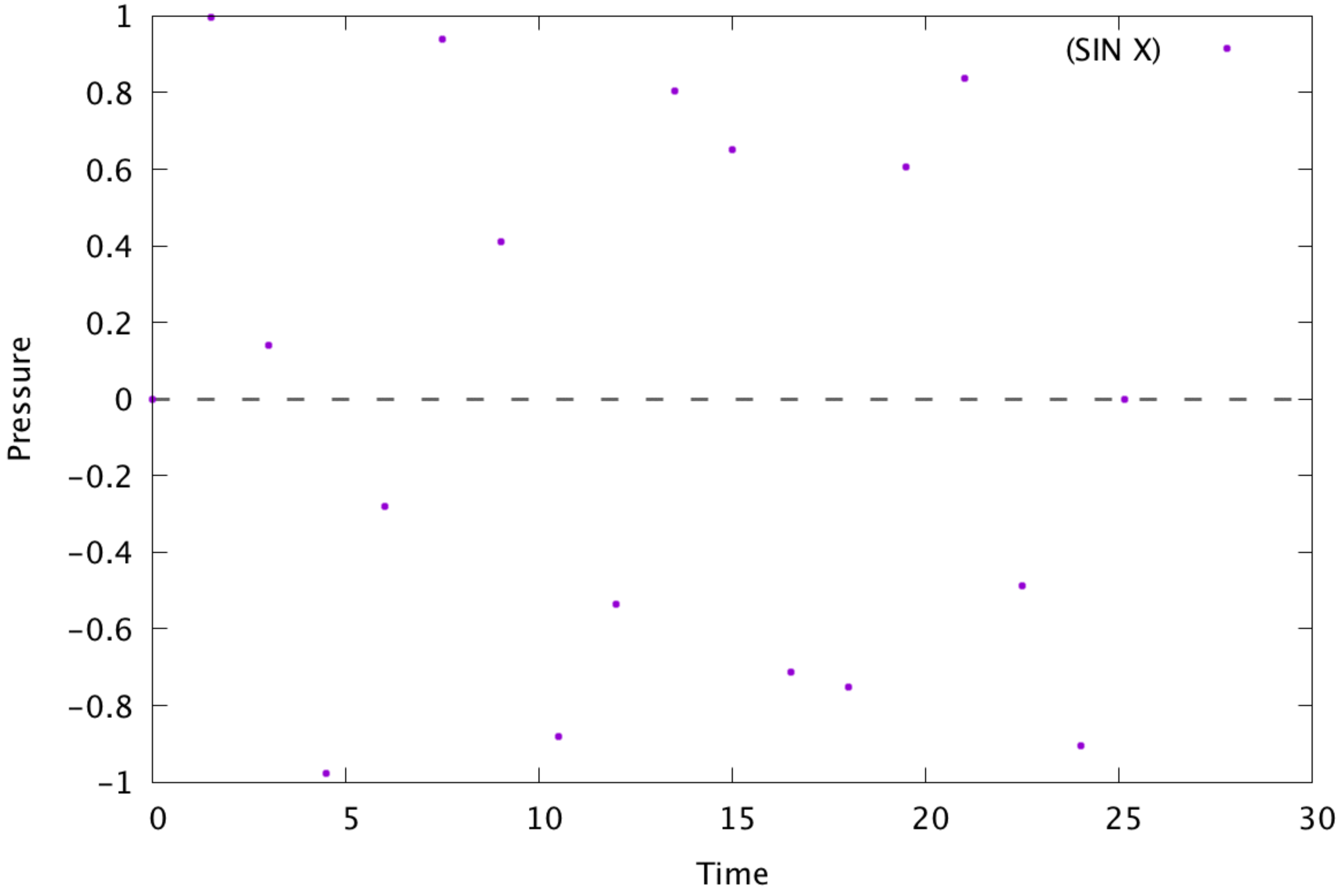


$$p = \begin{cases} \frac{2t}{\pi} & \text{if } t < \frac{\pi}{2} \\ -\frac{2t}{\pi} + 2 & \text{if } t < \frac{3\pi}{2} \\ \frac{2t}{\pi} - 4 & \text{if } t \geq \frac{3\pi}{2} \end{cases}$$

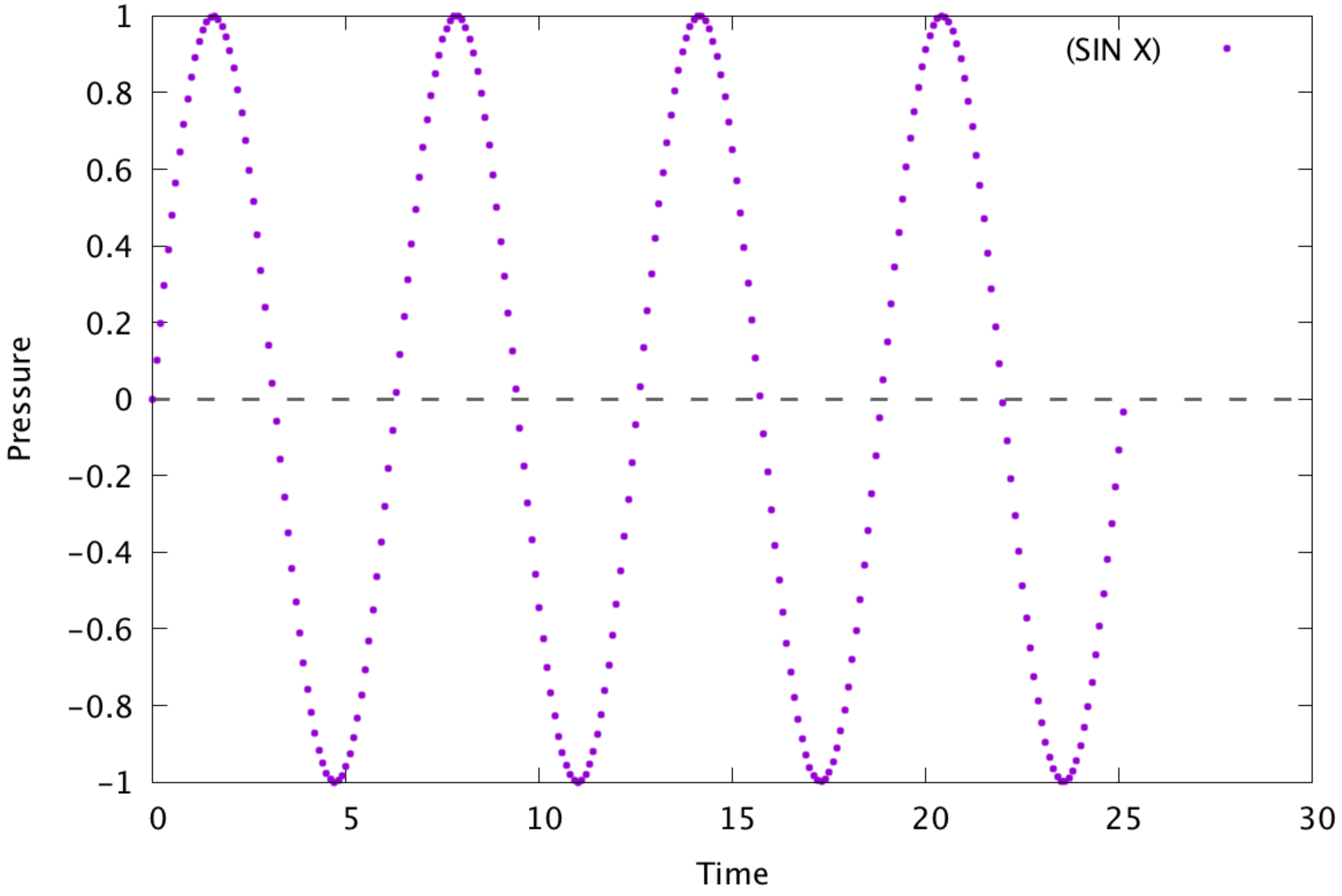


```ruby
def triangle(time)
  if time < Math::PI_HALVES
    time / Math::PI_HALVES
  elsif time < Math::THREE_PI_HALVES
    time / -Math::PI_HALVES + 2
  else
    time / Math::PI_HALVES - 4
  end
end

```

Sampling



1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
Pressure
0
5
10
15
20
25
30
Time
(SIN X) ●



(SIN X)
Pressure
Time
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
0
5
10
15
20
25
30

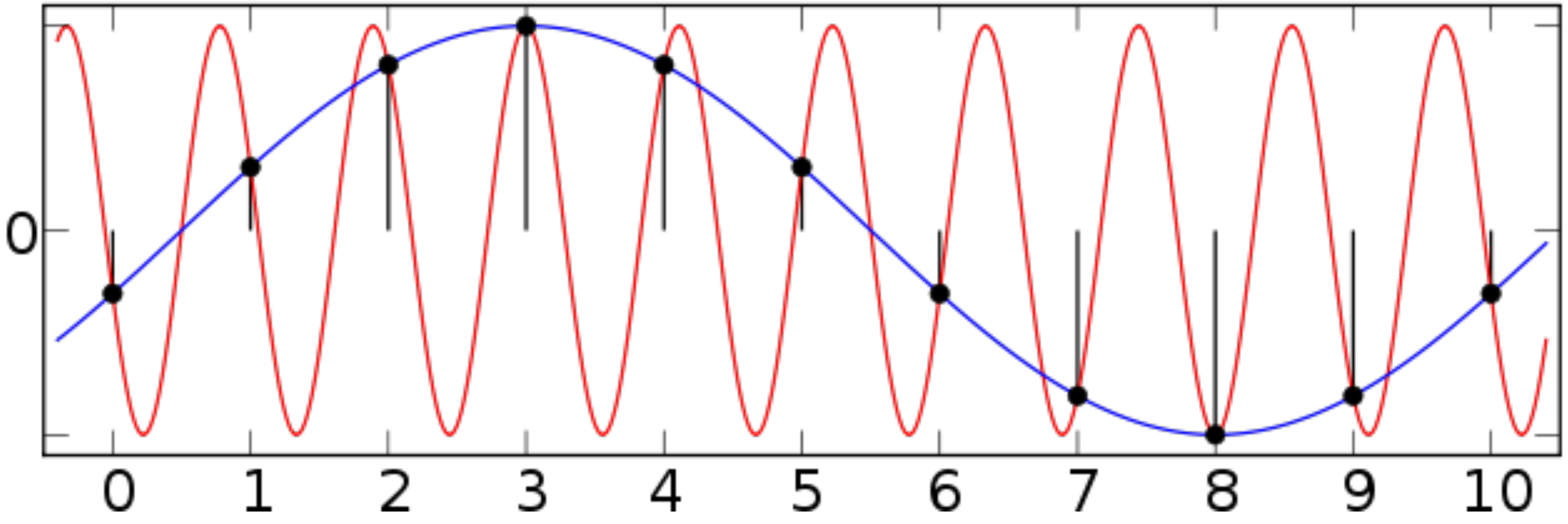
44100 Hz

Nyquist–Shannon

If a function $x(t)$ contains no frequencies higher than B hertz, then it can be completely determined with a sampling rate of $2B$ samples / second.

Nyquist–Shannon

If a function x(t) contains no frequencies higher than B hertz, then it can be completely determined with a sampling rate of 2B samples / second.



Human hearing:
~20Hz to ~20,000Hz

Sample rate: 44100 Hz

Buffering…

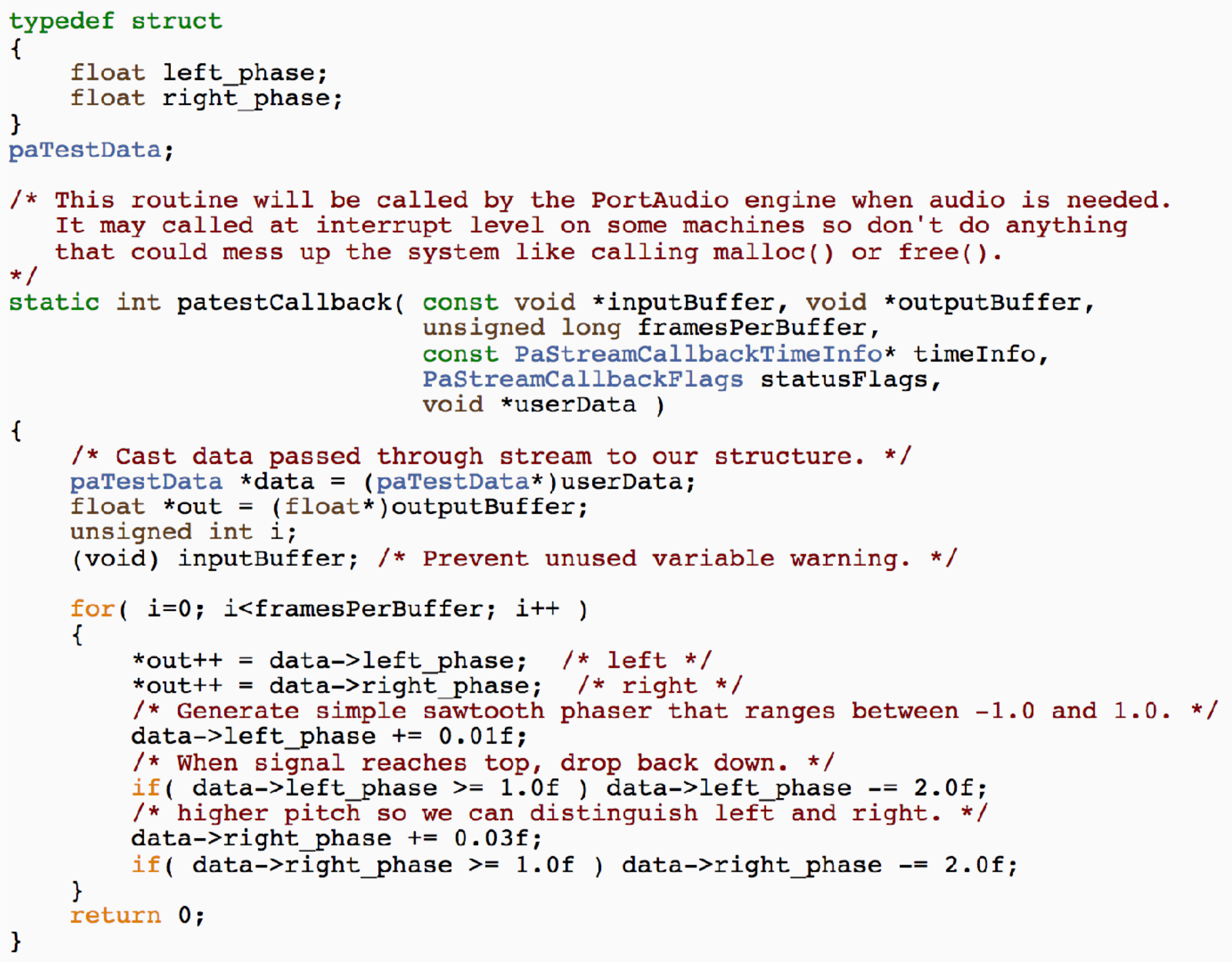
Buffer size: 512

PortAudio

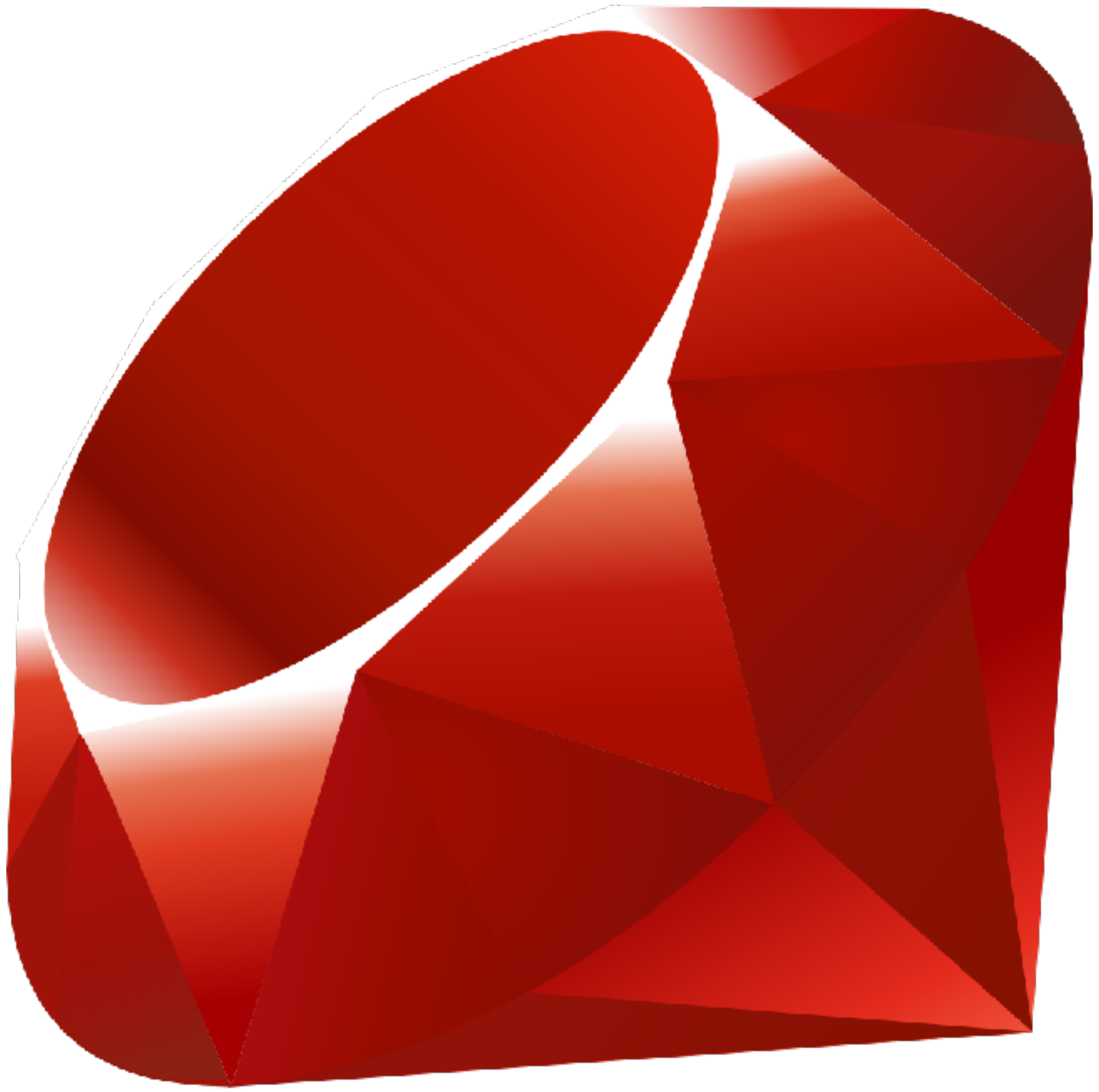
Portable Cross-platform Audio I/O

PortAudio is a free, cross-platform, open-source, audio I/O library.

It lets you write simple audio programs in C or C++ that will compile and run on many platforms.

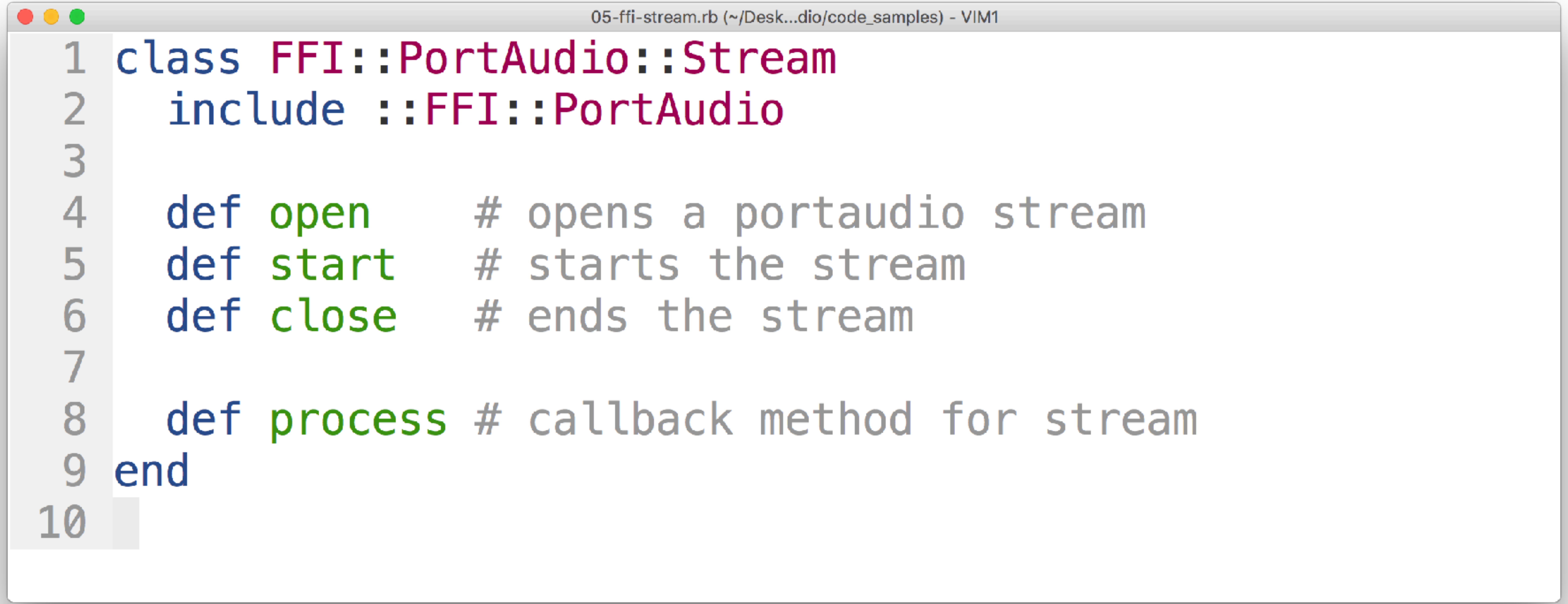


```
typedef struct
{
    float left_phase;
    float right_phase;
}
paTestData;

/* This routine will be called by the PortAudio engine when audio is needed.
   It may called at interrupt level on some machines so don't do anything
   that could mess up the system like calling malloc() or free().
*/
static int patestCallback( const void *inputBuffer, void *outputBuffer,
                           unsigned long framesPerBuffer,
                           const PaStreamCallbackTimeInfo* timeInfo,
                           PaStreamCallbackFlags statusFlags,
                           void *userData )
{
    /* Cast data passed through stream to our structure. */
    paTestData *data = (paTestData*)userData;
    float *out = (float*)outputBuffer;
    unsigned int i;
    (void) inputBuffer; /* Prevent unused variable warning. */

    for( i=0; i<framesPerBuffer; i++ )
    {
        *out++ = data->left_phase;  /* left */
        *out++ = data->right_phase;  /* right */
        /* Generate simple sawtooth phaser that ranges between -1.0 and 1.0. */
        data->left_phase += 0.01f;
        /* When signal reaches top, drop back down. */
        if( data->left_phase >= 1.0f ) data->left_phase -= 2.0f;
        /* higher pitch so we can distinguish left and right. */
        data->right_phase += 0.03f;
        if( data->right_phase >= 1.0f ) data->right_phase -= 2.0f;
    }
    return 0;
}
```

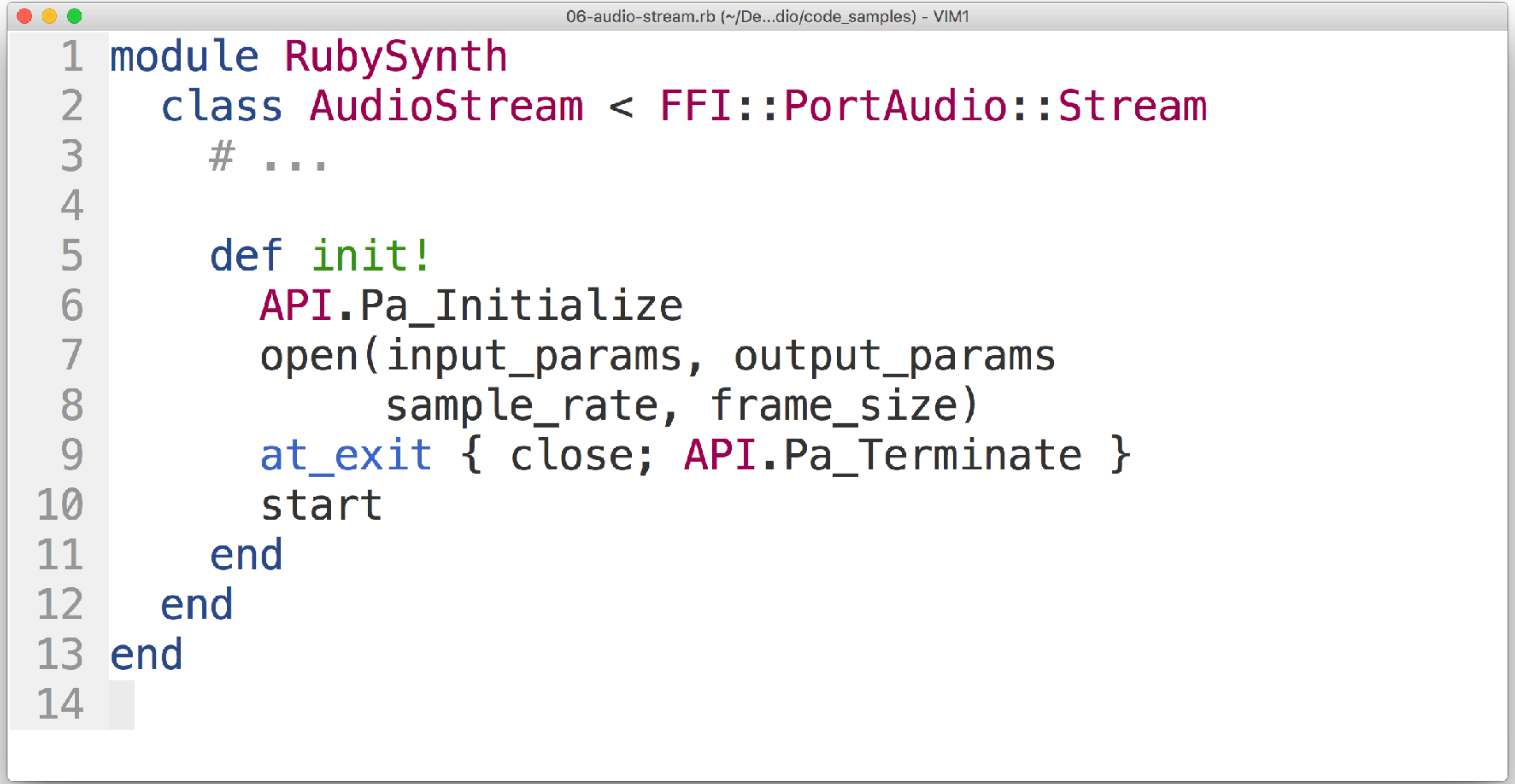


Fast Enough™

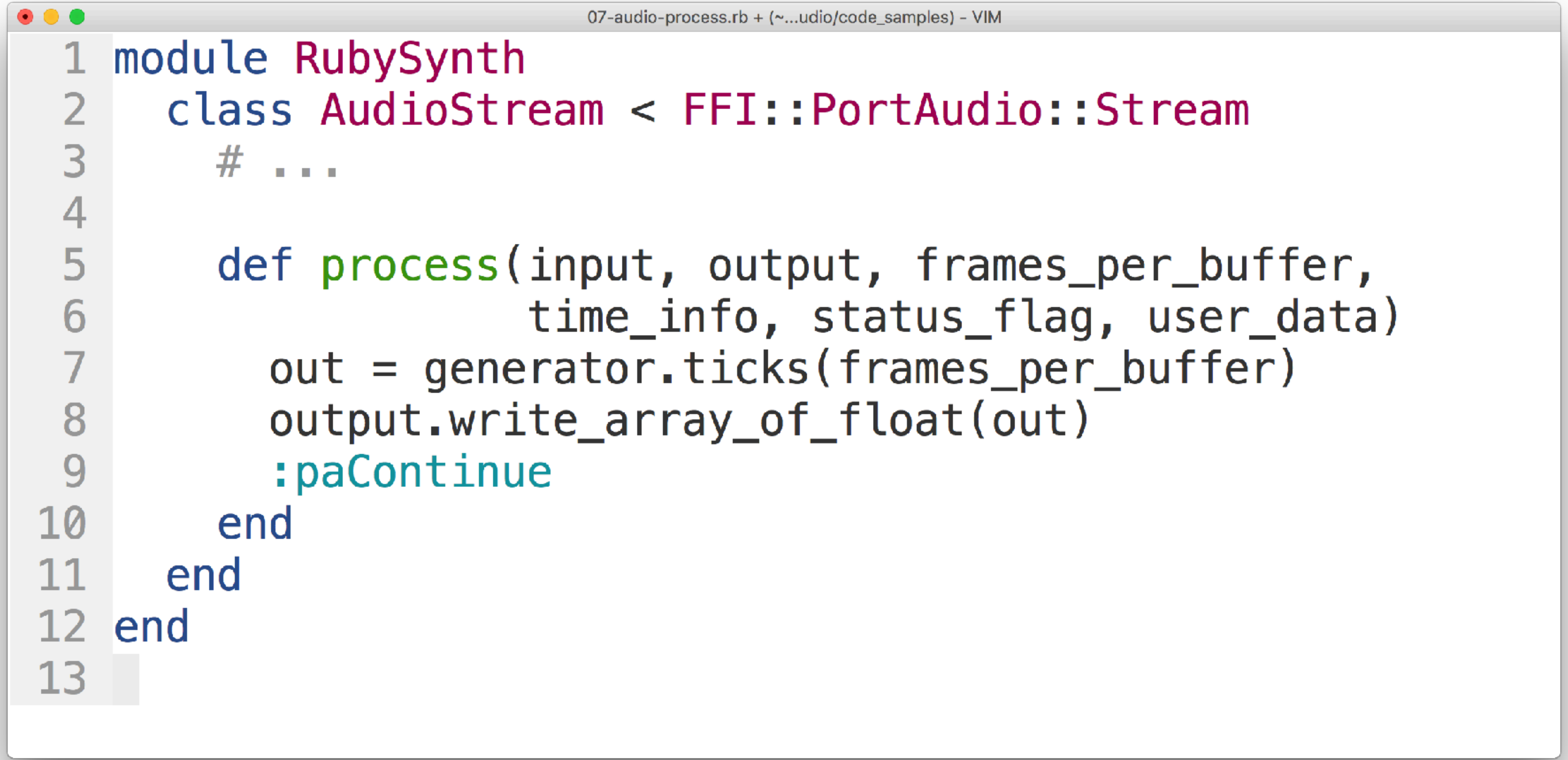
https://github.com/nanki/ffi-portaudio



```ruby
class FFI::PortAudio::Stream
  include ::FFI::PortAudio

  def open    # opens a portaudio stream
  def start   # starts the stream
  def close   # ends the stream

  def process # callback method for stream
end
```

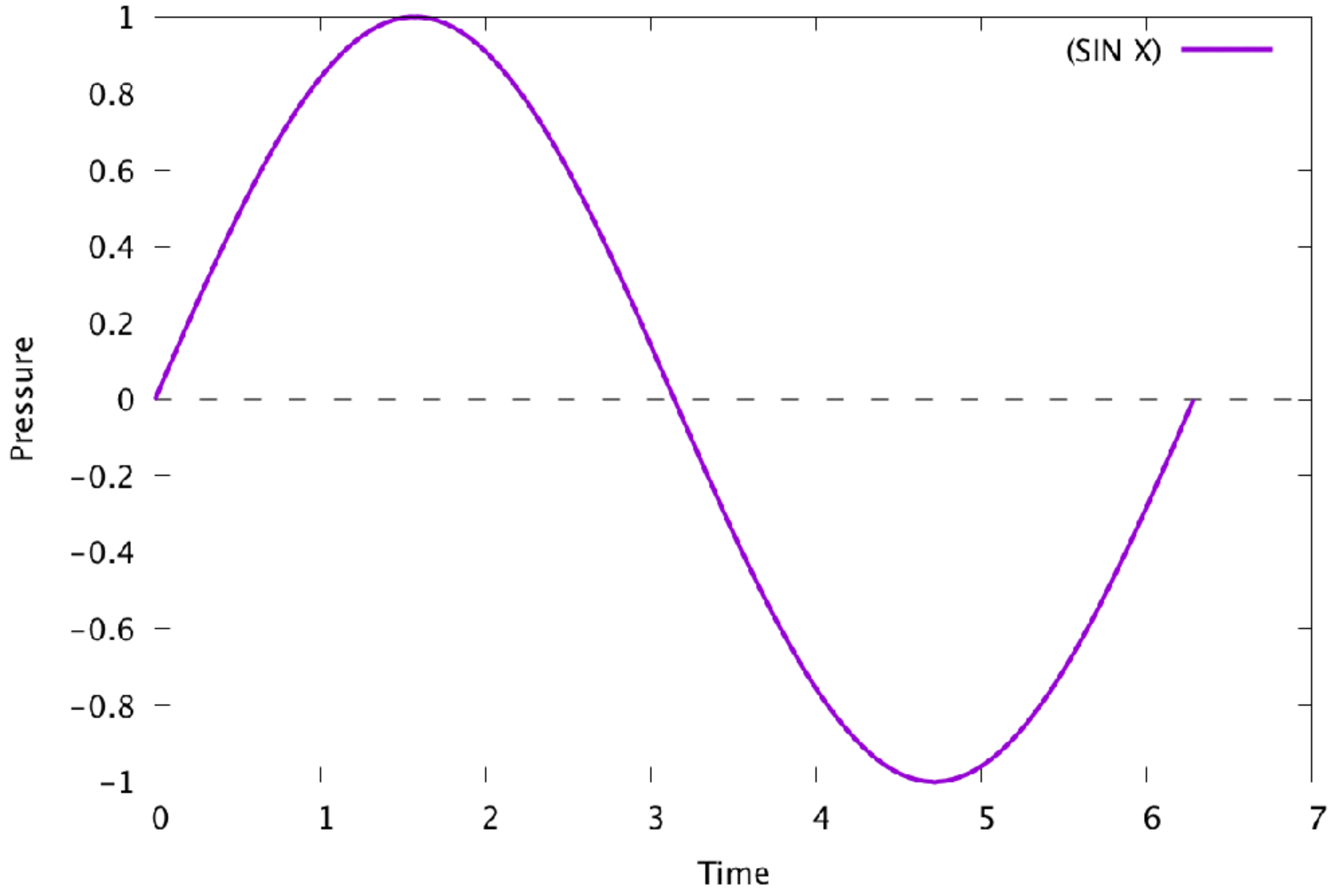


```ruby
module RubySynth
  class AudioStream < FFI::PortAudio::Stream
    # ...

    def init!
      API.Pa_Initialize
      open(input_params, output_params
           sample_rate, frame_size)
      at_exit { close; API.Pa_Terminate }
      start
    end
  end
end

```

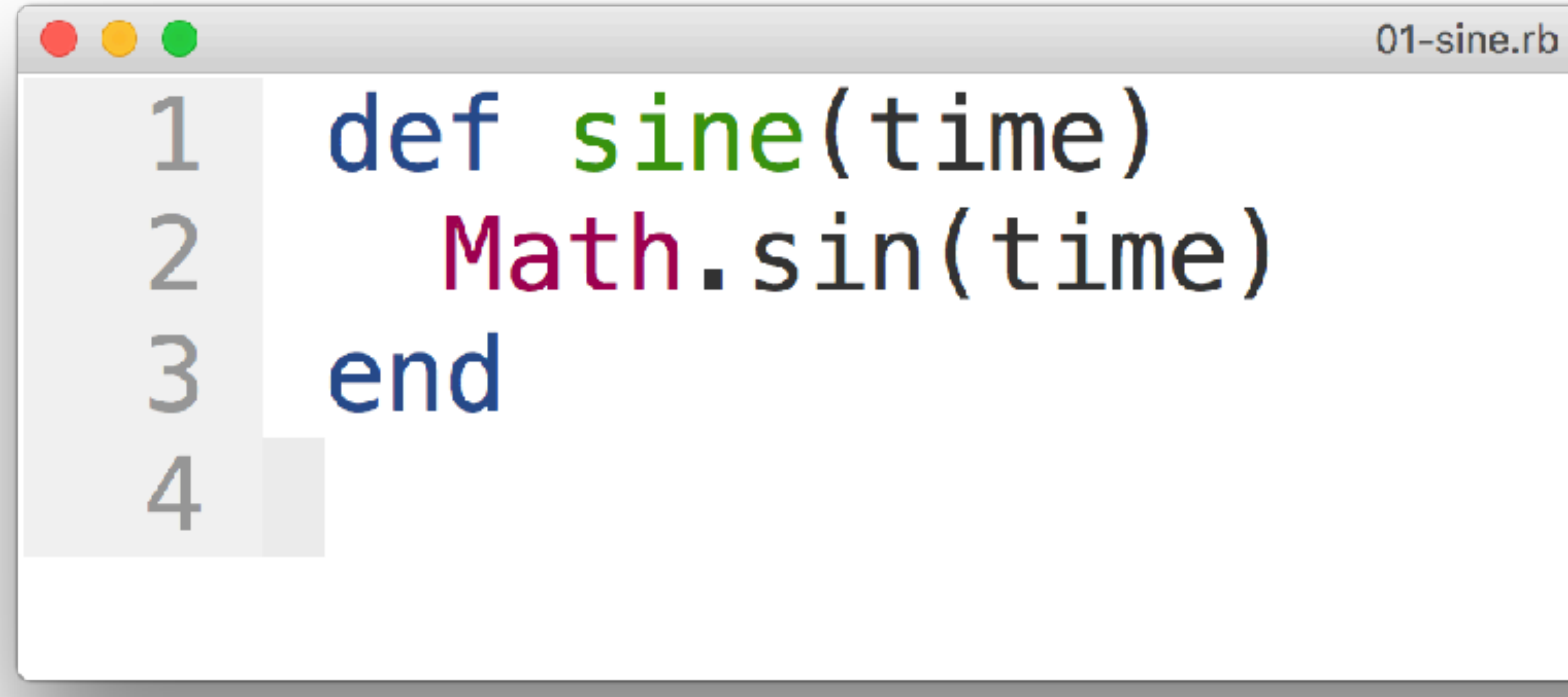


```ruby
module RubySynth
  class AudioStream < FFI::PortAudio::Stream
    # ...

    def process(input, output, frames_per_buffer,
                time_info, status_flag, user_data)
      out = generator.ticks(frames_per_buffer)
      output.write_array_of_float(out)
      :paContinue
    end
  end
end

```

Sine Wave



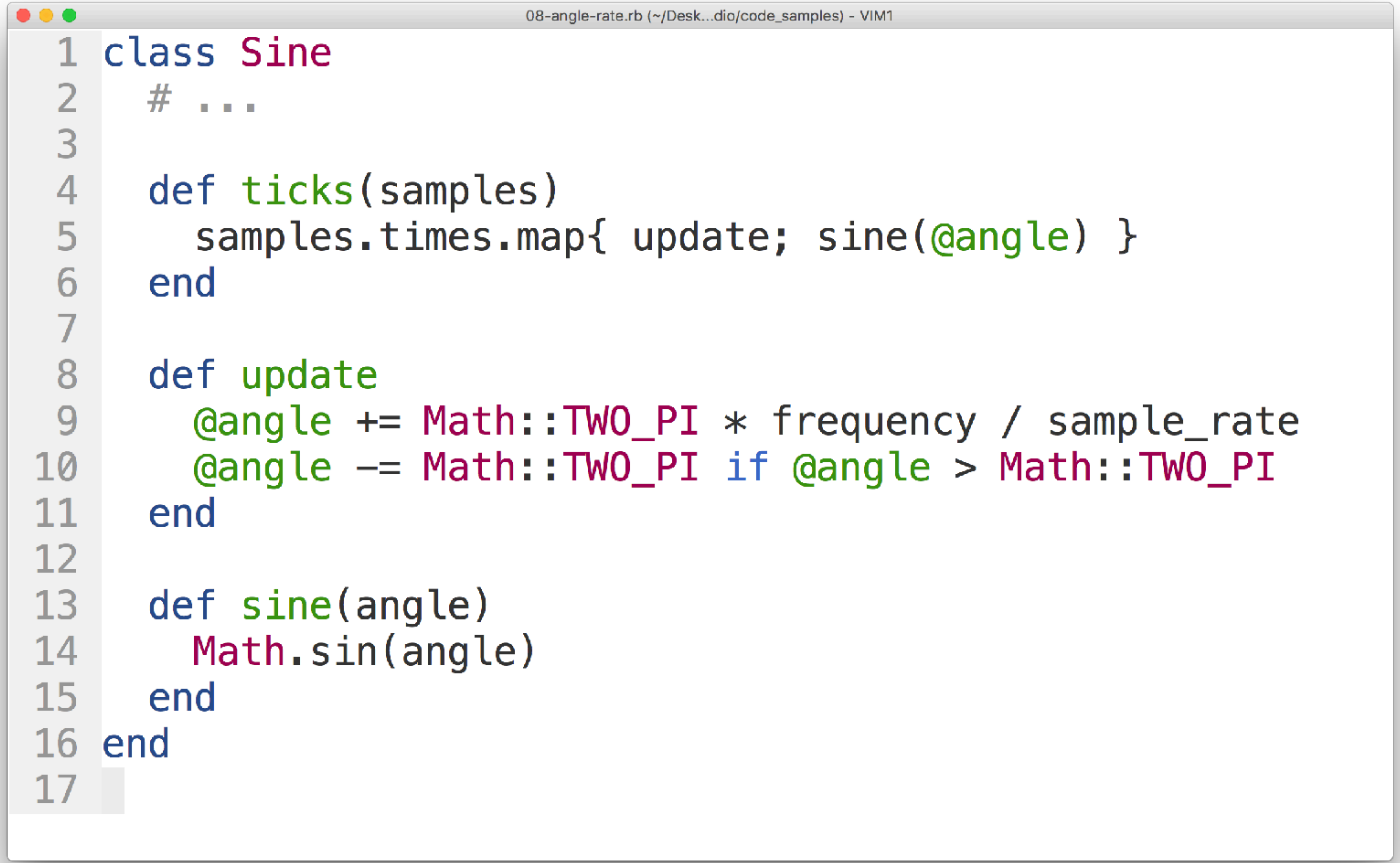
$$p = sin(t)$$



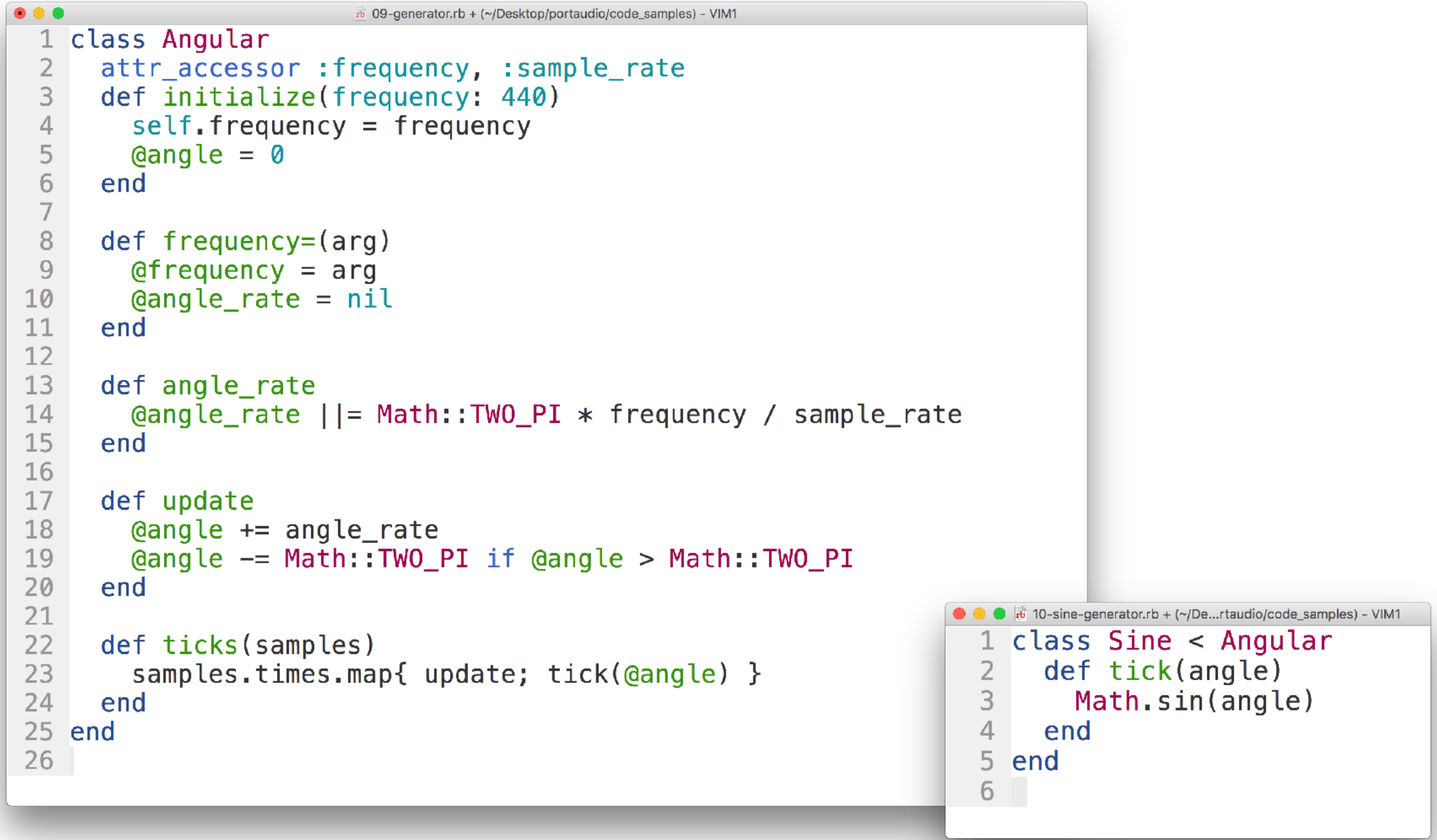
```ruby
def sine(time)
  Math.sin(time)
end

```

	C	C#	D	Eb	E	F	F#	G	G#	A	Bb	B
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

$$\text{angle}_i = \frac{2\pi * \text{frequency}}{\text{sample rate}} * i$$



```ruby
class Sine
  # ...

  def ticks(samples)
    samples.times.map{ update; sine(@angle) }
  end

  def update
    @angle += Math::TWO_PI * frequency / sample_rate
    @angle -= Math::TWO_PI if @angle > Math::TWO_PI
  end

  def sine(angle)
    Math.sin(angle)
  end
end
```



```ruby
class Angular
  attr_accessor :frequency, :sample_rate
  def initialize(frequency: 440)
    self.frequency = frequency
    @angle = 0
  end

  def frequency=(arg)
    @frequency = arg
    @angle_rate = nil
  end

  def angle_rate
    @angle_rate ||= Math::TWO_PI * frequency / sample_rate
  end

  def update
    @angle += angle_rate
    @angle -= Math::TWO_PI if @angle > Math::TWO_PI
  end

  def ticks(samples)
    samples.times.map{ update; tick(@angle) }
  end
end

```

```ruby
class Sine < Angular
  def tick(angle)
    Math.sin(angle)
  end
end

```

Demos!

- Basic waves
- Change frequency
- Keyboard
- Chords

Credits

- Thanks to Steve Losh for the inspiration
  (and several of the graphs):
  http://stevelosh.com/blog/2016/12/chip8-sound

- Thanks to http://www.portaudio.com/

- Thanks to NANKI Haruo for ffi-portaudio:
  https://github.com/nanki