Stochastic Music Experiment - Read Me

Wild Sound Explorers 2022 Summer

yiman@stanford.edu

In [ ]:

```
# import python modules
import numpy as np
π = np.pi
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (20, 5)  # make the graphs wider
from numpy.random import default_rng
rng = default_rng()
import sounddevice as sd
sd.default.channels = 1
from scipy.fft import rfft, irfft, rfftfreq
from scipy.io.wavfile import write
```

In [ ]:

```
def plot_fft(x):
    Ts = 1/fs
    f_fft = rfftfreq(x.size, Ts)
    x_fft = rfft(x)
    plt.plot(f_fft[200:500], np.abs(x_fft)[200:500])
    plt.xlabel("frequency (Hz)")
    plt.ylabel("amplitude");

def bandpass(spectrum, fs, fmin, fmax):
    filtered = np.zeros(spectrum.size, dtype=complex)
    N = 2 * spectrum.size
    imin = int(fmin * N / fs)
    imax = int(fmax * N / fs)
    filtered[imin:imax] = spectrum[imin:imax]
    return filtered
```

In [ ]:

```
# defining all the signals we are going to use as building blocks for music
# generic parameters
tmax = 0.5                      # end of signal in "real" time (seconds)
fs = 48000                     # sampling frequency
t = np.arange(0, tmax, 1/fs)   # time vector t[n]

chrom_scale = [261.63, 277.18, 293.66, 311.13, 329.63, 349.23, 369.99, 392.00, 415.3
chrom_names = ['C4', 'C#4', 'D4', 'D#4', 'E4', 'F4', 'F#4', 'G4', 'Ab4', 'A4', 'Bb4'
chrom = {}
for i in range(len(chrom_names)):
    name = chrom_names[i]
    f = chrom_scale[i]
    chrom[name] = np.sin(2*π*f*t) # signal vector x[n]
```

In [ ]:

```
chrom
```

Out[ ]:

```
{'C4': array([ 0.        ,  0.03424059,  0.06844103, ..., -0.95364699,
        -0.94278385, -0.93081505]),
 'C#4': array([ 0.        ,  0.03627482,  0.07250189, ..., -0.44093339,
        -0.47320128, -0.5048463 ]),
 'D4': array([ 0.        ,  0.03843054,  0.0768043 , ..., -0.92591905,
        -0.91071899, -0.89417338]),
 'D#4': array([ 0.        ,  0.04071556,  0.0813636 , ..., -0.28233434,
        -0.32115932, -0.35945167]),
 'E4': array([ 0.        ,  0.04313508,  0.08618986, ..., -0.96134187,
        -0.94856953, -0.93403143]),
 'F4': array([ 0.        ,  0.04569818,  0.09130088, ..., -0.55255283,
        -0.590064  , -0.62634228]),
 'F#4': array([ 0.        ,  0.04841265,  0.09671176, ..., -0.17579241,
        -0.12792755, -0.07976269]),
 'G4': array([ 0.        ,  0.05129017,  0.10244531, ..., -0.15333078,
        -0.10244531, -0.05129017]),
 'Ab4': array([ 0.        ,  0.05433587,  0.1085112 , ..., -0.70284554,
        -0.74045865, -0.77588402]),
 'A4': array([ 0.        ,  0.05756403,  0.11493715, ..., -0.1719291 ,
        -0.11493715, -0.05756403]),
 'Bb4': array([0.        , 0.06098234, 0.12173769, ..., 0.31418135, 0.37149099,
        0.42741782]),
 'B4': array([ 0.        ,  0.06460372,  0.12893752, ..., -0.54042096,
        -0.48493479, -0.42742256]),
 'silence': array([0., 0., 0., ..., 0., 0., 0.])}
```

In [ ]:

```
def rand_message(num):
    message = []
    while len(message) < num:
        r = rng.integers(0, 13)
        if r not in message:
            message.append(r)
    return message
```

In [ ]:

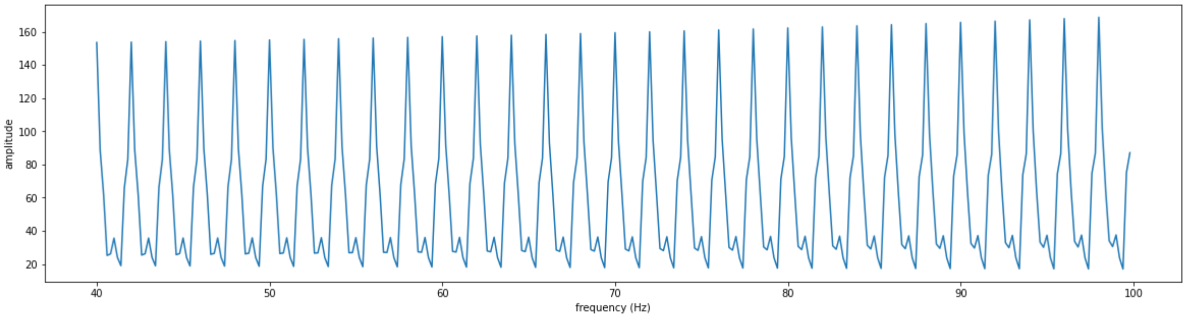
```
def rand_notes(num):
    message = rand_message(num)
    notes = np.array([])
    for n in message:
        notes = np.concatenate((notes, chrom[chrom_names[n]]), axis=None)
    return notes
```

In [ ]:

```
sine_notes1 = rand_notes(10)
```

In [ ]:

```
plot_fft(sine_notes1)
sd.play(sine_notes1)
```



In [ ]:

```
# make the note seqence longer
def generate_sequence(num):
    sine_notes = np.array([])
    for i in range(num):
        new_notes = rand_notes(10)
        sine_notes = np.concatenate((sine_notes, new_notes))
    return sine_notes
```

In [ ]:

```
# next step is to add sounds in nature; import all recorded samples
# audio samples processed on Audacity
import soundfile as sf
sample1, any = sf.read('bird_sample_0.5sec.wav')
sample2, any = sf.read('bird_sample_1sec.wav')
sample3, any = sf.read('bird_sample_1.5sec.wav')
sample4, any = sf.read('bird_sample_2sec.wav')
sample5, any = sf.read('whistle_sample_10sec.wav')
sample1 = np.transpose(sample1).flatten()
sample2 = np.transpose(sample2).flatten()
sample3 = np.transpose(sample3).flatten()
sample4 = np.transpose(sample4).flatten()
sample5 = np.transpose(sample5).flatten()
```

In [ ]:

```
# first generate a beat (track1) using the shortest sample
# generate a list of random 0s and 1s
message = rng.integers(0, 2, 30)
track1 = np.array([])
for i in message:
    if i==1:
        track1 = np.concatenate((track1, sample1))
    else:
        track1 = np.concatenate((track1, np.zeros(int(0.5*fs))))
rec_track1 = sd.playrec(track1, fs, blocking=True)
```

In [ ]:

```
# do the same thing for tracks 2 through 4
message = rng.integers(0, 2, 15)
track2 = np.array([])
for i in message:
    if i==1:
        track2 = np.concatenate((track2, sample2))
    else:
        track2 = np.concatenate((track2, np.zeros(int(1*fs))))
rec_track2 = sd.playrec(track2, fs, blocking=True)
```

In [ ]:

```
message = rng.integers(0, 2, 10)
track3 = np.array([])
for i in message:
    if i==1:
        track3 = np.concatenate((track3, sample3))
    else:
        track3 = np.concatenate((track3, np.zeros(int(1.5*fs))))
rec_track3 = sd.playrec(track3, fs, blocking=True)
```

In [ ]:

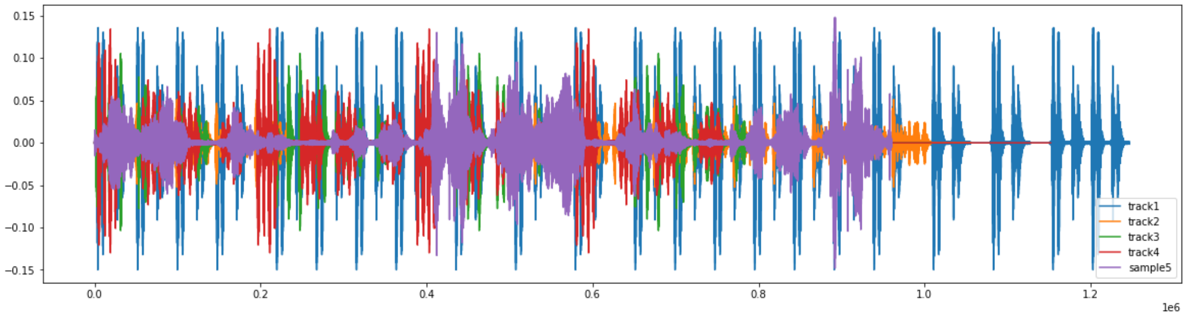
```
message = rng.integers(0, 2, 8)
track4 = np.array([])
for i in message:
    if i==1:
        track4 = np.concatenate((track4, sample4))
    else:
        track4 = np.concatenate((track4, np.zeros(int(2*fs))))
rec_track4 = sd.playrec(track4, fs, blocking=True)
```

In [ ]:

```
plt.plot(track1, label='track1')
plt.plot(track2, label='track2')
plt.plot(track3, label='track3')
plt.plot(track4, label='track4')
plt.plot(sample5, label='sample5')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x27e00f8ee80>
```

```
c:\Users\Yiman\AppData\Local\Programs\Python\Python39\lib\site-packages\IPython\core
\pylabtools.py:151: UserWarning: Creating legend with loc="best" can be slow with la
rge amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



In [ ]:
```
def padding(track):
    pad_length = int(1246769-len(track))
    track = np.concatenate((track.flatten(), np.zeros(pad_length)))
    return track
```

In [ ]:
```
sine_notes = generate_sequence(3)
```

In [ ]:
```
track_names = [rec_track1*0.3, rec_track2*1.2, rec_track3*0.5, rec_track4*0.3, samp
padded_tracks = []
for track in track_names:
    padded_tracks.append(padding(track))

final_prod = padded_tracks[0] + padded_tracks[1] + padded_tracks[2] + padded_tracks[
final_prod = final_prod * 1000
write("example.wav", fs, final_prod.astype(np.int16))
```

In [ ]:
```
sd.play(final_prod)
```