ENGR 76 Project 2d: Hiding Messages in Music !!

Due: May 27, 2022 at 5:00 PM

cgarcia0@stanford.edu

Part 1 Setup

This is where I import files and code.

In [3]:
```
import numpy as np

π = np.pi
import matplotlib.pyplot as plt
import sounddevice as sd
sd.default.channels = 1

from scipy.fft import rfft, irfft, rfftfreq
from viterbi import viterbi_decode

def conv_encode(message):
    """Encodes the `message` using the convolutional code discussed in lecture.
    `message` should be a 1-D `np.ndarray` of integers, all 1 or 0.
    Returns a 1-D `np.ndarray` of integers, all 1 or 0.
    """
    encoded = []
    message = np.insert(message, 0, [0,0])
    message = np.append(message, [0,0])

    for i in range(len(message) - 2):
        encoded.append(message[i] ^ message[i+1] ^ message[i+2])
        encoded.append(message[i] ^ message[i+2])

    return np.array(encoded,dtype=np.int8)
```

The Wav Files

> I looked at two different files. The first was a wav formatted version of Toto's Africa that I found on a website made by The University of Colombia EE department. It has a sampling rate of 22.05kHz. I also chose a random piano sample from a website that has free sound effects called freesound.com. I wanted this additional sample for comparison because it has a sample rate of 44.1kHz like in all other projects.

In [4]:
```
import soundfile as sf
piano , piano_fs = sf.read('piano.wav') # reading from file
```

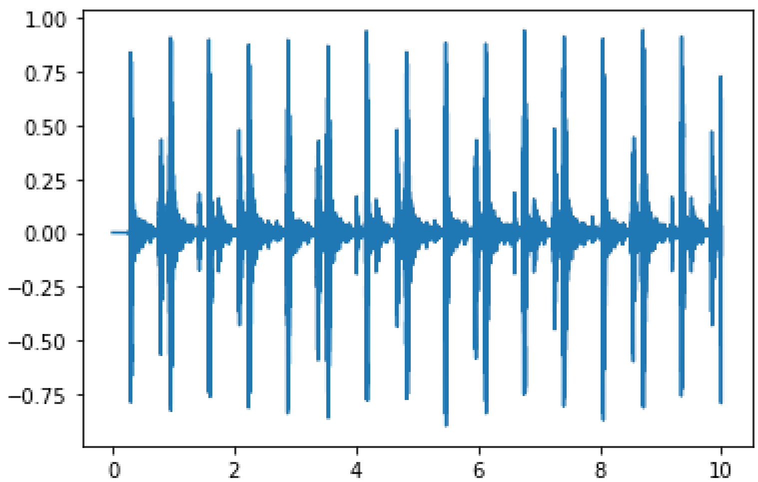
```
africa , africa_fs = sf.read('africa-toto.wav') # reading from file
```

In [5]:
```
print(f"Africa's sample rate: {africa_fs}")
print(f"Piano's sample rate: {piano_fs}")
```

```
Africa's sample rate: 22050
Piano's sample rate: 44100
```

In [6]:
```
# Plot of Africa
tmax = 10
t = np.arange(0, tmax, 1/africa_fs)
plt.plot(t, africa[:africa_fs*tmax])
```

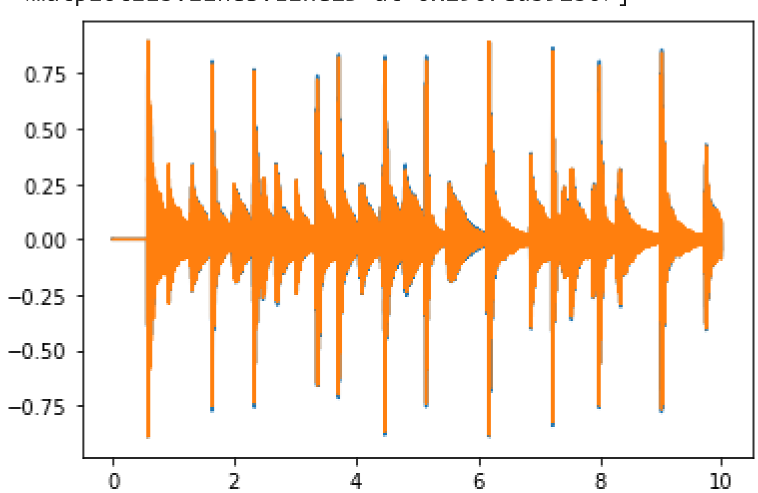
Out[6]:
```
[<matplotlib.lines.Line2D at 0x1907cc57ac0>]
```



In [7]:
```
# What 5 seconds sound like a few seconds in
sd.play(africa[500000:500000+africa_fs*5], africa_fs)
```

In [8]:
```
# Plot of Africa
tmax = 10
t = np.arange(0, tmax, 1/piano_fs)
plt.plot(t, piano[0:piano_fs*tmax])
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x1907edb9250>,
 <matplotlib.lines.Line2D at 0x1907edb92b0>]
```



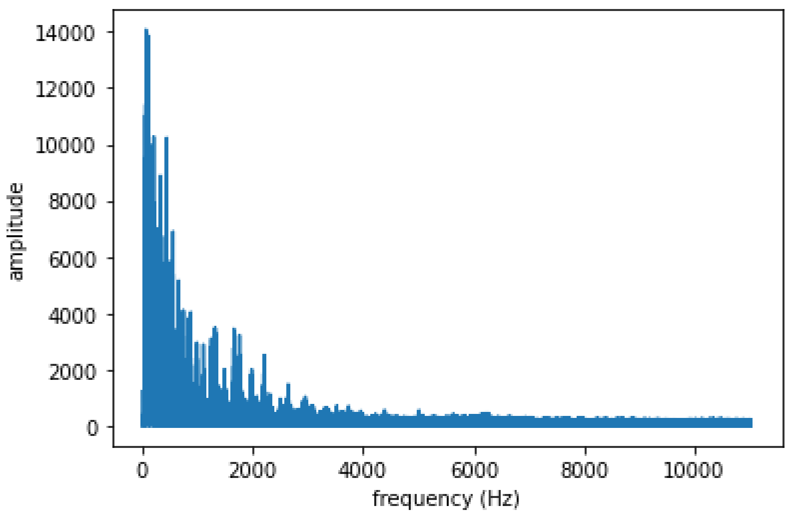
In [9]:
```
# What 5 seconds from the start sounds like
```

```
sd.play(piano[:piano_fs*5], piano_fs)
```

Part 2 Removing a Frequency Band From the Sample

Here I got the spectrum of the samples and removed a band of frequencies from them to leave room to insert data.

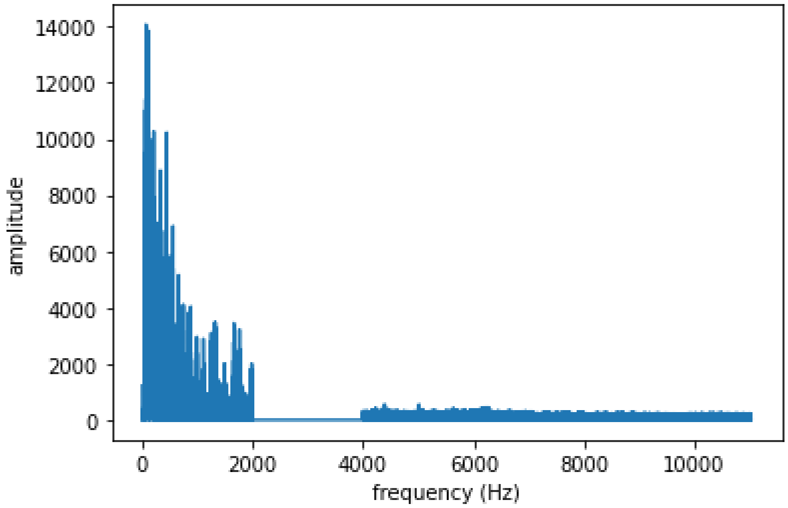
In [10]:
```
# The Spectrum of Toto's Africa
Ts = 1/africa_fs
f_fft = rfftfreq(africa.size, Ts)   # equivalent to: np.arange(x_fft.size) * fs / x.si
x_fft = rfft(africa)

plt.plot(f_fft, np.abs(x_fft))
plt.xlabel("frequency (Hz)")
plt.ylabel("amplitude");
```



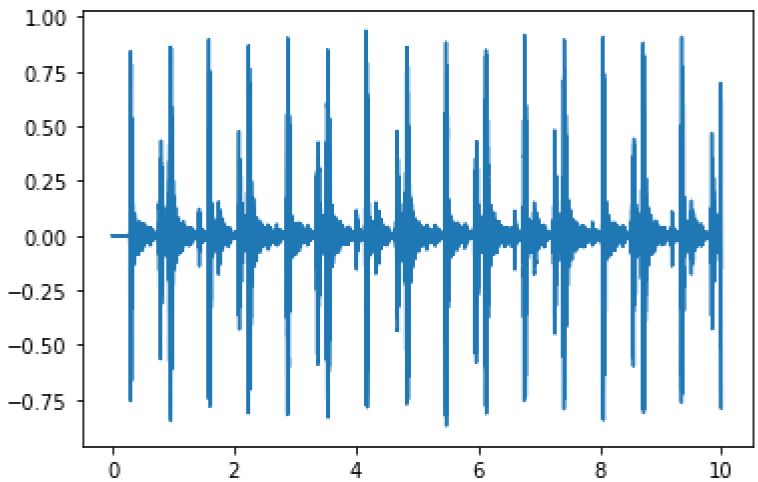
> First I get the sample's spectrum and then I pass it through a filter (from project 2b). I chose to remove the band 2kHz-4kHz and use the optimal 2.76kHz as the carrier frequency.

In [11]:
```
# The filtering code
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 [12]:
```
# Now I filter it
x_filtered = bandpass(x_fft, africa_fs, 0, 2000) +  bandpass(x_fft, africa_fs, 4000, 1
plt.plot(f_fft, np.abs(x_filtered))
plt.xlabel("frequency (Hz)")
plt.ylabel("amplitude");
```



In [13]:
```
# Now I convert it back to an audio wave
africa_filtered = irfft(x_filtered)
plt.plot(np.arange(0, tmax, 1/africa_fs), africa_filtered[:africa_fs*tmax])
plt.show()
```



In [153…
```
# This is what the filtered Africa sounds like
sd.play(africa_filtered[1500000:], africa_fs)
```

Part 3 Injecting Data

Now I want to inject a message into this waveform but I want to do something more interesting than injecting random bytes instead I am going to embead an ASCII message into the wave form by taking the code from part 2a and modifying it.

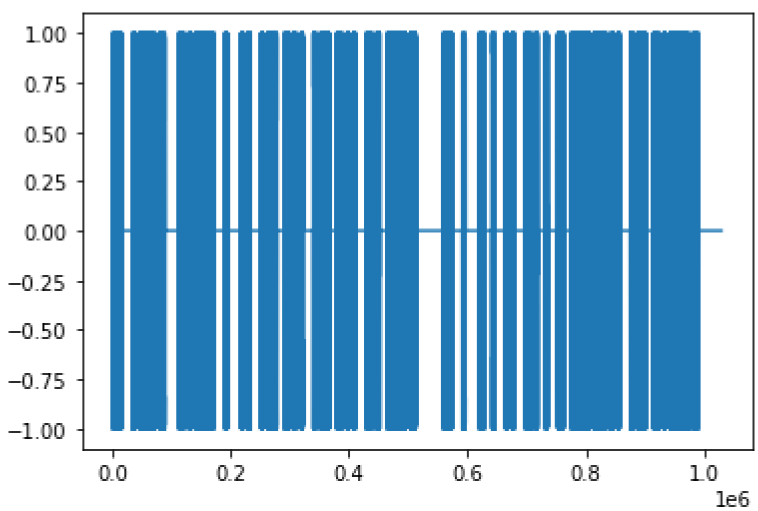
> Also I kept the data rate quite low at 4 bits/second for two reasons. I am going to place ASCII into the code and even one bit flip can crash the whole program because certain hex bytes in ASCII do not translate to a symbol but to some other instruction. I am also keeping it low to make it less noticable to the listener.

In [15]:
```
# This function takes in data, a rate, and a fc and returns a waveform that represents
```

```python
def waveform_gen(data, rate, fc, fs):
    #data_bytes = bytes(data, 'ASCII')
    message = np.array([])
    data_bytes = ''.join(format(ord(i), '08b') for i in data)
    message = np.append(message, int(1))
    for i in range(len(data_bytes)):
        message = np.append(message, int(data_bytes[i]))

    #message = np.insert(data_bytes, 0, 1)  # always start the message with a 1

    message = np.array(message,dtype=np.uint8)
    encoded = conv_encode(message)

    m_length = len(encoded)
    tmax = round(m_length/rate)   # end of signal in "real" time (seconds
    t = np.arange(0, tmax, 1/fs)   # time vector t[n]

    # generate `x` here, then plot it against time
    bandpass = []
    ratio = tmax*fs/m_length # this is the ratio to see which part of the sine waves c
    for i in range(tmax*fs):
        if i//(ratio) < m_length:
            bandpass.append(encoded[int(i/(ratio))])

    x = np.sin(2*π*fc*t) * bandpass

     # add 350ms of silence
    z = np.zeros(15435) # 350ms of nothing
    x = np.concatenate( (x, z), axis=None)
    return (x, encoded)
```

In [105…
```python
# This is what Hello looks like
x = waveform_gen("Hello World", rate=4, fc=2760, fs=africa_fs)
plt.plot(x[0])
```

Out[105]: [<matplotlib.lines.Line2D at 0x1909bb57cd0>]



Slight Detour: Decoding this Wave

> This is just a modified version of the final code from project 2c. I want to demonstrate how decoding this message back into text will look and demonstrate the version of the function I will be working with from now on. Also the bin_to_ascii function will take the viterbi decoded sequence that is received and turn it back from binary to text. There was also a multiplier parameter added that can change the amplitude of the received wave. This will be used to amplify the received wave that has the shruken down data later on.

In [203…

```
def bin_to_ascii(binary):
    length = len(binary)
    binary_letter = str()
    codeword = str()
    for i in range(length):
        binary_letter += str(binary[i])
        if ((i+1) % 8 == 0 and i != 0):
            binary_int = int(binary_letter, 2)
            letter = binary_int.to_bytes(1, 'big')
            letter_decoded = letter.decode()
            codeword += letter_decoded
            binary_letter = ""

    return str(codeword)


def decode(message, x, rate, fs, threshold=0.01, e_thresh=0.012, fmin=0, fmax=6000, mu
    m_length = len(message)
    tmax = round(m_length/rate)
    ratio = (tmax*fs/(m_length)) # this is the ratio to see which part of the sine wav

    y = sd.playrec(x, fs, blocking=True)
    y *= multiplier

    # New code for project2b
    y_fft = rfft(y.flatten())
    y_spec = bandpass(y_fft.flatten(), fs, fmin, fmax)
    y = irfft(y_spec)
    plt.plot(y)
    # decode

    start = np.nonzero(np.abs(y) > threshold)[0][0]

    cols = (m_length)
    symbols = []
    for i in range(m_length):
        symbols.append(list())

    curIndex = 0
    for i in range(0, m_length):
        while curIndex//ratio == i:
            symbols[i].append(y[curIndex+start])
            curIndex += 1

    energy = []
    for i in range(m_length):
```

```
        energy.append(list())

    for i in range(m_length):
        energy[i] = sum([number**2 for number in symbols[i]])

    decoded = np.array([int(number > e_thresh) for number in energy])
    decoded[0] = 1 # We have to do this because the start ramps up so its energy is lo

    b_errs = np.sum(decoded != message) / decoded.size

    # decoding

    print(energy[20:40])
    print(decoded[20:40])
    print(message[20:40])

    decoded = viterbi_decode(decoded)

    # error check
    print(f"bit error rate before decoding: {b_errs}")

    # I am going to keep this commented out because I doubt there will be many errors
    # data rate anyways
    #print(f"bit error rate after decoding: {np.sum(decoded != message) / message.size
    print(f"The messgae is: {bin_to_ascii(decoded[1:])}")
```

In [107…

```
# Here is the end result !!!
decode(message=x[1], x=x[0], rate=4, fs=africa_fs, threshold=0.005, e_thresh=3, fmin=0
```

```
bit error rate before decoding: 0.0
The messgae is: Hello World
```

Combining the Two Waves

> Now that we have the song we want to play with its frequencies removed and the wave for hello world with the carrier frequency 2.76kHz we can now combine the two.

In [183…

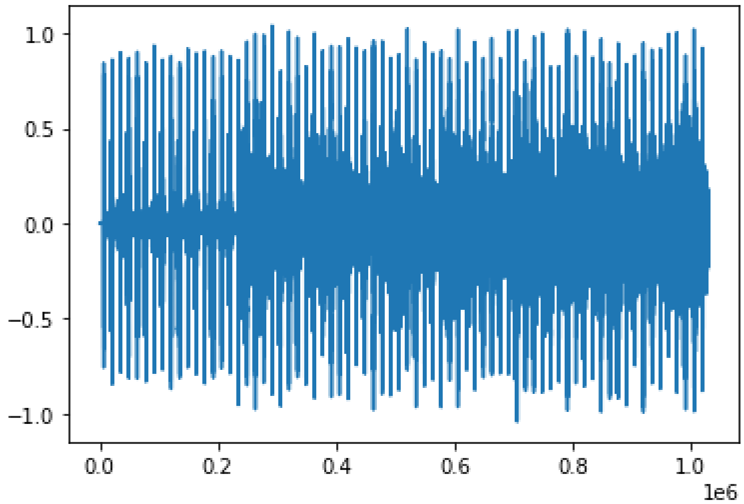
```
combined = (x[0] * 0.002) + africa_filtered[:len(x[0])]
```

In [184…

```
# Here is what the combined plot looks like
plt.plot(combined)
```

Out[184]:

```
[<matplotlib.lines.Line2D at 0x1900fce1a90>]
```



In [179…

```
# Here is what it sounds like
sd.play(combined[900000:], africa_fs)
```

Part 4: Decoding the Secret Message

Now we put the combined audio waves into the decoder. The only difficult part is tweaking certain parameters. I chose 2.5kHz and 3.1kHz as the cutoff frequencies just to further cut down on any noise. I decided to attenuate the beeping message to 0.2% of its original amplitude to make it less audio to the listener (this was done in the part just above) but I then set the multiplier to 500 to get it back around what its real amplitude would be.

But in the end it boils down to one function call.

In [ ]:

```
# This the final audio that plays
sd.play(combined, africa_fs)
```

In [199…

```
# Now the function decodes the hidden message and says hi !
decode(message=x[1], x=combined, rate=4, fs=africa_fs, threshold=0.3, e_thresh=400, fm
```

```
bit error rate before decoding: 0.0
The messgae is: Hello World
```

Part 5: Wrapping it in A Function and Conclusion

This functionality can be generalized to send any message at any rate. The function is a little touch ad go and the parameters have to be adjusted for each message (likely because the different parts of the song change in loudness and frequency) but any message can be discretly sent.

> Alas, another parameter that has to vary is the amplitude of the beepings and typically the longer the data is the higher it has to be. And because 2.76kHz is an audiable frequency it never really disappears. The beeping can be very buried but if the listener know what to look for they can hear it. I'd be willing to bet that if

> the amplitude is more than 0.5% its original value I can probably decode it by ear but it is still fun to send secret messages through Toto's africa.

In [210…
```
def send_message(message, rate=4, multiplier=500):
    divider = 1/multiplier
    x = waveform_gen(message, rate=rate, fc=2760, fs=africa_fs)
    combined = (x[0] * divider) + africa_filtered[:len(x[0])]
    decode(message=x[1], x=combined, rate=rate, fs=africa_fs, threshold=0.3, e_thresh=
```

In [208…
```
send_message("Goodbye ENGR76! It was fun!")
```

```
bit error rate before decoding: 0.0
Goodbye ENGR76! It was fun!
```

> And that is all. Now filtered music can hide audio messgaes !!

In [215…
```
# Also this is how I saved the wav files to submit !!

import scipy.io
scipy.io.wavfile.write('combined_audio.wav', africa_fs, combined)
```