Dial Tone

October 8, 2020

1 Dial tone example

1.1 Encoder

[2]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['figure.dpi'] = 200 # 200 e.g. is really fine, but slower
```

[3]:
```
DIALTONE_TABLE = {
    '1': (697, 1209),
    '2': (697, 1336),
    '3': (697, 1477),
    '4': (770, 1209),
    '5': (770, 1336),
    '6': (770, 1477),
    '7': (852, 1209),
    '8': (852, 1336),
    '9': (852, 1477),
    '*': (941, 1209),
    '#': (941, 1477)
}
LO_FREQS = np.array([697.0, 770.0, 852.0, 941.0])
HI_FREQS = np.array([1209.0, 1336.0, 1477.0])
```

[4]:
```
# this is the system clock, the rate at which we sample from the continous␣
 ↪signal to form our discrete approximation
# of the signal.
FS = 24_000
SIGNAL_LENGTH = 0.1 * FS
```

[5]:
```
def dial(number):
    full_signal = np.array([])
    # GENERATE an ndarray of integers that starts at 0 and ends before 2400
    n = np.arange(0, int(SIGNAL_LENGTH))
```

```
    # similarly, we generate a space of no signal that has the same length as␣
↪the signal.
    zeros = np.zeros(int(SIGNAL_LENGTH))
    for d in number:
        first = np.sin(2 * np.pi * DIALTONE_TABLE[d][0]/FS * n)
        second = np.sin(2 * np.pi * DIALTONE_TABLE[d][1]/FS * n)
        full_signal = np.concatenate((full_signal, first + second, zeros))
    return full_signal
```

[6]:
```
x=dial('123##45')

IPython.display.Audio(x, rate=FS)
```

[6]:
```
<IPython.lib.display.Audio object>
```

1.2 Decoder

when given the signal as input, the job of the decoder is to 1. find the sections that are not silent 2. do a dft on each of them 3. map those frequencies back to the table to get the original numbers

[7]:
```
def split_signal(signal):
    """
        signal: the full signal to split between tones and silence
        Assumptions:
        1. silence really is just the section where the amp is 0
    """
    # due to the nature of the sin fn, we oscillate which will produce a wrong␣
↪dft.
    # Let's look at windows this long
    window = 240
    signal = np.reshape(signal, (-1, window))
    # squaring takes of negative values (while keeping 0 at 0) and then we sum␣
↪within each window to 1 value.
    signal = np.sum(signal * signal, axis = 1)
    edges = []
    in_zero_region = True
    tone_section_start = 0
    for i,s in enumerate(signal):
        if s > 0:
            if in_zero_region:
                tone_section_start = i
                in_zero_region = False
        else:
            if not in_zero_region:
                edges.append((window * tone_section_start, i * window))
                in_zero_region = True
    return edges
```

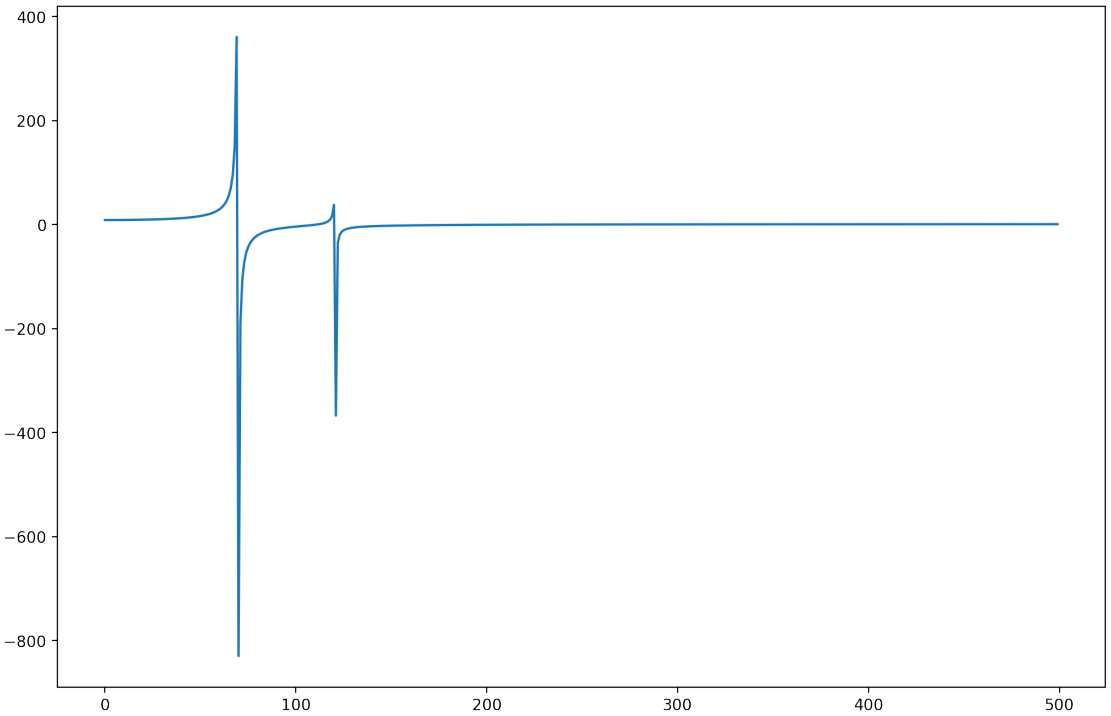
[8]:
```
split_signal(x)
```

[8]:
```
[(0, 2400),
 (4800, 7200),
 (9600, 12000),
 (14400, 16800),
 (19200, 21600),
 (24000, 26400),
 (28800, 31200)]
```

[10]:
```
print(len(x))
X = np.fft.fft(x[0:2400])
plt.plot(X[0:500]);
```

```
33600
```

```
/usr/lib/python3/dist-packages/numpy/core/_asarray.py:85: ComplexWarning:
Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



[13]:
```
def decode(encoded_signal):
    KEYS = [['1', '2', '3'], ['4', '5', '6'], ['7', '8', '9'], ['*', '0', '#']]

    HI_RANGE = (1180.0, 1500.0)
```

```
    dialed_number = []
    edges = split_signal(encoded_signal)
    for e in edges:
        fft_result = abs(np.fft.fft(encoded_signal[e[0]:e[1]]))
        res = FS / len(fft_result)
        # find the peak location within the low range
        a = int(680 / res)
        b = int(960 / res)
        lo = a + np.argmax(fft_result[a:b])

        # find the peak location within the high range
        a = int(1180 / res)
        b = int(1500 / res)
        hi = a + np.argmax(fft_result[a:b])

        # now match the results to the DTMF frequencies
        # by finding the closest match in each of LO_FREQS and HI_FREQS␣
↪discrete freqs
        row = np.argmin(abs(LO_FREQS - lo * res))
        col = np.argmin(abs(HI_FREQS - hi * res))
        dialed_number.append(KEYS[row][col])

    return dialed_number
```

[14]:
```
decode(x)
```

[14]:
```
['1', '2', '3', '#', '#', '4', '5']
```

[ ]:
```

```