Analysis of DTMF Implementations

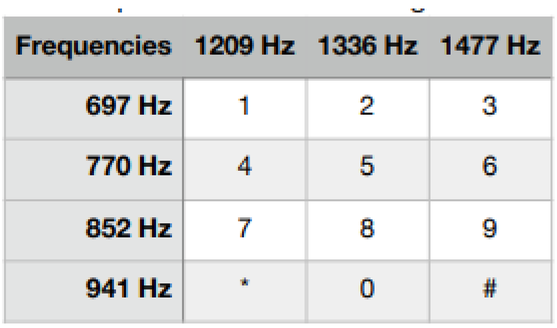
Justin Cata
Florida State University, USA
jcata@math.fsu.edu

***Abstract*—This paper discusses the implementation and analysis of DTMF methods for encoding and decoding "key-press" audio tones. The two implementation methods discussed are the bank of band-pass filters and the Goertzel Algorithm.**

.

I. Description of Mathematics

Telephone touch pads generate dual tone multi frequency (DTMF) signals to dial a telephone. When a key is pressed, the tones of the corresponding column and row (in Fig. 1)



Frequencies	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Figure 1: DTMF encoding table for Touch Tone dialling. When any key is pressed the tones of the corresponding column and row are generated and summed.

One way of decoding a DTMF signal is by passing the signal through a filter bank of FIR band pass filters constructed in parallel following an initial bifurcating low and high pass filter as shown in Fig. 2.

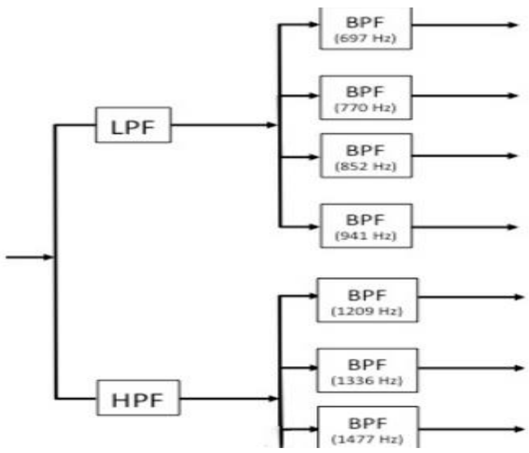


Figure 2:

are generated and summed, hence dual tone.

A bandpass filter is useful when the general location of the noise in the frequency domain is known. The bandpass filter allows frequencies within the chosen range through and suppresses frequencies outside of the given range.

[1] With a Butterworth bandpass filter, frequencies at the center of the frequency band are unattenuated and frequencies at the edge of the band are attenuated by a fraction of the maximum value. The Butterworth filter does not have sharp discontinuities between frequencies that are passed and filtered. The centered FFT is filtered by one of the following functions, where D_0 is the center of the frequency band, W is the width of the frequency band, $D=D(u,v)$ is the distance between a point (u,v) in the frequency domain and the center of the frequency rectangle, and n is the dimension of the Butterworth filter:

$$\text{Lowpass } (D_L = 0, D_H < 1): \quad H(u,v) = \frac{1}{1+[D/D_H]^{2n}}$$

$$\text{Bandpass } (D_L > 0, D_H < 1): \quad H(u,v) = 1 - \frac{1}{1+\left[(DW)/(D^2-D_0^2)\right]^{2n}}$$

$$\text{Highpass } (D_L > 0, D_H = 1): \quad H(u,v) = \frac{1}{1+[D_L/D]^{2n}}$$

Now the Goertzel method is a special case of the general Fast Fourier Transform. It is particularly useful for cases where we only need calculate a few DFT coefficients as in DFMT decoding. Unlike the DFT, the Goertzel algorithm can detect components without analyzing the entire spectrum. It is a recursive filter with a pair of poles on the unit circle at the frequency we want to detect. The filter has two stages:
[2]

The first stage calculates an intermediate sequence, $s[n]$:

$$s[n] = x[n] + 2\cos(\omega_0)s[n-1] - s[n-2].$$

The second stage applies the following filter to $s[n]$, producing output sequence $y[n]$:

$$y[n] = s[n] - e^{-j\omega_0}s[n-1].$$

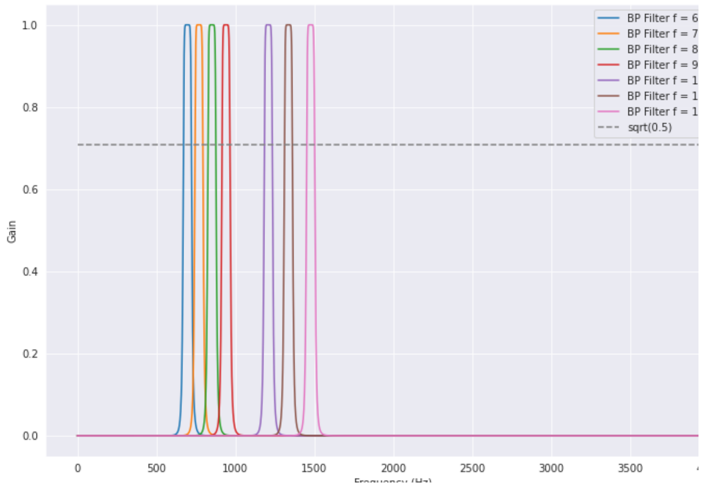
A useful analogy for understanding this algorithm is that it is in some ways a mix between an FFT filter and a band pass bank filter.

II. Description of the Implementation

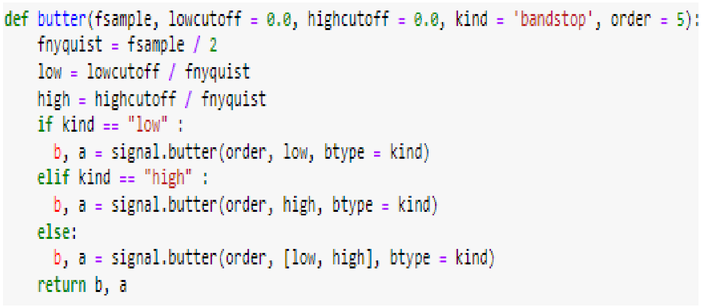
A. *Band Pass Filter Bank Method*

This implementation utilizes a bit of a divide and conquer approach where we first split the signal into tones by calling the function split_signal() and then move onto uncovering the two frequencies that make up the encoded digit.

We will pass each signal partition through a low pass and high pass filter to acquire two separate low frequency and high frequency signals respectively. Then the low and high frequency signals are respectively passed into four or three band pass filters to find which exact high and low frequency is present.



These filter implementations are all of type butterworth (Fig 3.) and carried out by the popular python package scipy and in particular the signal module.

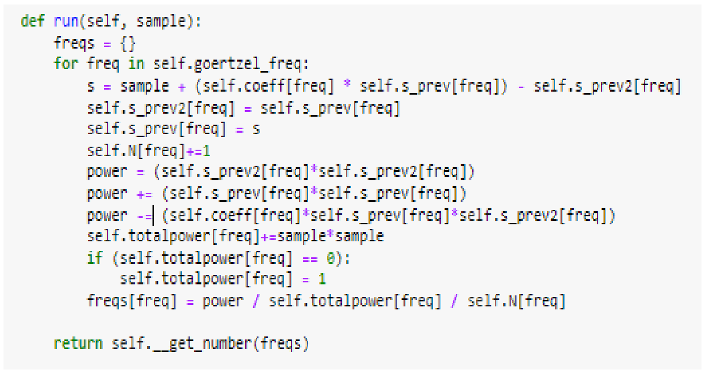


```
def butter(fsample, lowcutoff = 0.0, highcutoff = 0.0, kind = 'bandstop', order = 5):
    fnyquist = fsample / 2
    low = lowcutoff / fnyquist
    high = highcutoff / fnyquist
    if kind == "low" :
      b, a = signal.butter(order, low, btype = kind)
    elif kind == "high" :
      b, a = signal.butter(order, high, btype = kind)
    else:
      b, a = signal.butter(order, [low, high], btype = kind)
    return b, a
```

Figure 3: Pythonic implementation of a generic butterworth method utilized in generating low pass, high pass, band pass digital filters.

B. *Goertzel Algorithm*

The signal is split into segments like the band pass bank filter and each segment is analyzed for all the DTMF frequencies. In the interest of time, third party code was utilized for this implementation. This implementation utilizes an object-oriented paradigm but was cumbersome regarding number of lines so quite a bit was refactored by me for efficiency and readability. Below is the more relevant piece of the implementation [3]:



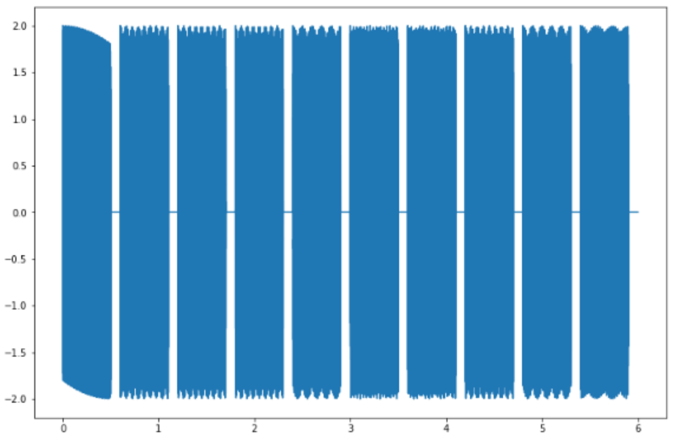
```
def run(self, sample):
    freqs = {}
    for freq in self.goertzel_freq:
        s = sample + (self.coeff[freq] * self.s_prev[freq]) - self.s_prev2[freq]
        self.s_prev2[freq] = self.s_prev[freq]
        self.s_prev[freq] = s
        self.N[freq]+=1
        power = (self.s_prev2[freq]*self.s_prev2[freq])
        power += (self.s_prev[freq]*self.s_prev[freq])
        power -= (self.coeff[freq]*self.s_prev[freq]*self.s_prev2[freq])
        self.totalpower[freq]+=sample*sample
        if (self.totalpower[freq] == 0):
            self.totalpower[freq] = 1
        freqs[freq] = power / self.totalpower[freq] / self.N[freq]

    return self.__get_number(freqs)
```

III. Results

A. *Well-Conditioned Signal*

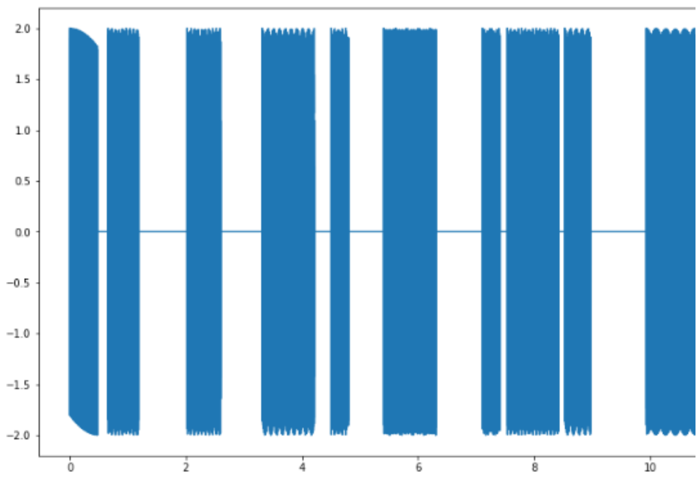
First a signal needed to be generated. The following signal was encoded: 3055967891. Below is the plot of the generated signal, note the equal spacing and lack of noise:



The statistic used for measuring correctness was number of correctly decoded digits over total number of digits. As expected of a well-conditioned signal, the test statistic after several sequences was 100%.

B. *Duration Perturbed Signal*

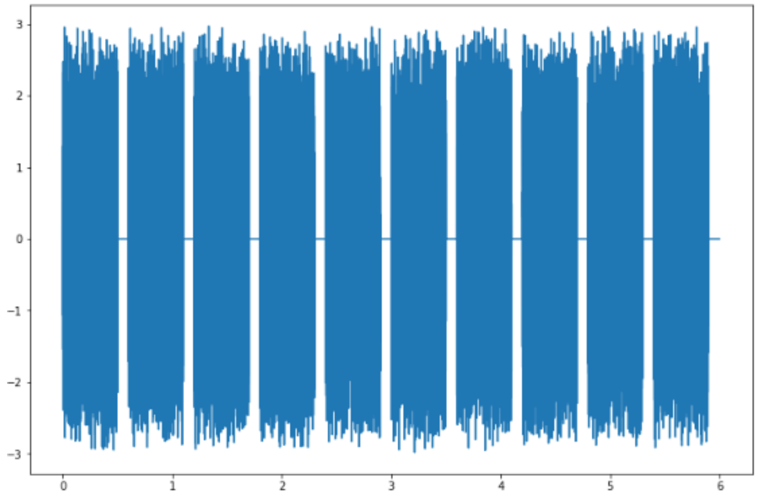
We want to test the correctness involved with varying the duration of both the tone and the blank space between each tone as shown below:



After several sequences, the test statistic returned 95.2% accuracy. The error involved is due to durations beings so long that digits get bucketed into multiple tone segments and when durations were so small the tone was virtually non existent in the bucket it fell in relative to the blank space.

C. *Noise Added Signal*

We want to continue testing robustness of the decoders by applying noise to the signals. When noise perturbed the data slightly, the decoder performed perfectly 100%, but as soon as the noise was significant, the decoder always missed the mark 0%.



IV. CONCLUSION

The set of frequencies used into DFMT implementations are chosen because the dual nature of high and low frequency signals is hard to imitate which helps prevent interference. The band pass filter bank algorithm and Goertzel algorithm are quite accurate at decoding digits and can be highly accurate at decoding sequences depending on the spacing and noise involved.

If the duration of the key press is larger than the signal splitting width, then the same digit will appear in two segments concluding in an extra decoded digit. If the duration is significantly less than the duration of the split, then it’s possible that two distinct digits will be lumped together in one segment and the decoding algorithm will either miss either digit or complete miss both digits while also returning a wrong digit. If the duration is so small, it virtually does not exist because it cannot be picked up.

When adding noise to the signal, if the noise is too powerful then the entire signal will be lost and only appear as noise. On the other hand, if the noise is not low enough, its possible it can trick the decoder into filtering out digits that aren’t there. One way of combating noise is to amplify the tones significantly enough to overshadow the present noise. If interference is superimposed on the signal, it can very well degrade the signal and render the decoder useless.

The computational burden of the band pass filter bank decoder is that for every digit to be decoded it must be passed through all filters.

REFERENCES

[1] L3HARRIS website. [Online]. Available: www.l3harrisgeospatial.com/

[2] Wikipedia website. [Online]. Available: wiki/Goertzel_algorithm

[3] Codeberg.org website. [Online]. Available: DTMF/src/branch/main/pygoertzel.py