Creating The Expected Signal, Morse Coefficients

By Terry Bondy, VA3TYB

In [1]:
```
printf(strftime ("Last updated: %A %e %B %Y", localtime (time ())))
```

```
Last updated: Sunday  1 December 2019
```

Background

In choosing an expected signal, we want to:

1. Spread the signal across the radio's audio bandwidth as much as is possible,
2. Make it identifiable when looking at a waterfall display, whether using LSB or USB, if the signal is visually detectable above the band noise,
3. Make it distinguishable from other identifiers used for the same purpose (i.e. VA3TYB, VA3ASE, VE3YRA all being different, visually and for the detector ultimeatly used).
4. Optimize properties of the signal for detection (more on this later).

To that end a message containing the operator's call sign at least once will be:

1. First converted to Morse represented by a row vector where each 1 represents a key down interval for a *dit* time and each 0 represents a key up interval for a *dit* time. For example [1 0 1 1 1] represents 'A'.
2. The row vector is concatenated with [0 0 0 0 0 0 0] representing a space, and then a left-to-right mirror image of itself. Taking the example for 'A' a bit further = [1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1].
3. The resulting row vector is spread evenly across the audio bandwidth of the radio, and if the value is 1, a sinusoidal tone will be produced on that frequency, and if the value is 0, no sinusoidal tone will be produced on that frequency. Taking the example further, if the radio had a bandwidth 200 - 3400 Hz, the message could be spread across the bandwidth at an interval of 200 Hz, so that tones would be sounded at 200 Hz, 600 Hz, 800 Hz, 1000 Hz, 2600 Hz, 2800 Hz, 3000 Hz, and 3400 Hz. No tones would be found on 400 Hz, 1200 Hz, 1400 Hz, 1600 Hz, 1800 Hz, 2000 Hz, 2200 Hz, 2400 Hz, and 3200 Hz.

The row vector so derived represents sound/silence coefficients for the following signal that is played by the radio:

$$played\ signal(t) = \sum_{i=0}^{bits-1} a_s(i)\ a_r(f)\ cos(2\pi t f + \phi_r(f) + \phi_d(i)),$$

where

$$f = (LF + i\frac{HF - LF}{bits - 1})$$

to result in an expected signal of:

$$expected\ USB\ signal(t) = \sum_{i=0}^{bits-1} a_s(i)\ e^{j2\pi tf + \phi_d(i)}$$

for USB and

$$expected\ LSB\ signal(t) = \sum_{i=0}^{bits-1} a_s(i)\ e^{-j2\pi tf + \phi_d(i)}$$

for LSB and where:

- $bits$ is the number of bits required by the message,
- $a_s(i)$ is the i^{th} sound/silent coefficient for the message, $a_s(i) \in [0, 1]$
- $a_r(f)$ is an amplitude coefficient required for compensating for the audio frequency dependent amplitude attenuation of the SignaLink and the radio, $a_r(f) \in R, a_r(f) > 0$
- $\phi_r(f)$ is a phase value required for compensating for the audio frequency dependent phase delay of the SignaLink and the radio, $\phi_r(f) \in R, 0 \leq \phi_r(f) < 2\pi$
- $\phi_d(i)$ is a detection optimizing phase value for the message, $\phi_d(i) \in R, 0 \leq \phi_d(i) < 2\pi$
- LF is the lower limit on the audio bandwidth of the radio, e.g. 400 Hz
- HF is the upper limit on the audio bandwidth of the radio, e.g. 2600 Hz

Radio Bandwidths

Yaesu FT-817

The manual of the FT-817 gives the audio SSB bandwidth of the radio as:

- 400 Hz-2600 Hz (–6 dB)

Yaesu FT-847

The manual of the FT-847 gives the audio SSB bandwidth of the radio as:

- 400 Hz - 2600 Hz (-6 dB)

Available Bandwidth

So the available bandwidth in Hz is:

```
In [2]: bw = 2600 - 400
```

```
bw =  2200
```

Number of Bits For Various Messages

VA3TYB

```
In [3]: 7 + 2 * size(alphaToMorse("QRG DE VA3TYB?"))(2)
```

```
ans =  297
```

```
In [4]: 7 + 2 * size(alphaToMorse("QRG DE VA3TYB VA3TYB?"))(2)
```

```
ans =  445
```

```
In [5]: 7 + 2 * size(alphaToMorse("QRG DE VA3TYB VA3TYB VA3TYB?"))(2)
```

```
ans =  593
```

VA3ASE

```
In [6]: 7 + 2 * size(alphaToMorse("QRG DE VA3ASE?"))(2)
```

```
ans =  269
```

```
In [7]: 7 + 2 * size(alphaToMorse("QRG DE VA3ASE VA3ASE?"))(2)
```

```
ans =  389
```

```
In [8]: 7 + 2 * size(alphaToMorse("QRG DE VA3ASE VA3ASE VA3ASE?"))(2)
```

```
ans =  509
```

VE3YRA

```
In [9]: 7 + 2 * size(alphaToMorse("QRG DE VE3YRA?"))(2)
```

```
ans =  289
```

```
In [10]: 7 + 2 * size(alphaToMorse("QRG DE VE3YRA VE3YRA?"))(2)
```

```
ans =  429
```

```
In [11]: 7 + 2 * size(alphaToMorse("QRG DE VE3YRA VE3YRA VE3YRA?"))(2)
```

```
ans =  569
```

VY0JJJ

Call sign chosen because it represents a maximally long Canadian morse callsign.

```
In [12]: 7 + 2 * size(alphaToMorse("QRG DE VY0JJJ?"))(2)
```

```
ans =  353
```

```
In [13]: 7 + 2 * size(alphaToMorse("QRG DE VY0JJJ VY0JJJ?"))(2)
```

```
ans =  557
```

```
In [14]: 7 + 2 * size(alphaToMorse("QRG DE VY0JJJ VY0JJJ VY0JJJ?"))(2)
```

```
ans =  761
```

Target Number of Bits

For a margin of "safety" (i.e. to get away from 6dB frequencies), take 80% only of BW, i.e.

```
In [15]: bw = .8 * (2600 - 400)
```

```
bw =  1760
```

Want the detector to work with a 0.25 second slice. So

```
In [16]: bits = bw/4 + 1
```

```
bits =  441
```

Back check

```
In [17]: (bits - 1) * 4
```

```
ans =  1760
```

Dual identifier message for VA3TYB will be tight. Lets determine percentage of bandwidth

```
In [18]: ((445 - 1) * 4 * 100)/(2600 - 400)
```

```
ans =  80.727
```

Close enough!

Summary

I have:

- Determined the format of the message to compute the sound/silence coefficients to yeild a number of bits that will allow the detector to work quickly enough that:
  - Hopefully Doppler shift changes should not effect the results of the detection
  - Minimize interference to other users of the band

  That format is "QRG DE *callsign callsign*?" with space (key up for 7 dit times) followed by the reverse image of itself.

- Determined the frequency spacing of the tones represented by the sound/silence coefficients. The frequency spacing is 4 Hz.

Coefficients

Morse coefficients to make up 551 bits can be computed using the following function. At 551 bits, the first sound/silence coefficient represents 400 Hz, the -6dB LF limit of the FT-817 and FT-847.

In [19]:
```
function toRet = makeMorseCoeff(message)
    baseCoeff = alphaToMorse(message);
    toRet = horzcat(baseCoeff, [ 0 0 0 0 0 0 0 ], flip(baseCoeff));
    sz = size(toRet)(2);
    # See if needs padding
    if (sz < 551)
        half = (551 - sz)/2;
        toRet = horzcat(zeros(1,half), baseCoeff, [ 0 0 0 0 0 0 0 ], flip(b
    endif
endfunction
```

In [ ]:
```
```