Using AI to Improve the Readability of Weak Beacons

Discover how a modest setup, Python programming, and *ChatGPT*-assisted code can recover 10-meter CW beacons from the noise.

Hal Feinstein, WB3KDU

I enjoy hunting for beacons on 10 meters. However, propagation on this band can make it challenging to hear beacons from certain parts of the world if the band isn't open in that direction. Ten meters is often open with weak or barely audible signals that are at or near the noise floor. These kinds of signals can be heard for a few seconds, then fall in strength, becoming nearly inaudible before dropping below the noise.

In an effort to improve weak signals by making them loud enough to recover the beacon's CW ID and message, I decided to look into simple signal-processing methods, which no longer require expensive custom equipment (or skilled technicians to build and operate it).

My requirements for this project were to use equipment I already had on hand, to learn what I could do with Python programming language signal-processing routines, and to not get bogged down learning the details of using these tools and the idiosyncrasies of the calling parameters. Artificial intelligence (AI) is being used to successfully write computer code, so I wanted to see if I could use it to write code for this project.

I focused on using digital signal processing (DSP) by filtering spectrum noise as close to the beacon signal as possible. A similar function is provided by *SDRplay* and *SDRuno* and is available with other software-defined radios (SDRs) and even analog radios. It's a good place to start to become familiar with equipment capabilities and procedures, as well as with programming DSP routines using AI.

There are many good reasons to process in-phase (I) and quadrature (Q) components (I/Q) versus basic audio, including spectrum capture, phase control, advanced demodulators, filter options, and several noise-reduction treatments. These are substantial advantages, but they're also considerably complex. This didn't fit with my goal of using a simple approach. (I want to try I/Q-format DSP after exploring what can be done with the audio signal.)

My programming was done in the Anaconda distribution of Python 3, and I used Jupyter Notebook

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

def plot_audio_spectrum(wav_file):
    # Read the WAV file
    sample_rate, data = wavfile.read(wav_file)

    # If stereo, take one channel
    if data.ndim > 1:
        data = data[:, 0]

    # Normalize data
    data = data / np.max(np.abs(data))

    # Compute the Fourier Transform
    n = len(data)
    fft_data = np.fft.fft(data)
    freqs = np.fft.fftfreq(n, d=1/sample_rate)

    # Get the magnitude spectrum
    magnitude = np.abs(fft_data)

    # Limit to the first 5 kHz
    limit = 3000
    indices = np.where(np.abs(freqs) <= limit)

    plt.figure(figsize=(12, 6))
    plt.plot(freqs[indices], magnitude[indices])
    plt.title('Audio Spectrum up to 3 kHz')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')
    plt.grid()
    plt.xlim(0, limit)
    plt.show()

# Replace 'your_audio_file.wav' with the path to your WAV file
plot_audio_spectrum('/Users/argo/Desktop/SDRuno_20240915_103635_
28274000HZ.wav')
```

Figure 1 — *ChatGPT*-generated Python code to read a .wav file and produce an audio spectrum of up to 3 kHz.

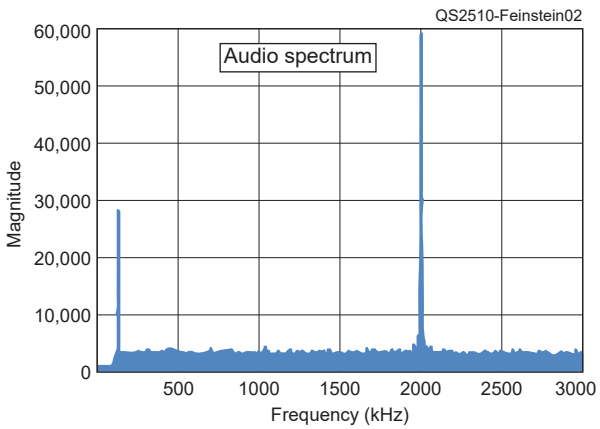

Figure 2 — Audio spectrum of the .wav file read by the program shown in Figure 1. The spike near 0 is ignored.

```
import numpy as np
import scipy.io.wavfile as wav
import matplotlib.pyplot as plt

def find_max_frequency(wav_file):
    # Read the WAV file
    sample_rate, data = wav.read(wav_file)

    # If stereo, take only one channel
    if data.ndim == 2:
        data = data[:, 0]

    # Normalize the data
    data = data / np.max(np.abs(data))

    # Perform FFT
    n = len(data)
    fft_data = np.fft.fft(data)
    fft_magnitude = np.abs(fft_data)

    # Frequency array
    freqs = np.fft.fftfreq(n, d=1/sample_rate)

    # Find the peak frequency
    peak_index = np.argmax(fft_magnitude[:n // 2]) # Only look at
        the positive frequencies
    peak_freq = freqs[peak_index]
    peak_magnitude = fft_magnitude[peak_index]

    return peak_freq, peak_magnitude

# Example usage
if _name_ == "_main_":
    wav_file = '/Users/argo/Desktop/SDRuno_20240915_103635_282740
    00HZ.wav'
    frequency, magnitude = find_max_frequency(wav_file)
    print(f'Max frequency: {frequency:.2f} Hz, Magnitude:
    {magnitude:.2f}')

Max frequency: 2000.77 Hz, Magnitude: 58907.04
```

Figure 3 — *ChatGPT*-generated Python code that finds the frequency of the strongest component in the audio spectrum of a .wav file.

```
import numpy as np
from scipy.io import wavfile
from scipy.signal import butter, lfilter

def butter_bandpass(lowcut, highcut, fs, order=5):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a

def bandpass_filter(data, lowcut, highcut, fs, order=5):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = lfilter(b, a, data)
    return y

def filter_wav_file(input_file, output_file, lowcut=1950.0,
    highcut=2200.0):
    #Read the WAV file
    sample_rate, data = wavfile.read(input_file)

    # Check if the data is stereo or mono
    if len(data.shape) > 1:
        data = data[:,0] # Take only one channel if stereo

    # Apply the bandpass filter
    filtered_data = bandpass_filter(data, lowcut, highcut, sample_
    rate)

    # Save the filtered audio to a new WAV file
    wavfile.write(output_file, sample_rate, filtered_data.
    astype(np.int16))
# Example usage
if _name_ == '_main_':
    input_wav_file = '/Users/argo/Desktop/SDRuno_20240918_112528_2
    8274000HZ.wav' # Replace with your input WAV file path
    output_wav_file = '/Users/argo/Desktop/output.wav' # Replace
    with your desired output file path

    # Set the cutoff frequencies
    lowcut = 1980.0 # Low cutoff frequency in Hz
    highcut = 2050.0 # High cutoff frequency in Hz

    filter_wav_file(input_wav_file, output_wav_file, lowcut,
    highcut)
```

Figure 4 — *ChatGPT*-generated Python code that reads a .wav file, applies a Butterworth band-pass filter on the audio, and writes the output to another .wav file.

(**https://jupyter.org**), a web application for creating and sharing computational documents, for code development. Python has sub-routine libraries that provide pre-written routines for various tasks. I used SciPy (which includes a signal library with Fast Fourier Transform, various filters, and helper functions), as well as Matplotlib (which provides versatile tools for showing signal and spectrum displays).

Modest Equipment

I used an *SDRplay* RSPdx receiver and *SDRuno* receiving software. The later versions of this software allow for capturing signals in audio and I/Q format. My initial experiments used received audio captured to .wav file format. *SDRuno* can also capture audio signals in MP3, which is a lossy compression scheme that is not suitable for this work, although it's much more compact than .wav format. With .wav, 60 seconds of audio becomes a 10-megabyte file. This means capturing more than short audio spans results in big .wav files, so that's something to keep in mind.

SDRuno runs on Windows, so I transferred my saved files to a thumb drive, which I then opened on my iMac for signal processing. I chose this method because it was quick.

Writing Python Programs with AI

One of my goals for this project was to spend most of my time trying various DSP methods, and less time writing code and debugging. In the past, this required programming skills, such as knowledge of DSP, the underlying DSP libraries, a programming language, and the language constructs that supported DSP data formats. AI large language models (LLMs), such as *ChatGPT*, *Copilot*, *Gemini*, and others, have been used to write programs that solve many types of tasks.

For this project, I used *ChatGPT-4o* to generate DSP programs in Python. (Keep in mind that AI models like *ChatGPT* occasionally generate programs with bad syntax. I discovered this when using it to generate a program in a different scripting language. When I asked *ChatGPT* if a specific statement it generated used the correct syntax, it fixed the error and rewrote the program.)

I developed three tools to try to improve the 10-meter beacon signal: a spectrum display, a program that reports the frequency of a peak signal, and a .wav-to-.wav file filter program. I used my iMac's .wav file player to listen for any improvements to the readability of the signal.

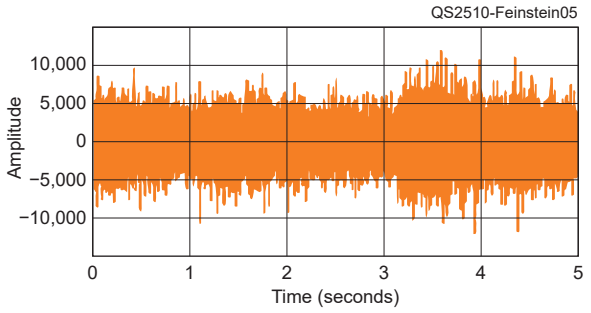

Figure 5 — The first 5 seconds of the original audio from the first beacon .wav file.

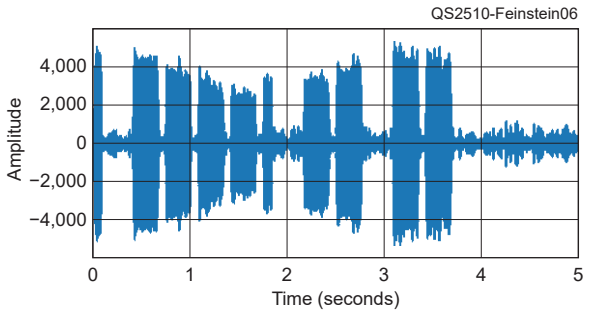

Figure 6 — The filtered output produced by program three. Here we see only the first 5 seconds of the filtered beacon .wav file filtered with a Butterworth audio band-pass filter shown in Figure 4. The CW letters visible are "9MM," which are part of the beacon's Maidenhead coordinate, EK19MM (near Mexico City, Mexico). Notice the scale difference between Figures 5 and 6.

Program One — Display the Audio Spectrum

Using the *SDRplay* RSPdx and *SDRuno* audio, I captured CW beacon signals in two .wav files, in which the beacon had some reasonably strong portions followed by weak and "washed-out" sections. Concentrating on these sections, I used *ChatGPT* to write code for and use an audio spectral display to get an idea of what the .wav file spectrum looked like to try to improve the signal.

Creating the audio spectrum display program using *ChatGPT* is straightforward. In the *ChatGPT* dialog box, enter: "Write a program that reads a .wav file and displays the first 3 kHz of its audio spectrum." The results are shown in Figure 1. The file name was changed to one of the captured .wav files, but otherwise, it's the exact code output by *ChatGPT*. The Python program code is well documented. *ChatGPT* added a brief explanation of what each part of the program does. The spectral display output is shown in Figure 2.

The audio tone generated by *SDRplay* and *SDRuno* occurs because the beacon signal (which is sent using unmodulated CW) is being received in sideband mode.

Program Two — Report the Peak Frequency

Next, I had *ChatGPT* write a program that reports the frequency of the strongest signal component in the audio spectrum. Figure 3 shows the code with the resulting strongest frequency listed at bottom left.

In program one's spectrum display there was a significant frequency component somewhere around 2 kHz, but you can't tell if it's above or below 2 kHz, or by how much. This program reports the file's peak value, showing a frequency of 2000.77 Hz. The fractional 0.77 Hz can be dropped, concentrating only on the 2 kHz value.

Program Three — Butterworth Audio Filter

The third program reads the .wav file and applies a Butterworth band-pass filter that removes noise at frequencies below and above a specific band-pass cutoff. A small section of spectrum showing the 2 kHz signal is what's left.

In the *ChatGPT* dialog box, enter: "Write a Python program to read a .wav file and filter it so only the frequencies between 1900 kHz and 2200 kHz are passed, and write it out as an output .wav file." The program is shown in Figure 4. Figures 5 and 6 show the output before and after the filter.

Parting Thoughts

This project successfully used simple signal-processing procedures to improve the readability of a weak beacon signal. Figure 6 shows the filtered beacon signal with part of its location coordinates being decipherable. The audio quality of the signal is significantly improved as well, extending the ability to read the beacon transmission at least partially down into the noise.

This was my first time using *ChatGPT* to write programs that support simple signal-processing tasks. The tasks were relatively simple, but I'd like to see how far this approach to program construction can go.

The techniques used in this article open new doors to amateur radio experimenters. Experimenting with code developed by *ChatGPT* is limited only by your imagination!

Hal Feinstein, WB3KDU, can be reached at **wbkdu588@gmail.com**.

For updates to this article, see the *QST* Feedback page at www.arrl.org/feedback.