University of Ljubljana
Faculty of mathematics and physics
Department of physics

Data Acquisition and Processing

Designing and implementing a near real-time bandpass Hilbert transformer

Author:
Elijan Jakob Mastnak

Student ID:
28181157

Assignment

Use the window method with an appropriate window function to design an FIR band-pass Hilbert transform filter for use with signals sampled at $f_s = 44\,100\,\text{Hz}$. The filter passband should span the frequency range from 1000 Hz to 2000 Hz; in the passband, the input and output signals should differ in phase by $\pi/2$ (as for a Hilbert transformer) and in amplitude by no more than one 1 dB. For frequencies below 500 Hz and above 2500 Hz, the output signal's amplitude should be at least 40 dB less than the input amplitude. Compute the FIR filter's coefficients, test the filter in an offline simulation, and finally ensure the filter works in real time.

Ljubljana, September 2021

Contents

1 Computing the Filter's Impulse Response

1.1 Choosing a Window Function

We select a window function from the requirement that the filter's stopband attenuation be less than $-40\,\text{dB}$, which requires a filter with peak approximation error (PAE) less than $-40\,\text{dB}$ [3, 6]. A number of common window functions are listed in Table 2; of these, the Hann, Hamming and Blackman windows all have a PAE less than $-40\,\text{dB}$. In principle, any of these would work; I chose a Hann window to avoid over-designing the filter, i.e. to avoid increasing the filter's transition band width in exchange for larger-than-required stopband attenuation.

Frequency f	Attenuation
$f < 500\,\text{Hz}$	$\lvert H\rvert < -40\,\text{dB}$
$1000\,\text{Hz} < f < 2000\,\text{Hz}$	$\lvert H\rvert = (0 \pm 1)\,\text{dB}$
$f > 2500\,\text{Hz}$	$\lvert H\rvert < -40\,\text{dB}$

Table 1: The problem's filter specifications.

We estimate an appropriate number of window coefficients M by equating the known Hann window main lobe width of $8\pi/M$ (from Table 2) to the filter's specified transition band width $\Delta\omega$ in units of normalized frequency. First, referring to the specifications in Table 1, the filter's transition band width[1] is

$$\Delta f = 1000\,\text{Hz} - 500\,\text{Hz} = 2500\,\text{Hz} - 2000\,\text{Hz} = 500\,\text{Hz}.$$

The corresponding transition band width in units of normalized frequency is

$$\Delta\omega = 2\pi\frac{\Delta f}{f_\text{s}} = 2\pi\frac{500\,\text{Hz}}{41\,000\,\text{Hz}} = \frac{\pi}{41}.$$

Using the just-computed transition band width $\Delta\omega$, an appropriate number of filter coefficients M is

$$\frac{8\pi}{M} = \Delta\omega = \frac{\pi}{41} \implies M = 328. \tag{1}$$

I implemented the actual filter with $M + 1 = 329$ coefficients, since an odd-length impulse response is more conducive to implementing a bandpass filter.[2]

Window Function	MLW	PAE [dB]
Rectangular	$\frac{4\pi}{M+1}$	-21
Hann	$\frac{8\pi}{M}$	-44
Hamming	$\frac{8\pi}{M}$	-53
Blackman	$\frac{12\pi}{M}$	-74

Table 2: Approximate main lobe width (MLW), in units of normalized frequency, and peak approximation error (PAE) of common window functions. M denotes the number of coefficients used to implement the window (and filter). Taken from [6].

[1] In this problem the filter's upper and lower transition widths are both 500 Hz. If they were different, we would use the smaller (i.e. more restrictive) of the two.

[2] Specifically, implementing a Hilbert transform produces an antisymmetric impulse response, and the frequency response of filter with odd-length, antisymmetric impulse response has zeros at $f = 0\,\text{Hz}$ (DC) and at $f = f_\text{s}/2$ (the Nyquist rate) [3]. This zero configuration is well-suited to a bandpass filter.

1.2 Computing Filter's Impulse Response

Below is an outline of the process used to compute the filter's impulse response:

1. Compute the impulse response h_{lp} of an ideal *lowpass* filter.
2. Write the impulse response h_{pb} of an ideal *bandpass* filter as the difference of two lowpass impulse responses.
3. Separately compute the impulse response h_{H} of an ideal Hilbert transformer.
4. Write the complete filter's ideal impulse response as the convolution $h_{\mathrm{bp}} * h_{\mathrm{H}}$.
5. Truncate and window the ideal impulse response to get the finite-length impulse response actually implemented in software.

Ideal Lowpass Impulse Response

The frequency response of an ideal lowpass filter with continuous-time cutoff frequency Ω_{c} is the rectangular function

$$H_{\mathrm{lp}}(i\Omega) = \begin{cases} 1 & -\Omega_{\mathrm{c}} < \Omega < \Omega_{\mathrm{c}} \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding continuous-time impulse response is

$$\begin{aligned} h_{\mathrm{lp}}(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} H(i\Omega)e^{i\Omega t}\,\mathrm{d}\Omega = \frac{1}{2\pi}\int_{-\Omega_{\mathrm{c}}}^{\Omega_{\mathrm{c}}} e^{i\Omega t}\,\mathrm{d}\Omega = \frac{1}{2\pi i t}\Big[e^{i\Omega t}\Big]_{-\Omega_{\mathrm{c}}}^{\Omega_{\mathrm{c}}} \\ &= \frac{1}{2\pi i t}\big(e^{i\Omega_{\mathrm{c}} t} - e^{-i\Omega_{\mathrm{c}} t}\big) = \frac{\sin\Omega_{\mathrm{c}} t}{\pi t} \\ &= \frac{\Omega_{\mathrm{c}}}{\pi}\operatorname{sinc}(\Omega_{\mathrm{c}} t). \end{aligned}$$

We convert the continuous-time impulse response to discrete time (assuming sample rate $f_{\mathrm{s}} = 1/T$) by changing $t \to nT$ and including an amplitude scaling factor $T = 1/f_{\mathrm{s}}$, i.e.

$$h_{\mathrm{lp}}[n] = T \cdot \frac{\Omega_{\mathrm{c}}}{\pi}\operatorname{sinc}[\Omega_{\mathrm{c}} nT] = \frac{2f_{\mathrm{c}}}{f_{\mathrm{s}}}\operatorname{sinc}\left[\frac{2\pi f_{\mathrm{c}}}{f_{\mathrm{s}}}n\right]. \tag{2}$$

Ideal Bandpass Impulse Response

The impulse response of an ideal *bandpass* filter with high and low passband frequencies f_{high} and f_{low} may be constructed as the difference of two lowpass filters with cutoff frequencies f_{high} and f_{low}, respectively, i.e.

$$\begin{aligned} h_{\mathrm{bp}}[n] &= h_{\mathrm{lp}}[n; f_{\mathrm{high}}] - h_{\mathrm{lp}}[n; f_{\mathrm{low}}] \\ &= \frac{2f_{\mathrm{high}}}{f_{\mathrm{s}}}\operatorname{sinc}\left[\frac{2\pi f_{\mathrm{high}}}{f_{\mathrm{s}}}n\right] - \frac{2f_{\mathrm{low}}}{f_{\mathrm{s}}}\operatorname{sinc}\left[\frac{2\pi f_{\mathrm{low}}}{f_{\mathrm{s}}}n\right]. \end{aligned}$$

This impulse response has a removable discontinuity at $n = 0$ which must be treated separately in a computer implementation. Using the limit $\lim_{x\to 0}\operatorname{sinc} x = 1$, this is

$$h_{\mathrm{bp}}[0] = \frac{2}{f_{\mathrm{s}}}(f_{\mathrm{high}} - f_{\mathrm{low}}).$$

Ideal Hilbert Transformer Impulse Response

The frequency response of an Hilbert transformer for frequencies in the range $(-\Omega_0, \Omega_0)$ is

$$H_{\mathrm{H}}(i\Omega) = \begin{cases} e^{-i\frac{\pi}{2}} & -\Omega_0 < \Omega < 0 \\ e^{+i\frac{\pi}{2}} & 0 < \Omega < \Omega_0 \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding continuous-time impulse response is

$$\begin{aligned} h_{\mathrm{H}}(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} H(i\Omega)e^{i\Omega t}\,\mathrm{d}\Omega = \frac{1}{2\pi}\int_{-\Omega_0}^{0} e^{-i\frac{\pi}{2}}e^{i\Omega t}\,\mathrm{d}\Omega + \frac{1}{2\pi}\int_{0}^{\Omega_0} e^{+i\frac{\pi}{2}}e^{i\Omega t}\,\mathrm{d}\Omega \\ &= \frac{1}{2\pi i t}\left[e^{+i(\Omega_0 t+\pi/2)} - e^{-i(\Omega_0 t+\pi/2)}\right] - \frac{1}{2\pi i t}\left(e^{+i\frac{\pi}{2}} - e^{-i\frac{\pi}{2}}\right) \\ &= \frac{1}{\pi t}\sin(\Omega_0 t + \pi/2) - \frac{\sin(\pi/2)}{\pi t} \\ &= \frac{1}{\pi t}\big[\cos(\Omega_0 t) - 1\big]. \end{aligned}$$

As in Equation 2 for the lowpass filter, we convert to discrete time by changing $t \to nT$ and including an amplitude scaling factor $T = 1/f_{\mathrm{s}}$, i.e.

$$h_{\mathrm{H}}[n] = \frac{T}{\pi n T}\big(\cos[\Omega_0 n T] - 1\big) = \frac{1}{\pi n}\left(\cos\left[\frac{2\pi f_0}{f_{\mathrm{s}}}n\right] - 1\right).$$

If the Hilbert transformer is to apply to the entire discrete frequency domain, i.e if $f_0 = f_{\mathrm{s}}/2$, the impulse response simplifies considerably to

$$\begin{aligned} h_{\mathrm{H}}[n] &= \frac{1}{\pi n}\left(\cos\left[\frac{2\pi f_{\mathrm{s}}}{2f_{\mathrm{s}}}n\right] - 1\right) = \frac{1}{\pi n}\big(\cos[\pi n] - 1\big) \\ &= \frac{1}{\pi n}\big[(-1)^n - 1\big]. \end{aligned}$$

However, this problem's specifications require the filter shift phase by $\pi/2$ only in the passband, so I choose a Hilbert transform cutoff frequency $f_0 = 3000\,\mathrm{Hz}$ instead of using the Nyquist rate $f_0 = f_{\mathrm{s}}/2 = 22\,500\,\mathrm{Hz}$.

Complete Ideal Impulse Response

Convolution is commutative and associative, which simplifies the practical implementation of the filter's impulse response. Namely, processing an input signal, say x, with a bandpass filter with impulse response h_{bp} and then passing the filtered signal, say x_{f}, through a Hilbert transformer with impulse response h_{H} is equivalent to passing the input x through a single system whose impulse response is the convolution of h_{bp} and h_{H}. This is shown symbolically in Equation 3 and graphically in Figure 1.

$$y = h_{\mathrm{H}} * x_{\mathrm{f}} = h_{\mathrm{H}} * (h_{\mathrm{bp}} * x) = (h_{\mathrm{H}} * h_{\mathrm{bp}}) * x \equiv h_{\mathrm{total}} * x. \tag{3}$$

The complete filter's ideal impulse response is thus

$$h_{\mathrm{ideal}} = h_{\mathrm{bp}} * h_{\mathrm{H}}.$$

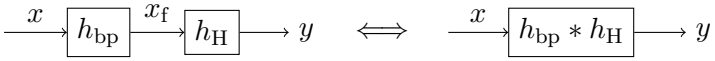

Figure 1: Convolution's associativity simplifies the filter implementation: the left and right systems produce identical outputs, but if $h_{\mathrm{bp}} * h_{\mathrm{H}}$ is computed beforehand, implementing the right version in real-time requires half as many convolutions. C.f. Equation 3.

Windowed Impulse Response

The discrete Hann window function for a filter with $M + 1$ coefficients is

$$w[n] = \begin{cases} \frac{1}{2}\left(1 + \cos\left[\frac{2\pi n}{M}\right]\right) & n = -\frac{M}{2}, \ldots, \frac{M}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding causal window, with center shifted to $n = M/2$, is

$$w_{\mathrm{causal}}[n] = w[n - M/2] = \begin{cases} \frac{1}{2}\left(1 - \cos\left[\frac{2\pi n}{M}\right]\right) & n = 0, \ldots, M \\ 0 & \text{otherwise.} \end{cases}$$

The windowed impulse response is

$$h[n] = h_{\mathrm{ideal}}[n - M/2] w_{\mathrm{causal}}[n].$$

The code used to compute $h[n]$ can be found in the `get_h` function in the file `kernels.py` in the project's source code repository on GitHub at https://github.com/ejmastnak/hilbert-bandpass.

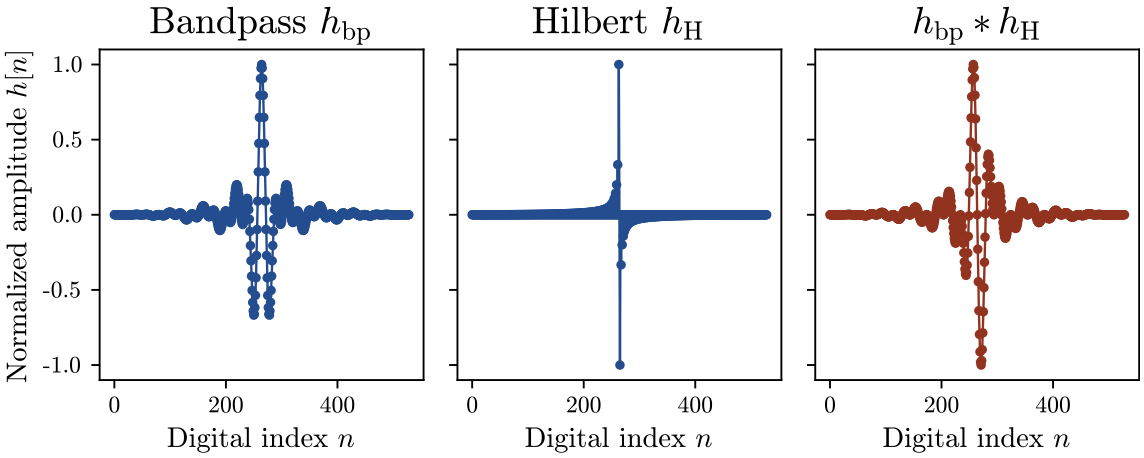

Figure 2: Windowed impulse responses of the bandpass and Hilbert transformer stages (left and center), together with their convolution (right).

1.3 Visualizing the Filter's Impulse and Frequency Responses

Figure 2 shows the windowed impulse responses of the bandpass stage, the Hilbert transformer stage, and the convolution of the two stages. Figure 3 shows the frequency response of the convolved bandpass-Hilbert system both with and without windowing. As expected theoretically, windowing the impulse response produces a far smoother frequency response and better stopband attenuation at the expense of a wider transition band. Figure 4 attempts to demonstrate that the filter meets the specifications in Table 1. Finally, Figure 5 shows the filter's phase response—the discontinuity in phase angle at $f = 0\,\mathrm{Hz}$ and $\theta \approx -45°$ shows the phase-shifting effect of the Hilbert transformer.

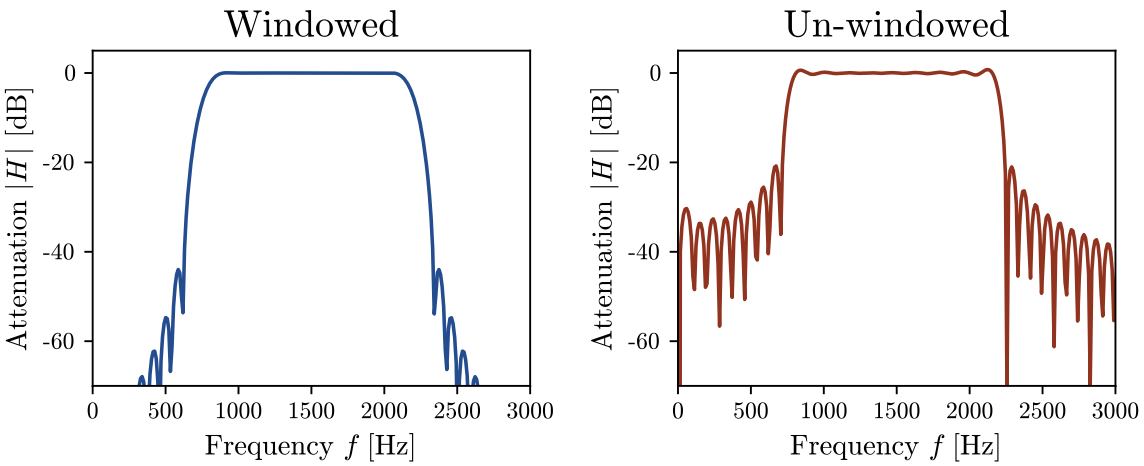

Figure 3: Windowed and un-windowed frequency responses.

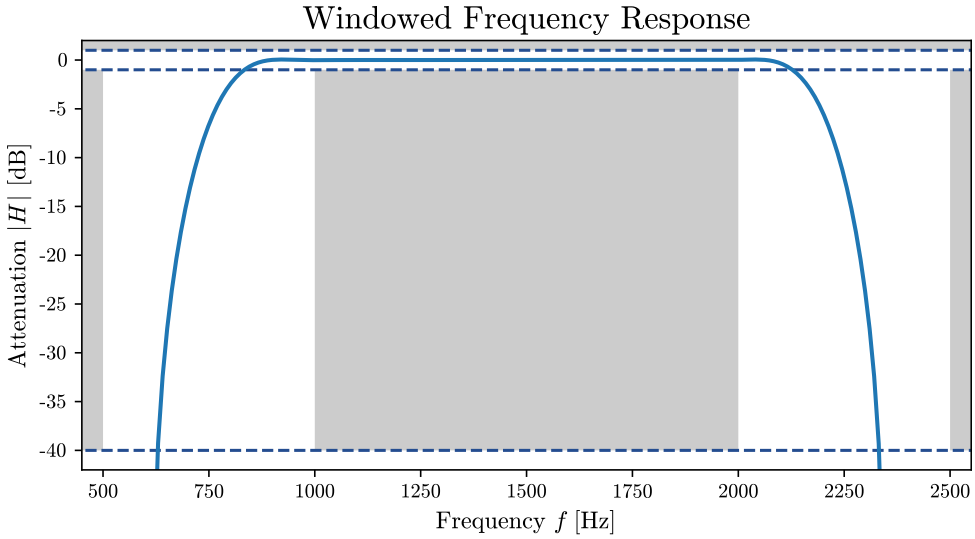

Figure 4: The filter's frequency response in the transition and passband regions. The curve must not touch the shaded grey areas to meet the filter specifications in Table 1.

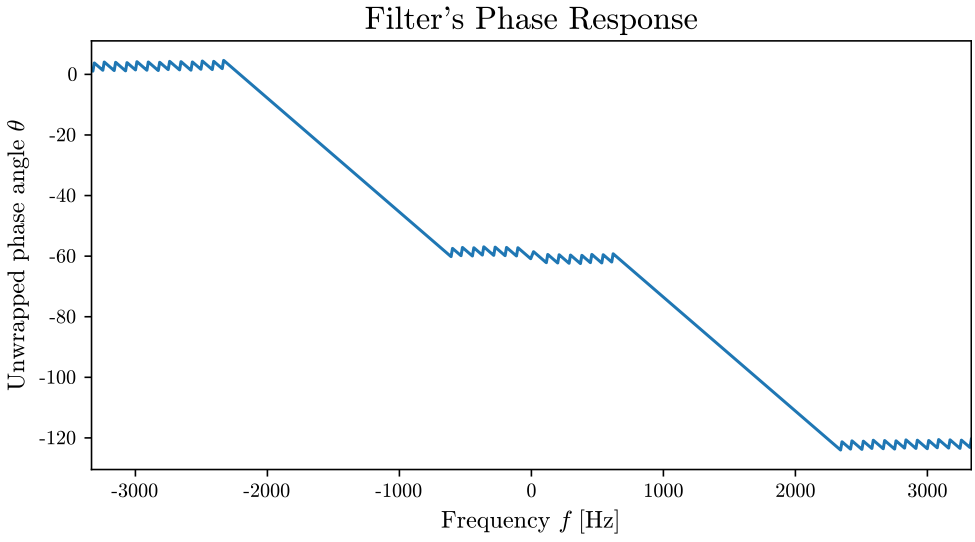

Figure 5: The filter's phase response. The phase is linear in the passband, while the discontinuity at $f = 0\,\text{Hz}$ corresponds to the Hilbert transformer phase shift.

2 Implementing Real-time Filtering

The hardware and software used to implement the filter is summarized below:

- Computer: 13-inch, 2012 MacBook Pro laptop with a 2.5 GHz Intel Core i5 processor running macOS 10.14.6. Audio was captured using the laptop's built-in 44 100 Hz, 2-channel microphone, but only one channel was used as filter input.
- I used version 19 of the `PortAudio` library [1], which is an audio I/O library written in C, to capture the audio input stream from the computer's microphone.
- I wrote the filter program in Python, using the `PyAudio` library [5] for Python bindings for the C code in `PortAudio` and the `NumPy` library [7] for efficient implementation of the fast Fourier transform, convolution, and common mathematical functions.

The full project source code and a video demonstration of the filter's real-time functionality can be found on GitHub at https://github.com/ejmastnak/hilbert-bandpass, while a summary of how the real-time filtering works follows below.

Audio data is captured from the microphone using `PortAudio` with an input stream with the specifications shown in Table 3. The number of frames per buffer is chosen so that the fast Fourier transform used to convolve the impulse response with the audio buffer has 4096 elements (i.e. a power of two, which is a general best practice for FFT algorithms). The initialization of the audio stream is shown in Listing 1.

Parameter	Value
Sample rate	44 100 Hz
Format	32-bit floating point
Channels	1
Frames per buffer	$4096 - \texttt{len(h)} + 1$

Table 3: Audio stream parameteters. `len(h)` is the length of the filter's impulse response.

```
import numpy as np
import pyaudio
h = get_impulse_response()  # load filter's impulse response
CHUNK = 4096 - len(h) + 1   # frames per buffer
pa = pyaudio.PyAudio()
stream = pa.open(format=pyaudio.paFloat32,  # 32-bit floating point
        channels=1,  # use only single-channel audio
        rate=44100,  # sample rate [Hz]
        input=True,  # use stream as an input stream
        frames_per_buffer=CHUNK)
stream.start_stream()
```

Listing 1: Source code for initializing the audio stream.

Data is read from the audio stream as a hexadecimal Python byte string, which is then converted to an array of 32-bit floating point numbers with `NumPy`'s `frombuffer` function. For orientation, a representative audio stream output buffer is shown in Listing 2. The real-time filtering process works by convolving each buffer of audio data with the filter's impulse response using the overlap-add method [4]. The code implementing the overlap-add method, with explanatory comments, appears in Listing 3.

```
>>> buffer = stream.read(10)  # read 10 frames
b'\x00\x82\x00\xe0\x9a\x00\x8c\xb0\x00<\x97\x00\xa0\x9c\x00\xfc\x92\x00
↪  \xc4\xbb\x00\x00\xf8\xa3\x00\x10'
>>> buffer_float32 = np.frombuffer(data, dtype=np.float32)
[0.2548523  0.30249023 0.34481812 0.29537964 0.3059082  0.28707886
↪  0.36672974 0.23553467 0.32025146 0.22174072]
```

Listing 2: Representative audio stream output as a byte string and corresponding 32-bit floating point array (which has been rounded for conciseness).

One specific feature of the code in Listing 3 deserves special comment. First, here is the context: the real-time filtering program listens for input audio data and displays a continuously-updated plot of both the unfiltered input audio waveform and its bandpass-filtered and Hilbert-transformed version. The input and filtered signal are intentionally shown together on the same time-domain plot to verify the filter's phase-shifting property, as in e.g. Figure 6. Resultantly, it is important that the unfiltered input signal and its filtered version correctly align in the time domain. FIR filtering inherently introduces a time delay to the filtered signal equal to the length M of the FIR filter kernel. To compensate for this delay, the code in Listing 3 creates a delayed copy of the unfiltered input signal by convolving the raw input signal with a unit impulse centered at $M/2$. This delayed version of the input signal and its FIR-filtered analog are then correctly aligned in the time domain.

3 Representative Filtered Signals

The following few figures were generated offline, all with sample rate $f_s = 44\,100\,\text{Hz}$, and are meant to demonstrate the filter's desired phase shift and bandpass properties in an offline scenario. A video showing the filter working with real-time audio signals can be found on the project's GitHub page.

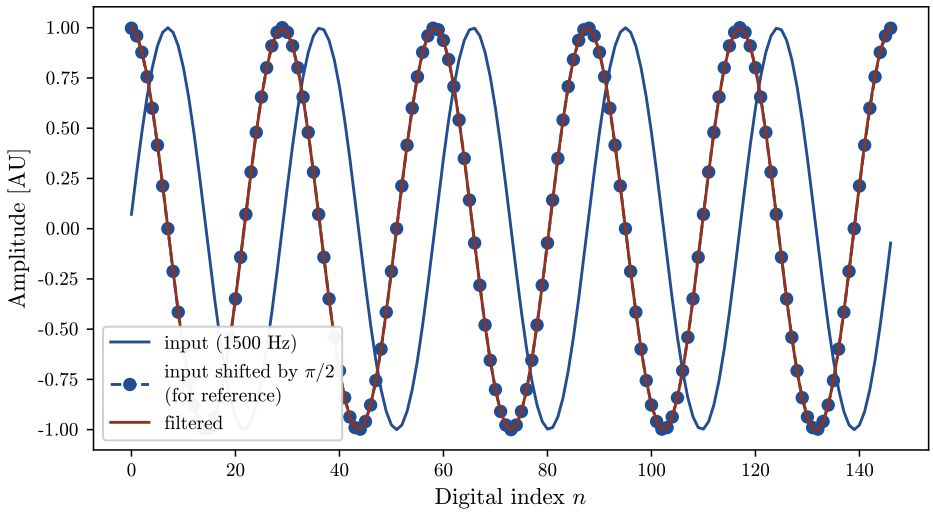

Figure 6: Demonstration of the filter's phase-shifting property with a 1500 Hz sinusoid. The dotted curve with blue circular markers is a copy of the input signal shifted by $\pi/2$, which the filtered output (in maroon-red) aligns with nicely, as it must to satisfy the phase-shifting property of a Hilbert transformer.

```python
import numpy as np

# Load filter's impulse response from separate function
h = get_impulse_response()
M = len(h)  # length of impulse response

# Create a delay-by-M kernel to compensate FIR-induced delay
h_delay_by_M = np.zeros(M, dtype=np.float32)
h_delay_by_M[int((M - 1)/2)] = 1.0

# Declare global variable to store last `M` points of previous
# iteration's convolution of filter kernel and audio buffer
filter_conv_prev = np.zeros(M - 1)  # preallocate with zeros

# Global variable to store last `M` points of previous
# iteration's convolution of delay-by-M kernel and audio buffer
conv_delayed_prev = np.zeros(M - 1)

def get_filtered_audio_buffer(buffer_in):
    """
    Applies the Hilbert transform filter to the inputted audio buffer
    `buffer_in`. Returns filtered output `buffer_out` and delayed
    copy of `buffer_in` aligned in time with `buffer_out`.
    """
    L = len(buffer_in)
    # Convolve kernel `h` and `buffer_in` and store first `L` points
    filter_conv = np.convolve(h, buffer_in)
    buffer_out = filter_conv[:L]

    # Append last `M-1` elements of previous iteration's convolution
    # to first `M-1` elements of current convolution
    buffer_out[:M - 1] += filter_conv_prev

    # Save last `M-1` elements of this iteration's convolution for
    # use in the next iteration
    filter_conv_prev = filter_conv[-(M - 1):]

    # Create delayed copy of `buffer_in` aligning with `buffer_out`
    delay_conv = np.convolve(h_delay_by_M, buffer_in)
    buffer_in_delayed = delay_conv[:L]
    buffer_in_delayed[:M - 1] += delay_conv_prev
    delay_conv_prev = delay_conv[-(M - 1):]

    return buffer_in_delayed, buffer_out
```

Listing 3: The function `get_filtered_buffer` implements the overlap-add method used for real-time filtering. Complete code for real-time filtering is in the `realtime.py` file in the project's source code repository, https://github.com/ejmastnak/hilbert-bandpass.

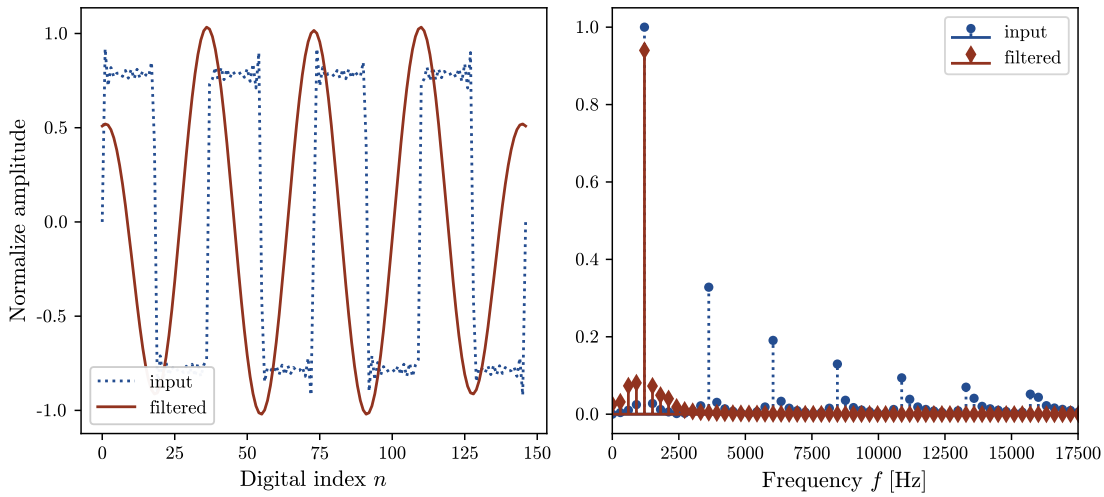

Figure 7: The filter's time domain and frequency domain output in response to a 10-term Fourier series approximation of a square wave with fundamental frequency $f_0 = 1200\,\text{Hz}$. All higher harmonics are filtered out, leaving only the 1200 Hz sinusoidal carrier wave in the filtered output. Note also the 90° phase shift between the input and output signals.

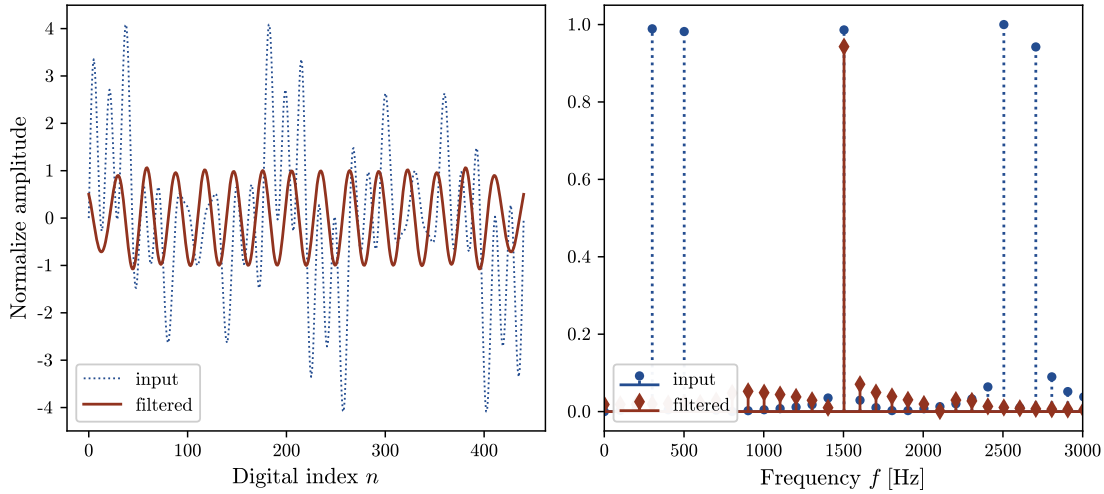

Figure 8: The filter's time domain and frequency domain responses to a signal containing sinusoidal terms with frequencies $(300, 500, 1500, 2500, 2700)\,\text{Hz}$, all of equal amplitude. Only the 1500 Hz component in the passband is passed through the filter.

References

[1] Ross Bencina and Phil Burk. *PortAudio: Portable Real-Time Audio Library*. Version 19. Aug. 31, 2021. URL: http://www.portaudio.com/.

[2] Alan Oppenheim. *RES.6-008 Digital Signal Processing*. https://ocw.mit.edu. 2011. URL: https://ocw.mit.edu/resources/res-6-008-digital-signal-processing-spring-2011.

[3] Sophocles Orfanidis. "FIR Digital Filter Design". In: *Introduction to Signal Processing*. Pearson, 1996, pp. 532–562. URL: http://eceweb1.rutgers.edu/~orfanidi/intro2sp/.

[4] Sophocles Orfanidis. "Overlap-Add Block Convolution Method". In: *Introduction to Signal Processing*. Pearson, 1996, pp. 143–146. URL: http://eceweb1.rutgers.edu/~orfanidi/intro2sp/.

[5] Hubert Pham. *PyAudio*. Version 0.2.11. Aug. 31, 2021. URL: https://people.csail.mit.edu/hubert/pyaudio/.

[6] John Proakis and Dimitris Manolakis. "Design of Digital Filters". In: *Digital Signal Processing: Principles, Algorithms, and Applications*. Third. Prentice-Hall, 1996, pp. 614–737.

[7] The Numpy Project. *NumPy*. Version 1.19.2. Aug. 31, 2021. URL: https://numpy.org/about/.