Project 1

Notch filter

Introduction

In this course we motivate our study of theory by first considering various practical problems that we can apply that theory to. Our first project is to remove a sinusoidal “buzz” of some known frequency that has corrupted an audio signal.

The file `sounds.zip`, available on the course website, contains various audio files in “wav” and “mp3” formats. Two of those files are `jfk.wav` and `jfkBuzz.wav`. The first is a recording of President John F. Kennedy famously proclaiming, “Ask not what your country can do for you. Ask what you can do for your country.” This has a sample rate of 8 kHz and is just under eight seconds long. A segment is shown at left in Fig. 1. In the second file, `jfkBuzz.wav`, this signal is corrupted by the addition of a loud 700-Hz sinusoidal buzz.

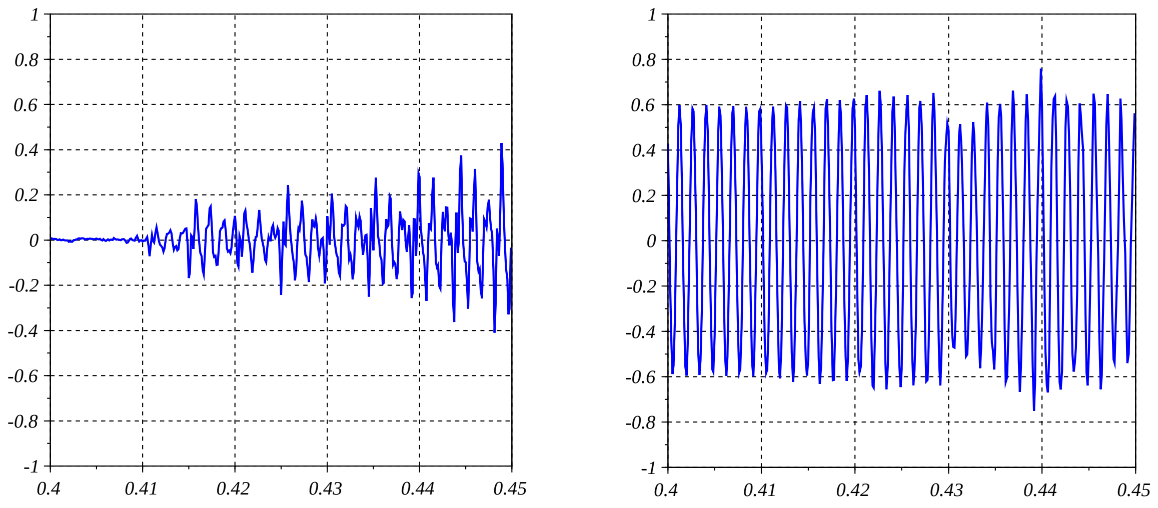

Fig. 1: (Left) voice signal segment. (Right) segment corrupted by 700-Hz sinusoidal buzz.

Our goal in this project is to design a *filter* to remove the 700-Hz buzz from the corrupted signal and “restore” the original voice signal. That is, our filter should take `jfkBuzz.wav` as input and output something very close to `jfk.wav`.

A filter that removes a single frequency component is often called a *notch filter,* hence the name of this project. Listen to both of these audio files. Brainstorm some ideas about how you might go about getting rid of that annoying 700-Hz buzz from the `jfkBuzz.wav` file. In particular, consider what tools and skills we will require.

What will we need?

To be able to "repair" the `jfkBuzz.wav` file we will probably require at least the following capabilities:

1. A software environment in which to implement our solution(s). In this course we use Scilab, some bare basics of which we cover below.
2. The ability to read, write, and listen to wav files. (We cover this below.)
3. The ability to manipulate signals in order to implement filters. (We cover this below and in various lectures.)
4. An understanding of the relation between an analog signal and its sampled, digital version. (We cover this in future lectures.)
5. Some basic theory to guide us in designing a filter. (We cover this in future lectures.)

Scilab

Scilab is free and open source software that provides an interactive environment for numerical computation. It is available for Windows, Mac and Linux at scilab.org. Scilab and the commercial package Matlab have very similar syntax. In this course we will use Scilab for computation.

Reading, writing, and listening to wav files

In Scilab we can read a wav file using the following command

```
[x,Fs,Nbits] = wavread('jfk.wav');
```

This returns the wav file samples as the array `x`. `Fs` is the sampling frequency (in Hz, samples per second) and `Nbits` is the number of bits per sample (typically 16). Both `Fs` and `Nbits` are returned by the function call to `wavread`.

To hear the sound in Scilab we use the command

```
sound(x,Fs,Nbits);
```

Suppose we apply a filter to `x` and produce a new signal `y`. We can write this to a new wav file using the command

```
wavwrite(y,Fs,Nbits,'new.wav');
```

Manipulating signals

Once we read a wav file into an array we can perform mathematical operations on the elements of that array to produce a "filtered" signal. Consider the following

```
--> length(x)
 ans  =   62379.

--> y = zeros(x);

--> for n=2:length(x)-1
  >   y(n)  = (x(n-1)+x(n)+x(n+1))/3;
  > end
```

This creates a new signal `y` of the same length as `x` and initialized to all zeros. At each time step (except the first and last) it sets the current element of `y` equal to the average of the current, preceding and following elements of `x`.

Exercise 1: Read `jfk.wav` into an array `x`. Print the sample frequency and length of the signal using the command `disp([Fs,length(x)]);`. Create an array of zeros, named `y`, of the same length as `x`. Set each element of `y` to be 0.5 times the corresponding element of `x`. Write `y` to a file named `y.wav`

A trial and error solution

Our task is to remove a sinusoidal buzz added to a speech signal. Suppose $y(n)$ is the “buzzless” speech signal. Our wav file contains the signal $x(n)$ which is formed as

$$x(n)=y(n)+s(n) \tag{1}$$

where $s(n)$ is the sinusoidal buzz. A discrete sinusoid has the form

$$s(n)=A\cos(2\pi(F_b/F_s)n+\theta) \tag{2}$$

where A is the *amplitude*, F_b and F_s are the buzz and sampling frequencies (in Hz), and θ is the *phase* (in radians).

The obvious solution is to simply subtract the sinusoidal buzz

$$y(n)=x(n)-s(n) \tag{3}$$

leaving the original speech signal. Commonly we know the buzz frequency, but we do not know the A,θ values. Let’s try to determine A,θ by trial and error using the following Scilab code.

```
A = 0.5; //amplitude
theta = 0; //phase in degrees
[x,Fs,Nbits] = wavread('jfkBuzz.wav');
n = [0:length(x)-1];
f = 700/Fs;
y = x-A*cos(2*%pi*f*n+theta*%pi/180);
sound(y,Fs,Nbits);
```

You can put this into a program file, say `debuzz1.sce`, and run it repeatedly while changing the amplitude and phase values until the buzz disappears. One approach is to start by varying the phase in 30-degree steps until you find the value that gives the least buzzing. Then try 10-degree steps. Then 1-degree steps. Then vary the amplitude in steps of 0.01. You may need to repeat the process.

This approach does (eventually) work, but it has a serious shortcoming. We have to search for the amplitude and phase of the buzz in order to precisely subtract it. Possibly we could come up with some way to have a computer (or DSP chip) automate that process. But, even then, it would depend on the buzz being *precisely* a sinusoid with uniform amplitude, phase, and frequency.

Once you’ve found the amplitude and phase that removes the buzz from `jfkBuzz.wav`, try using this approach on the files `jfkBuzz2.wav` and `jfkBuzz3.wav`. In these cases the amplitude or phase of the sinusoid slowly varies. You can set the amplitude and phase to cancel the buzz at one particular time, but not for the entire audio segment.

Instead of trying to precisely subtract out the buzz, a simpler and more robust approach is to design a filter that cuts out any signal at a frequency of 700 Hz. Of course this will also destroy the 700-Hz component of the speech signal, so we need to consider how the recovered audio signal might differ from the original.

FIR notch filter

FIR stands for *Finite Impulse Response*. We'll study these in detail in later lectures. For now let's define an FIR filter as a system in which the output is some linear combination of a given number of input samples, such as

$$y(n)=b_0x(n)+b_1x(n-1)+\cdots+b_Mx(n-M) \tag{4}$$

A very simple FIR filter is

$$y(n)=b_0x(n) \tag{5}$$

Each output sample is b_0 times the corresponding input sample. This is an *amplifier* with gain b_0 . The output is louder than the input if $b_0>1$ and quieter if $b_0<1$.

Amplifying simply makes both the signal and the buzz both louder or quieter. Clearly this is not going to get rid of the buzz. We need a filter that selectively suppresses the buzz while more-or-less leaving the signal intact. Let's try having the output be a linear combination of two samples, the current and previous input samples

$$y(n)=b_0x(n)+b_1x(n-1) \tag{6}$$

If the input is sinusoidal with frequency $\omega_b=2\pi f_b=2\pi(F_b/F_s)$, then we want the output to be zero. As we will see later, there are advantages to representing a sinusoid by a complex exponential using

$$e^{j\omega n}=\cos(\omega n)+j\sin(\omega n) \tag{7}$$

Let's set

$$x(n)=e^{j\omega_b n} \tag{8}$$

We want

$$y(n)=0=b_0e^{j\omega_b n}+b_1e^{j\omega_b(n-1)} \tag{9}$$

Solving for b_1 in terms of b_0 we find

$$b_1=-b_0e^{j\omega_b} \tag{10}$$

This would require a filter with complex coefficients. Usually we want our filters to have real coefficients. So, let's try a linear combination of three input samples

$$y(n)=b_0x(n)+b_1x(n-1)+b_2x(n-2) \tag{11}$$

Now we have

$$0=b_0e^{j\omega_b n}+b_1e^{j\omega_b(n-1)}+b_2e^{j\omega_b(n-2)} \tag{12}$$

Multiply through by $e^{-j\omega_b(n-1)}$ we get

$$0=b_0e^{j\omega_b}+b_1+b_2e^{-j\omega_b} \tag{13}$$

If we arbitrarily set $b_0=b_2=1$ this reduces to

$$0=2\cos(\omega_b)+b_1 \tag{14}$$

from which

$$b_1=-2\cos(\omega_n) \tag{15}$$

All three coefficients are real, and our FIR filter is described by

$$y(n)=x(n)-2\cos(\omega_b)x(n-1)+x(n-2) \tag{16}$$

Exercise 2: Verify that $x(n)=A\cos(\omega_b n+\theta)$ in (16) produces $y(n)=0$.

The follow Scilab commands apply our FIR filter to the file `jfkBuzz.wav`.

```
[x,Fs,Nbits] = wavread('jfkBuzz.wav');
b = [1,-2*cos(2*%pi*700/Fs),1];
y = filter(b,1,x);
sound(y,Fs,Nbits);
```

Run this code and listen to the result. Also apply the filter to `jfkBuzz2.wav` and `jfkBuzz3.wav`. The buzz is indeed gone in all cases.

Unfortunately, the audio does not sound like the original `jfk.wav` file. Instead it sounds like the "treble" has been turned way up and the "bass" way down. We have gotten rid of the 700-Hz component of the signal, but we've altered other frequency components. To quantify the effect of the filter on different frequency components of the signal let's input

$$x(n)=e^{j\omega n} \tag{17}$$

with ω arbitrary. The output is

$$\begin{aligned} y(n)&=e^{j\omega n}-2\cos(\omega_b)e^{j\omega(n-1)}+e^{j\omega(n-2)} \\ &=\left[1-2\cos(\omega_b)e^{-j\omega}+e^{-j\omega 2}\right]e^{j\omega n} \end{aligned} \tag{18}$$

The output signal is the input signal $e^{j\omega n}$ multiplied by the *frequency response*

$$H(e^{j\omega})=1-2\cos(\omega_b)e^{-j\omega}+e^{-j2\omega}=2e^{-j\omega}\left[\cos(\omega)-\cos(\omega_b)\right] \tag{19}$$

Since $H(e^{j\omega_b})=0$ we get rid of the buzz. For all $\omega\neq\omega_b$ we would ideally like to have $H(e^{j\omega})=1$ so that "non-nulled" frequencies would be unaffected by the filter. However, as shown in Fig. 2 the frequency response does not have this property. Instead, it amplifies higher frequencies and attenuates lower frequencies.

Exercise 3: Verify (19).

We could try an FIR filter with more coefficients

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \tag{20}$$

and attempt to specify the coefficients so the frequency response is zero at 700 Hz and unity at all other frequencies. We'll come back to this idea in a later lecture. For now, let's preview another type of filter.

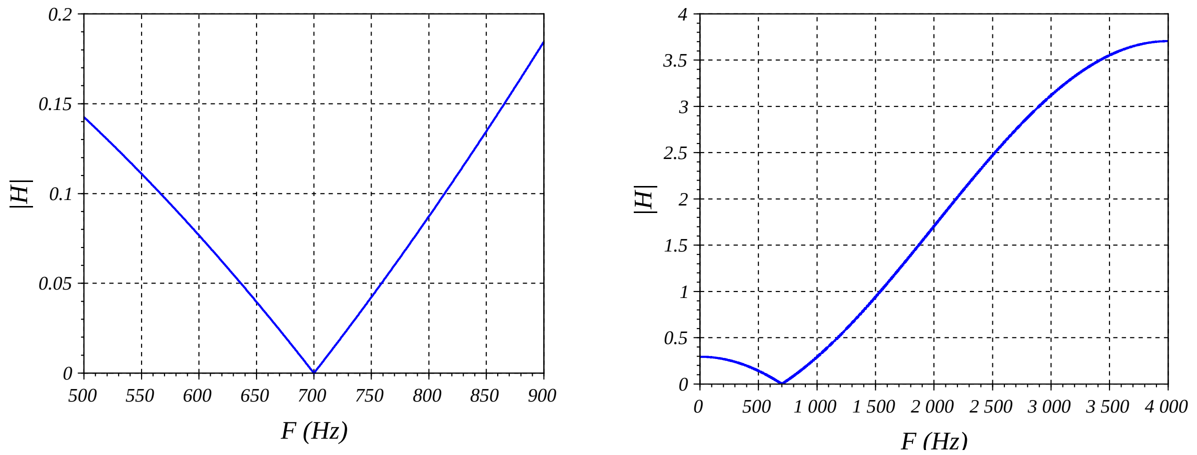

Fig. 2: Frequency response of simple FIR notch filter.

IIR notch filter

IIR stands for *Infinite Impulse Response*. We'll also study these in detail later. For now let's define an IIR filter as a system in which a linear combination of some number of output samples equals a linear combination of some number of input samples. Specifically, consider

$$y(n)+a_1 y(n-1)+a_2 y(n-2)=b_0 x(n)+b_1 x(n-1)+b_2 x(n-2) \tag{21}$$

Solving for $y(n)$

$$y(n)=b_0 x(n)+b_1 x(n-1)+b_2 x(n-2)-a_1 y(n-1)-a_2 y(n-2) \tag{22}$$

The current output is a linear combination of the current input, the two previous input samples, and the two previous output samples.

To compute the frequency response we set $x(n)=e^{j\omega n}$ and assume the output is $y(n)=H e^{j\omega n}$ where $H=H(e^{j\omega})$ is the frequency response. We have

$$\begin{aligned} H e^{j\omega n} &= -a_1 H e^{j\omega(n-1)} - a_2 H e^{j\omega(n-2)} \\ &\quad + b_0 e^{j\omega n} + b_1 e^{j\omega(n-1)} + b_2 e^{j\omega(n-2)} \end{aligned} \tag{23}$$

Moving all terms with an H factor to the left side of the equation

$$H\left[e^{j\omega n}+a_1 e^{j\omega(n-1)}+a_2 e^{j\omega(n-2)}\right]=b_0 e^{j\omega n}+b_1 e^{j\omega(n-1)}+b_2 e^{j\omega(n-2)} \tag{24}$$

and solving for the frequency response we find

$$H(e^{j\omega})=\frac{b_0+b_1 e^{j\omega}+b_2 e^{-j2\omega}}{1+a_1 e^{-j\omega}+a_2 e^{-j2\omega}} \tag{25}$$

The numerator has the same form as our FIR filter. Let's again take $b_0=b_2=1$ and $b_1=-2\cos(\omega_b)$ so the numerator is zero at ω_b. For other frequencies, if the denominator is approximately equal to the numerator then $H(e^{j\omega})\approx 1$. However, we need to make sure that the denominator is not 0 at ω_b since then $H(e^{j\omega_b})=0/0$ is undefined. We can achieve this by taking

$$H(e^{j\omega}) = \frac{1 - 2\cos(\omega_b)e^{-j\omega} + e^{-j2\omega}}{1 - 2r\cos(\omega_b)e^{-j\omega} + r^2 e^{-j2\omega}} \tag{26}$$

where $r \neq 1$. For $r = 0$ (26) reduces to the FIR result (19). The closer r is to 1, the closer $H(e^{j\omega})$ is to 1 for $\omega \neq \omega_b$. The resulting filter is described by

$$y(n) = x(n) - 2\cos(\omega_b)x(n-1) + x(n-2) + 2r\cos(\omega_b)y(n-1) - r^2 y(n-2) \tag{27}$$

A Scilab implementation with $r = 0.99$ is

```
[x,Fs,Nbits] = wavread('jfkBuzz.wav');
omegab = 2*%pi*700/Fs;
r = 0.99;
b = [1, -2*cos(omegab), 1];
a = [1, -2*r*cos(omegab), r^2];
y = filter(b,a,x);
sound(y,Fs,Nbits);
```

Run this code and listen to the result. Apply the filter to `jfkBuzz.wav`, `jfkBuzz2.wav`, and `jfkBuzz3.wav`. The buzz is indeed gone in all cases. Moreover, the result sounds like the original `jfk.wav` file. As shown in Fig. 3 this is because the frequency response is nearly unity at all frequencies except the buzz frequency.

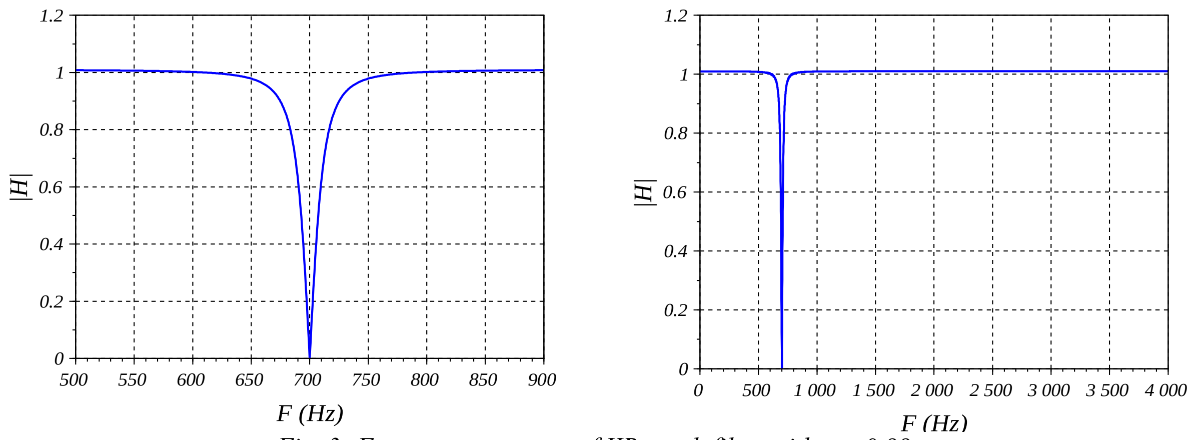

Fig. 3: Frequency response of IIR notch filter with $r = 0.99$.

IIR filters have a potential problem, however. Run the code again with the value of r changed to

```
r = 1.01;
```

and listen to the result (with the volume turned down). A very small change in the parameter r has created a *huge* change in the behavior of the filter. The filter has become *unstable*. This motivates us in a future lecture to study the conditions under which a filter has the property of *stability*.

Bandstop filters

In the file `jfkBuzz4.wav` the buzz frequency slowly varies about 700 Hz. Apply the FIR and IIR notch filters to this signal and listen to the results. They both fail to remove all the buzzing.

This is to be expected since the buzz frequency varies over a *range* of frequencies while a notch filter only nulls a single frequency. What we need is a filter that removes frequency components over an entire range, or "band," of frequencies. This is called a *bandstop* filter.

Bandstop filters can be implemented in both FIR and IIR forms. An FIR implementation can be written

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \tag{28}$$

The challenge is to determine both the filter length M and the coefficients b_k. We will see how to do this in a future lecture. For each value of y the filter requires $M+1$ multiplications and M additions. If M is large this can be a significant computational load. The frequency response of an FIR bandstop filter with $M=200$ is shown in Fig. 4.

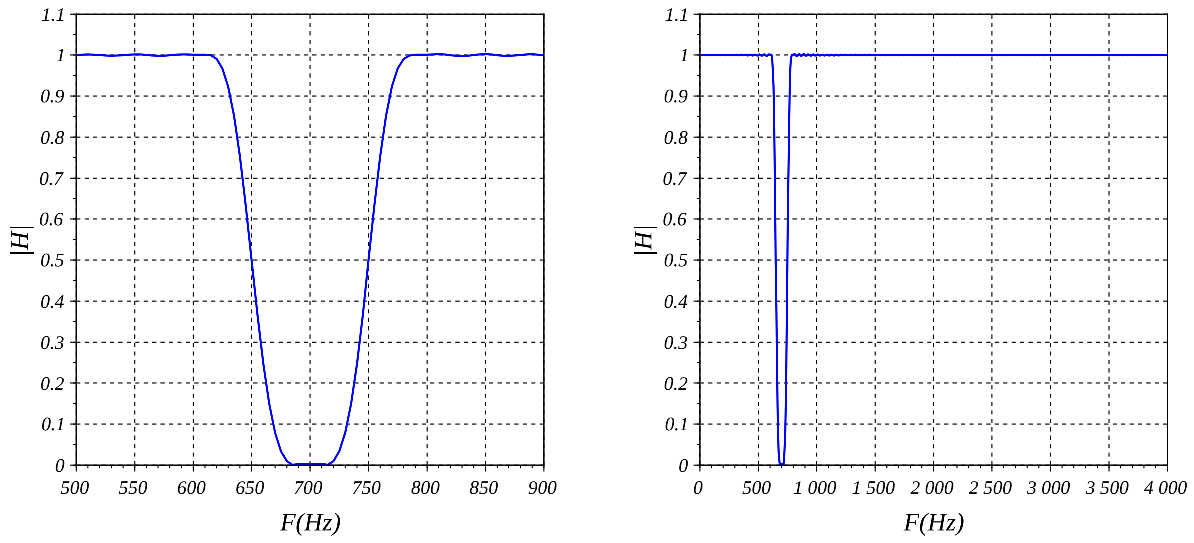

Fig. 4: FIR bandstop filter frequency response. Filter has 201 coefficients ($M=200$).

An IIR bandstop filter has the form

$$y(n)=\sum_{k=0}^{M} b_k x(n-k)-\sum_{k=1}^{N} a_k y(n-k) \tag{29}$$

where we need to determine the lengths M and N as well as the coefficients b_k and a_k. Again, we will study how to do this in a future lecture. This requires $N+M+1$ multiplications and $N+M$ additions for each value of y. As we will see, a bandstop IIR filter typically requires fewer arithmetic operations than the corresponding FIR filter with similar frequency response. Fig. 5 shows the response of an IIR bandstop filter with $M=N=8$.

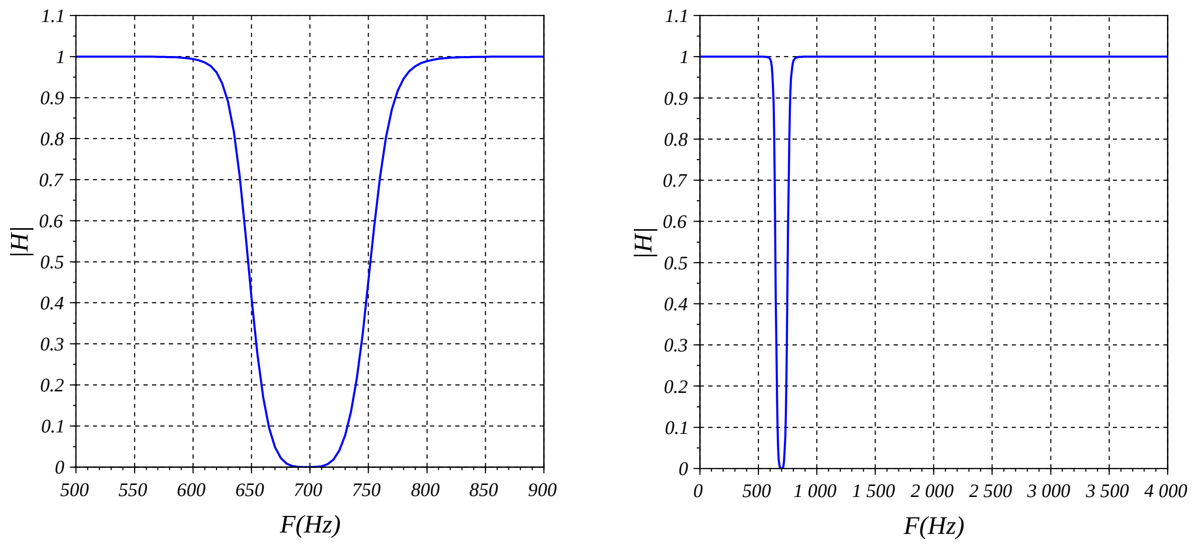

Fig. 5: IIR bandstop filter frequency response. Filter has 17 coefficients ($M=N=8$).