Notch Demo with Extras

Using Pluto with `PyCall` for interactive demos that leverage the `scikit-dsp-comm` Python code base.

Get Started with Package Imports

- The first import is a Julia code module I am developing to make Julia more user friendly for general purpose DSP
- The next imports are packages that are needed by my package and elsewhere in this notebook
- Finally, import `PyCall` so I can bring in `scikit-dsp-comm` packages that I am very familiar with

ingredients

```
alias_package_name = ingredients(path::String)
```

This function is used to import a user code module into the current workspace.

```
include("modules/ingredients_function.jl")
```

```
dsp_tools = Main.var"dsp_tools.jl"
```

```
dsp_tools = ingredients("modules/dsp_tools.jl")
```

```
import Plots, Polynomials, DSP, FFTW
```

```
using PlutoUI, PyCall, LaTeXStrings
```

```
ss =
PyObject <module 'sk_dsp_comm.sigsys' from '/Users/mwickert/.julia/conda/3/x86_64/lib/pyt
```

```
ss = PyCall.pyimport("sk_dsp_comm.sigsys")
```

```
dc =
PyObject <module 'sk_dsp_comm.digitalcom' from '/Users/mwickert/.julia/conda/3/x86_64/lib
```

```
dc = PyCall.pyimport("sk_dsp_comm.digitalcom")
```

```
fir_h =
PyObject <module 'sk_dsp_comm.fir_design_helper' from '/Users/mwickert/.julia/conda/3/x86
```

```
fir_h = PyCall.pyimport("sk_dsp_comm.fir_design_helper")
```

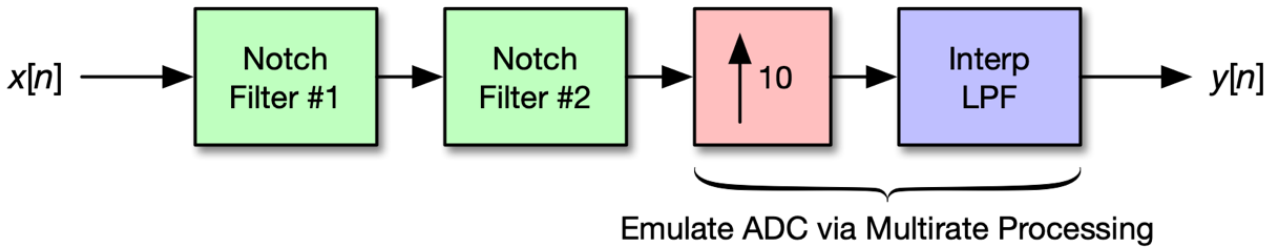
```
"support for image import"
```

```
begin
    struct Wow
        filename
    end

    function Base.show(io::IO, ::MIME"image/png", w::Wow)
        write(io, read(w.filename))
    end
    "support for image import"
end
```

The Upsampler and Interpolation Filter

Below we consider just the upsampler and interpolation lowpass filter stages that are used to make the output signal appear more like an analog waveform.



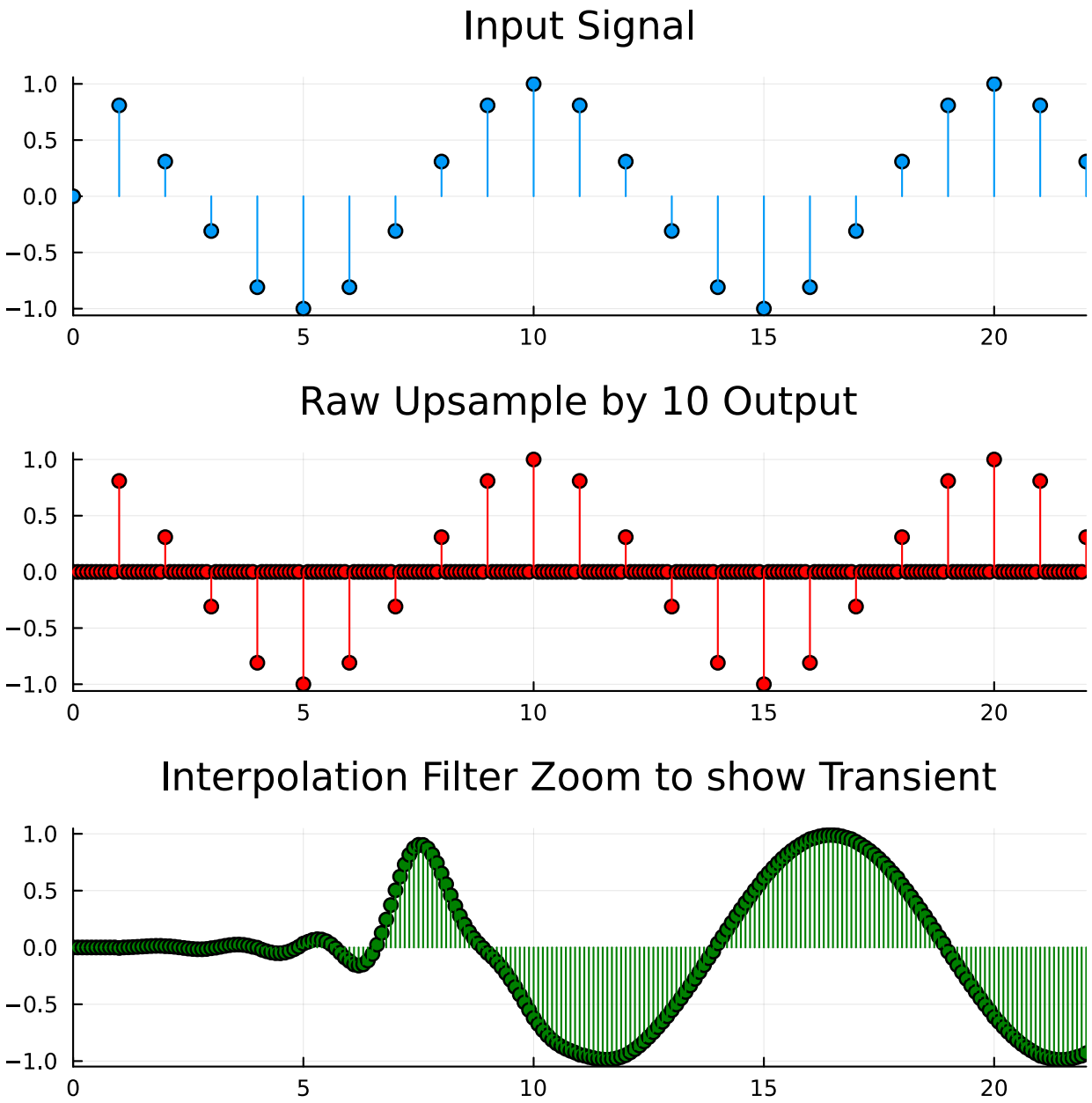
System Block Diagram to Explore IIR Notch Filter Properties

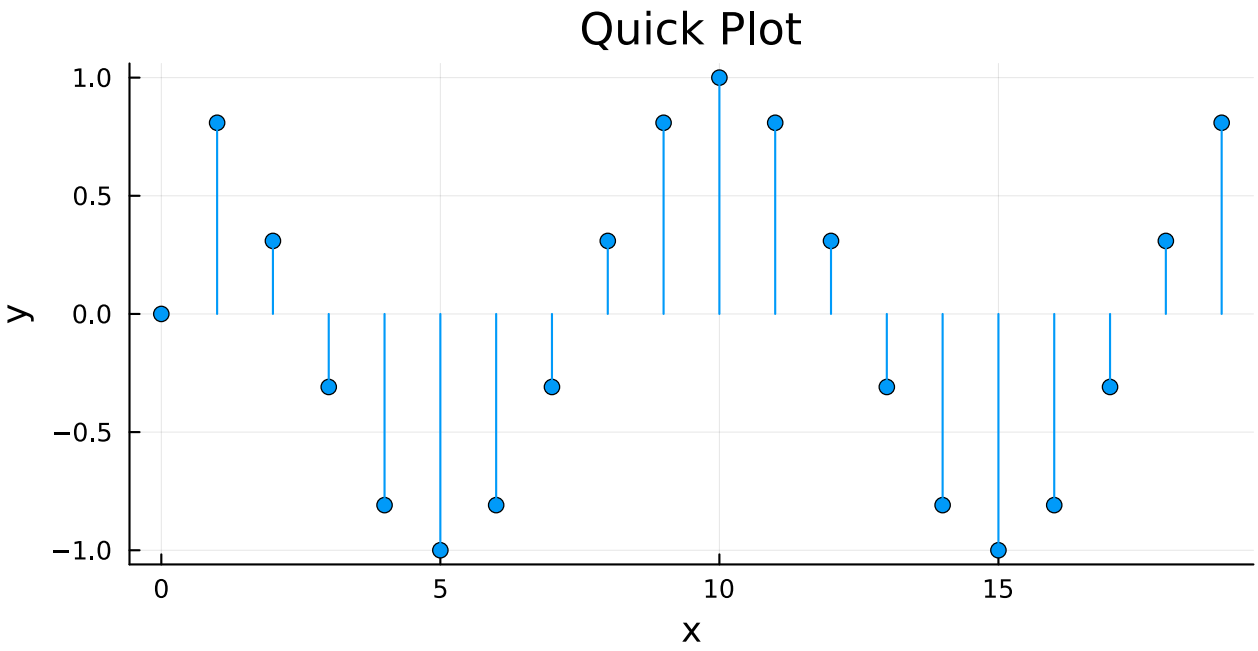
```
Wow("iir_notch_block.png")
```

Change the Sinusoid Frequency:

Input Sinusoid Frequency f_0 = 0.1 Hz,

Trim Start of Plot n_{start} = 0 Samples


Input Signal
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
Raw Upsample by 10 Output
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
Interpolation Filter Zoom to show Transient
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20




```
dsp_tools.stem(n_up[1:20],x_up[1:20],"Quick Plot","x","y",(600,300))
```

```
(100, 200)
```

```
(100,200)
```

Notch Filter

Consider a second-order notch filter of the form

$$H_i(z) = \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = \frac{1 - 2\cos(\omega_0) z^{-1} + z^{-2}}{1 - 2r\cos(\omega_0) z^{-1} + r^2 z^{-2}},\ i = 1, 2$$

where $0 \leq r < 1$ is the radius of the pole pair and $0 \leq \omega_0 \leq 0.5$ is the notch center frequency. **Note** to obtain the frequency response we let $z \rightarrow e^{-j\omega} = e^{j2\pi f/f_s}$, where f_s is the sampling rate.

Use `sk_dsp_comm.sigsys` for Notch Filter Coefficients

Below is a cascade of two second-order notch filters with the numerator and denominator coefficients retuyrned as arrays from a call to the `sk_dsp_comm.sigsys` function `fir_iir_notch(fc, fs, r_pole)`. So in particular `b1 = [1,b1,b2]` and `a1 = [1,a1,a2]`, etc. The cascade of two filters is the product of the system functions, e.g. $H(z) = H_1(z)H_2(z)$. The polynomial multiplication of the numerator denominator coefficient sets is managed by `cascade_filter()`.

5

```
begin
    # b = fir_h.fir_remez_lpf(.1,.18,.1,60,1.0)
    b1, a1 = ss.fir_iir_notch(fn1,1.0,rp)
    b2, a2 = ss.fir_iir_notch(fn2,1.0,rp)
    b, a = ss.cascade_filters(b1,a1,b2,a2)
    length(b)
end
```

Find the Frequency Response of the Notch Filter

5

```
begin
    f = collect(0:.0001:.5)
    H = dsp_tools.freqz(b,a,f)
    length(b)
end
```

Define the Test Waveform and an Interpolation LPF

130

```
begin
    nn = collect(0:100)
    s_in = 0cos.(2*pi*f0/5*nn) .+ cos.(2*pi*f0*nn)
    s_out = dsp_tools.lccde(b,a,s_in)
    b_up = fir_h.fir_remez_lpf(.05,.07,.1,60,1.0) #ones(10)/10
    length(b_up)
end
```

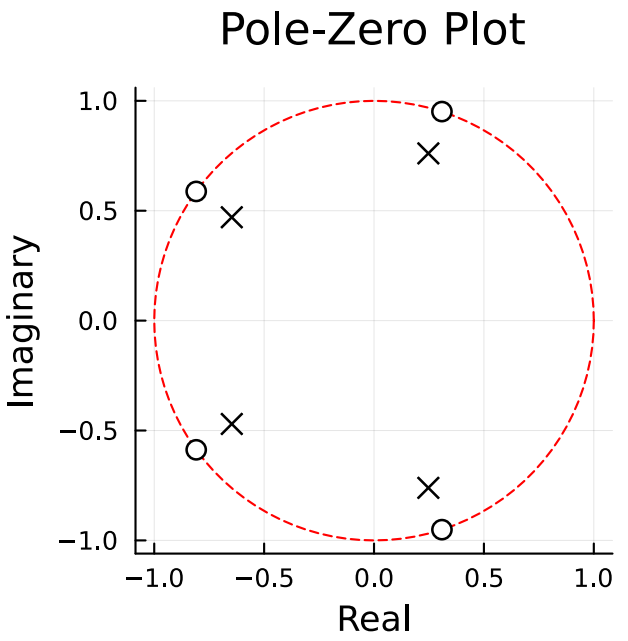
Change Parameters in the Model:

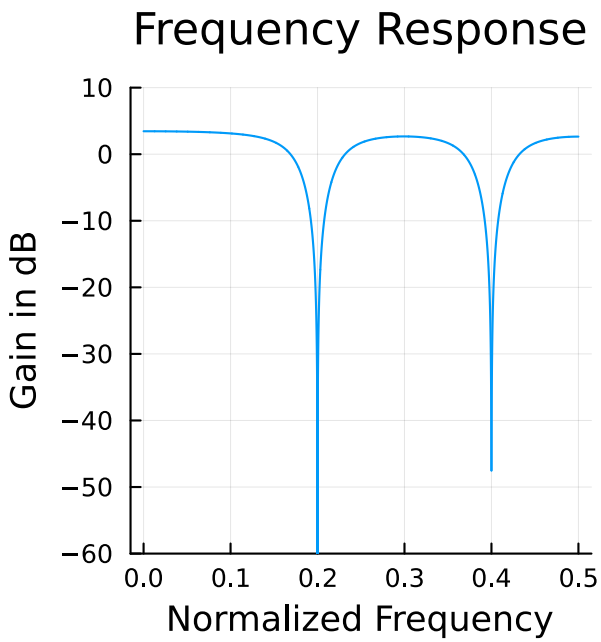
Input Sinusoid Frequency f_0 = 0.1 Hz,

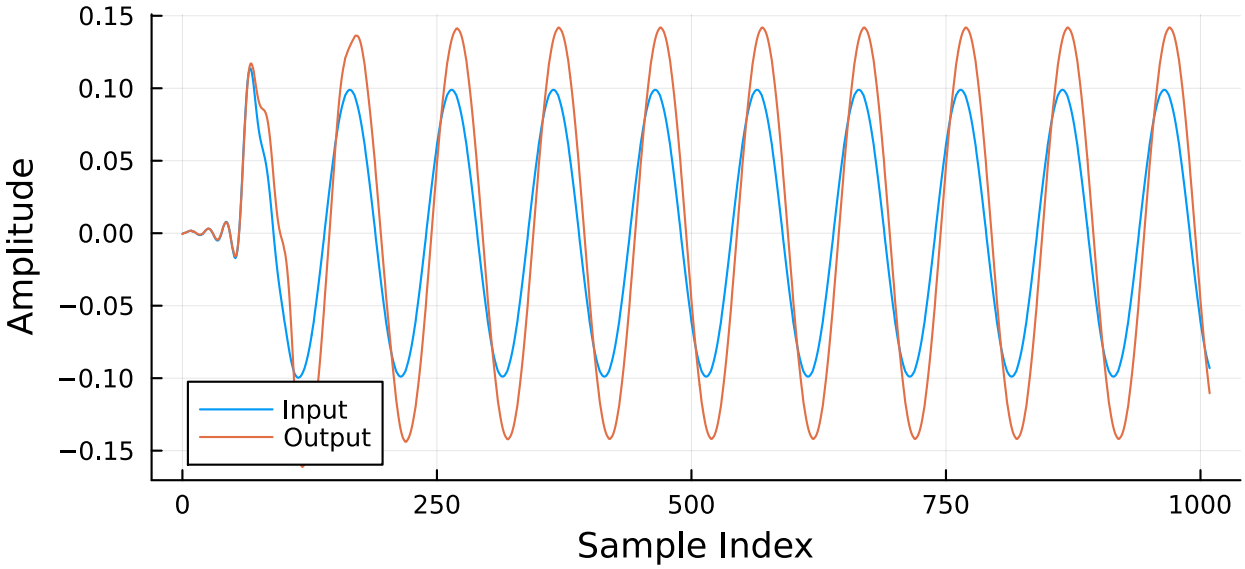
Notch Frequency f_{n1} = 0.2 Hz,

Notch Frequency f_{n2} = 0.4 Hz,

Notch Filter Pole radius r = 0.8


Pole-Zero Plot
Imaginary
Real
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


Frequency Response
Gain in dB
Normalized Frequency
10
0
−10
−20
−30
−40
−50
−60
0.0
0.1
0.2
0.3
0.4
0.5


Amplitude
Sample Index
Input
Output
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
250
500
750
1000

```
begin
    L = Plots.@layout [ plot_A plot_B ; plot_C ]
    plot_A = dsp_tools.zplane(b,a)
    plot_B = Plots.plot(f,20log10.(abs.(H)),leg=false,ylimit=(-60,10),
        xlabel="Normalized Frequency",ylabel="Gain in dB",title="Frequency
        Response")
    nn_up = collect(0:10*length(nn)-1)
    # DSP code for overlaying upsampled waveforms to be 'analog-like'
    # Here we upsample by 10 so waveforms become continuous-like
    s_in_up = dsp_tools.lccde(b_up,[1],dsp_tools.upsample(s_in,10))
    s_out_up = dsp_tools.lccde(b_up,[1],dsp_tools.upsample(s_out,10))
    plot_C = Plots.plot(nn_up,s_in_up,label="Input")
    plot_C = Plots.plot!(nn_up,s_out_up,label="Output")
    plot_C = Plots.plot!(xlabel="Sample Index",ylabel="Amplitude")
    Plots.plot(plot_A, plot_B, plot_C, layout = L, size=(600,600))
end
```

Animate a Mixture of Julia and Python Code

Here we consider gating a damped sinusoid with a rectangle pulse:

$$y_p(t) = \cos(\omega_p t)e^{-t/5} \cdot \Pi\left(\frac{t-5}{\tau}\right), \ -5 < t < 20$$

for $\omega_p = 2\pi/4$ and τ controlled by the slider over the range $\tau \in [1, 20]$.

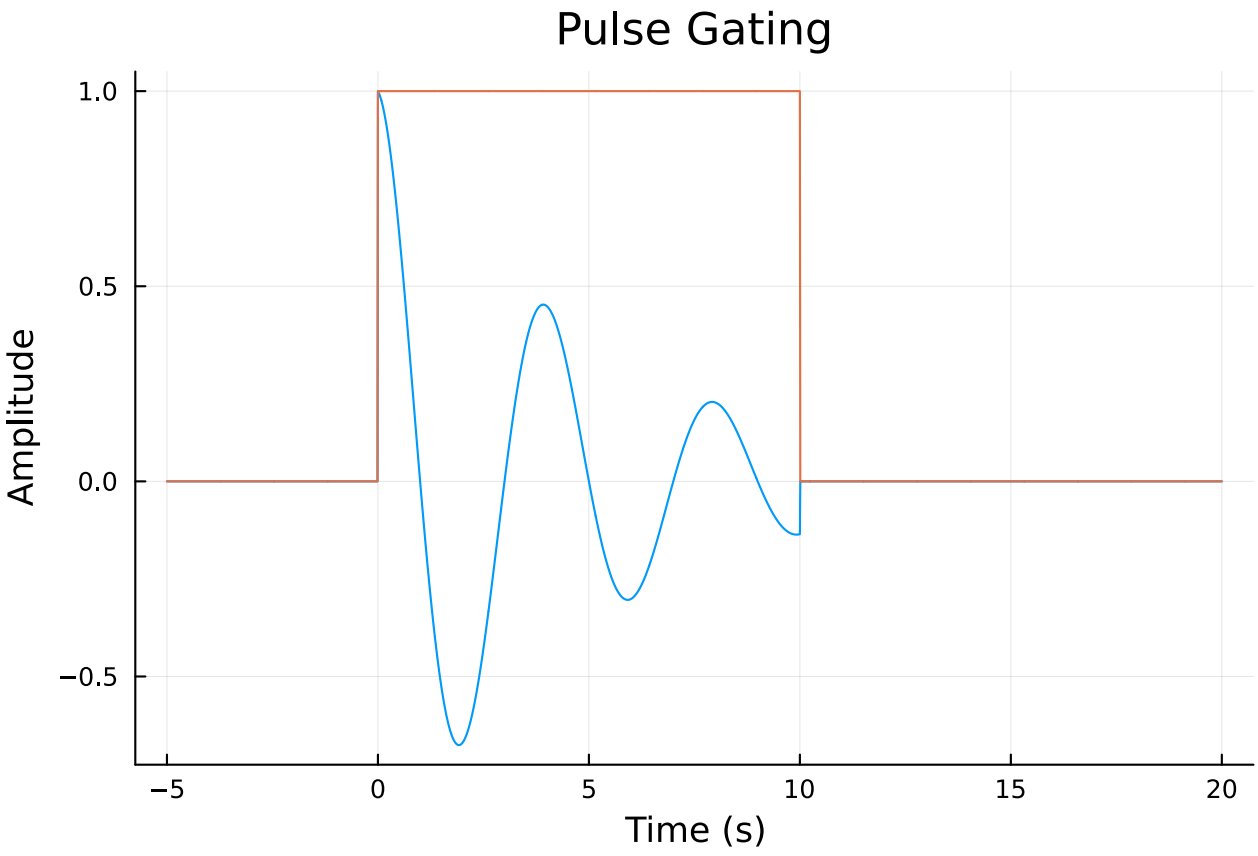
Define a Pulse gated Damped Sinusoid

2501

```
begin
    ω_p = 2π/4
    t_p = collect(-5:.01:20)
    x_p = cos.(ω_p * t_p)
    y_p = x_p .* ss.rect(t_p.-5,τ_p) .* exp.(-t_p/5)
    # y_p = x_p
    length(y_p)
end
```

Change Pulse Width

Set τ = 10.0



[0.000610392, -6.97396e-5, -0.00115891, -0.00271317, -0.0038523, -0.00347582, -0.0010058

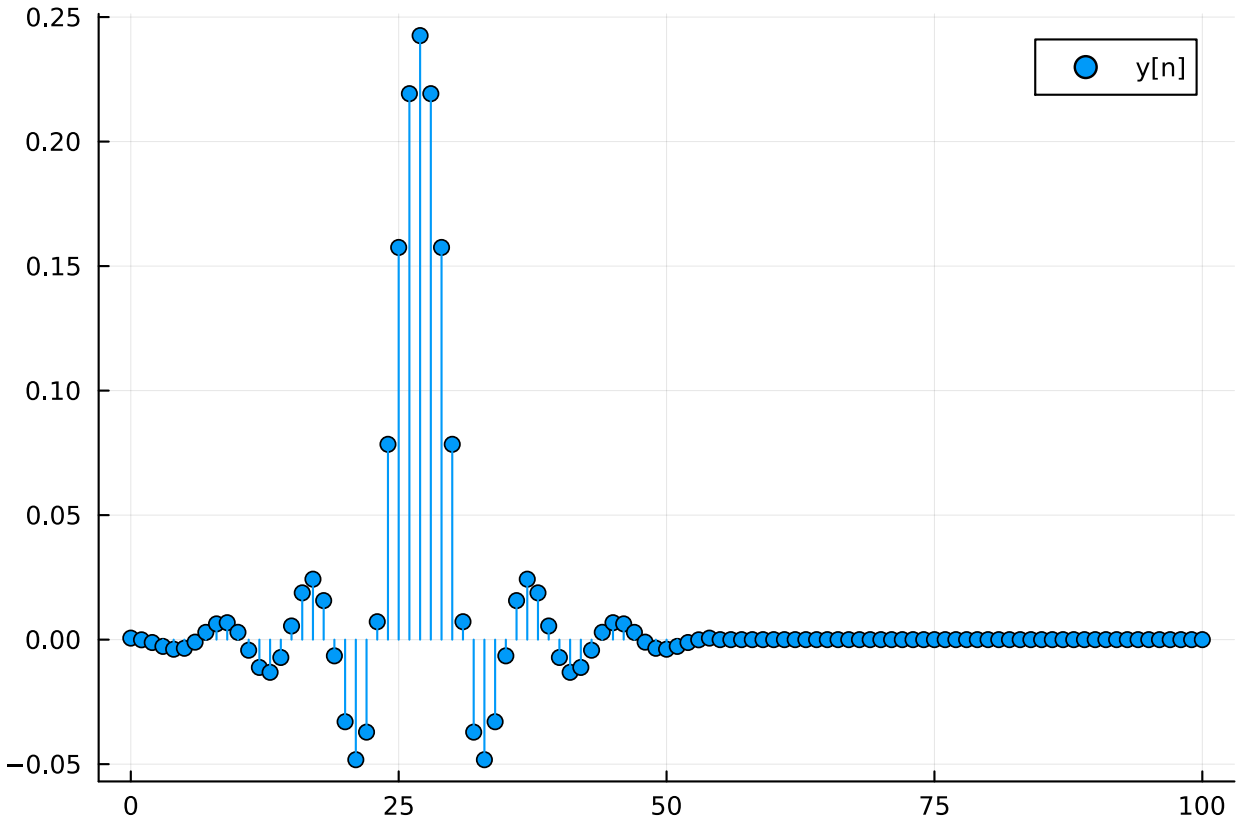
```
begin
    # df = DSP.DF2TFilter(DSP.Filters.PolynomialRatio(ones(5),[1,-.8]))
    b_df = fir_h.fir_remez_lpf(0.1,0.15,0.1,60,1.0)
    df = DSP.FIRFilter(b_df)
    x1 = vcat([1],zeros(50))
    y1 = DSP.filt(df,x1)
    y2 = DSP.filt(df,zeros(50))
    y = vcat(y1,y2)
end
```

44.099999999999994

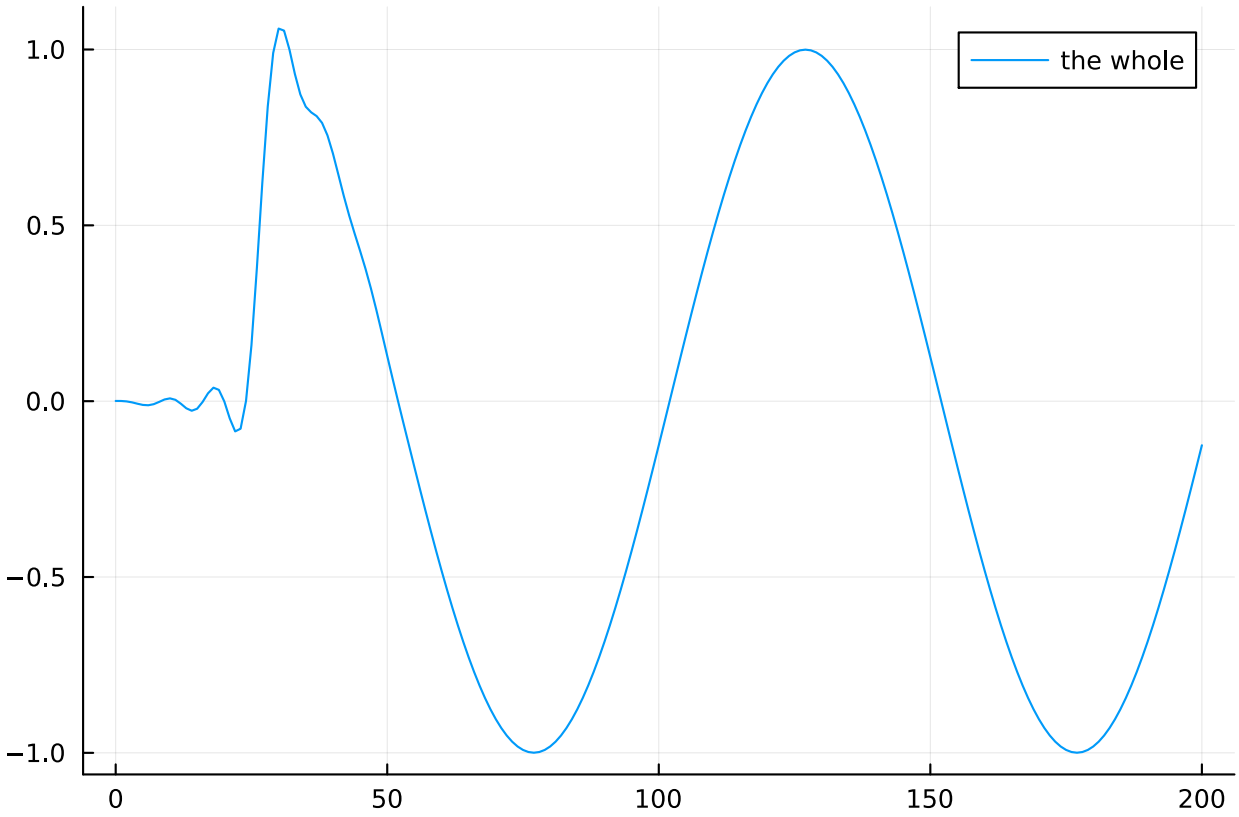
```
147/160*48
```

FIRFilter(FIRStandard([0.000610392, -6.97396e-5, -0.00115891, -0.00271317, -0.0038523,

```
df
```



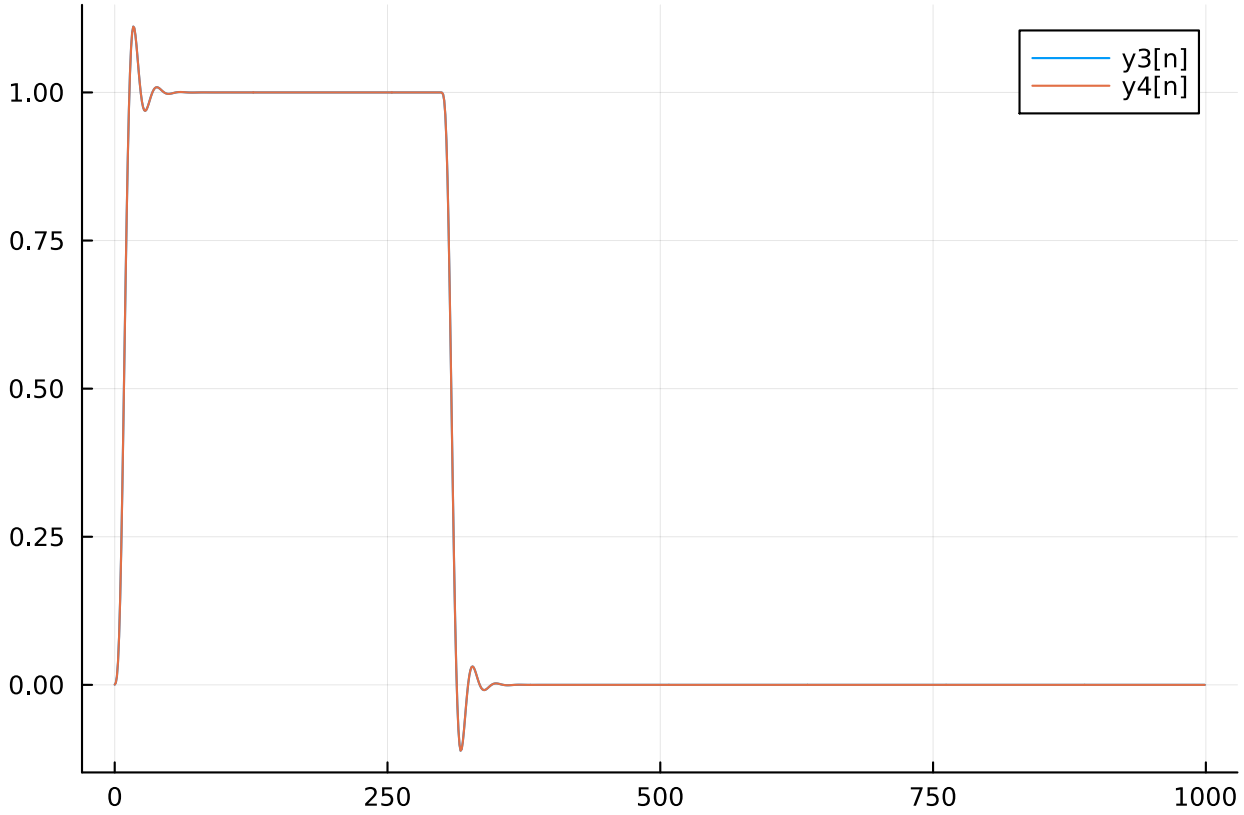

```
begin
    Plots.plot(Vector(0:length(y)-1),y,line=:stem, marker=:circle,label="y[n]")
end
```



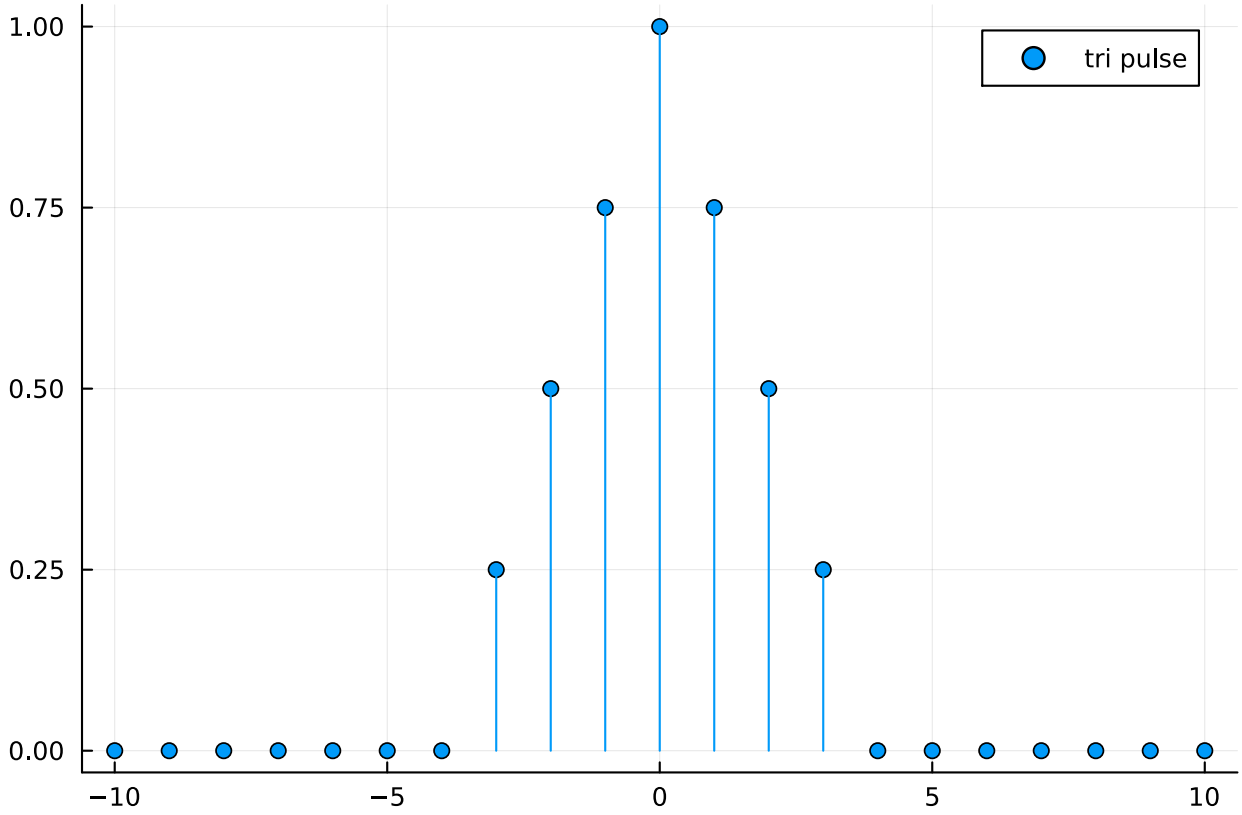

```
begin
    df
    n_df = collect(0:200)
    xa = cos.(2*pi*0.01*n_df)
    ya = DSP.filt(df,xa[1:100])
    yb = DSP.filt(df,xa[101:201])
    yab = vcat(ya,yb)
    Plots.plot(n_df,yab,label="the whole")
end
```

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0

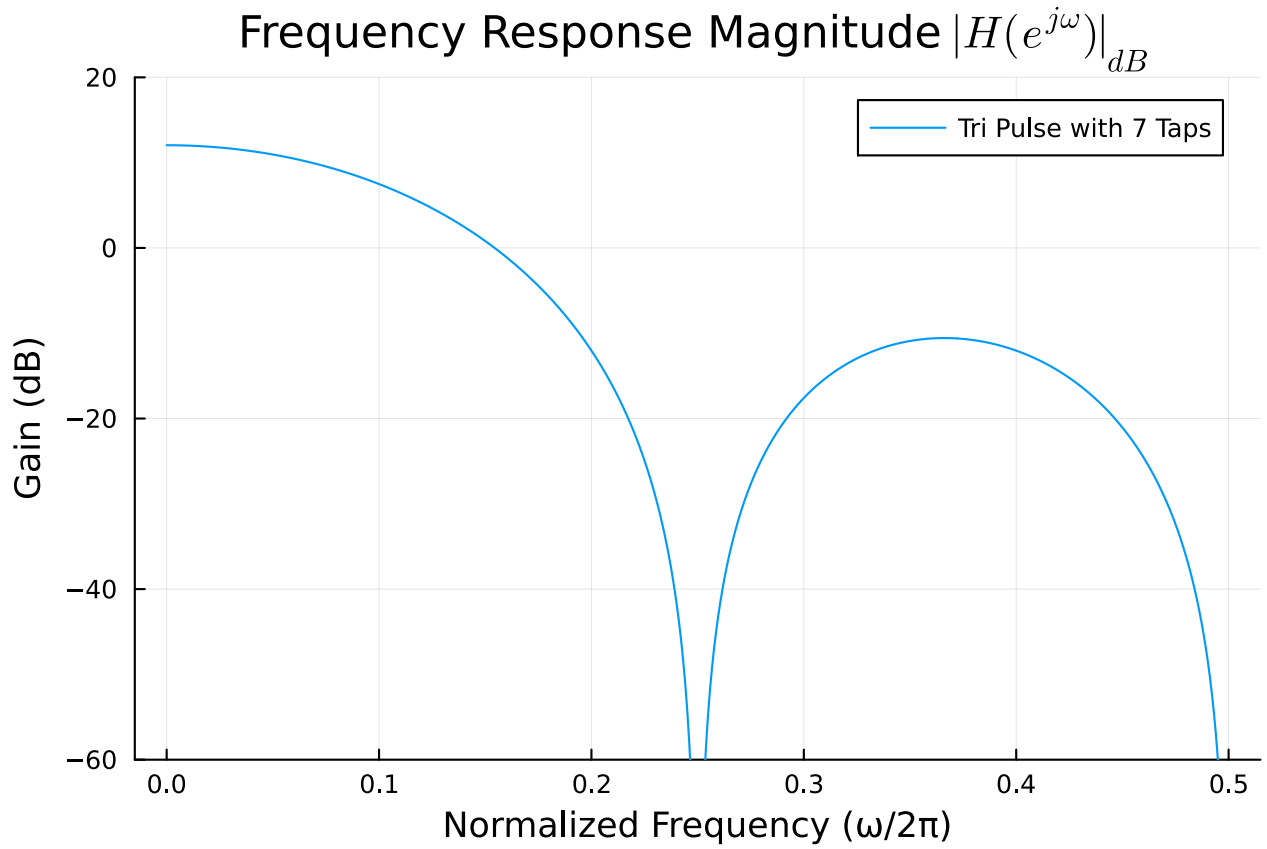
```
df.history = zeros(df.historyLen)
```




```
begin
    df2 = DSP.digitalfilter(DSP.Lowpass(0.1), DSP.Butterworth(4))
    f3 = DSP.DF2TFilter(df2)
    x = vcat(ones(300),zeros(700))#randn(1000)
    # apply the filter all in one go
    y3 = DSP.filt(df2, x)
    # apply the filter in chunks
    y4 = reduce(vcat, [DSP.filt(f3, x[(1:250).+os]) for os in [0,250,500,750]])
    # @assert y3 ≈ y4
    Plots.plot(Vector(0:length(y3)-1),y3,label="y3[n]")
    Plots.plot!(Vector(0:length(y4)-1),y4,label="y4[n]")
end
```

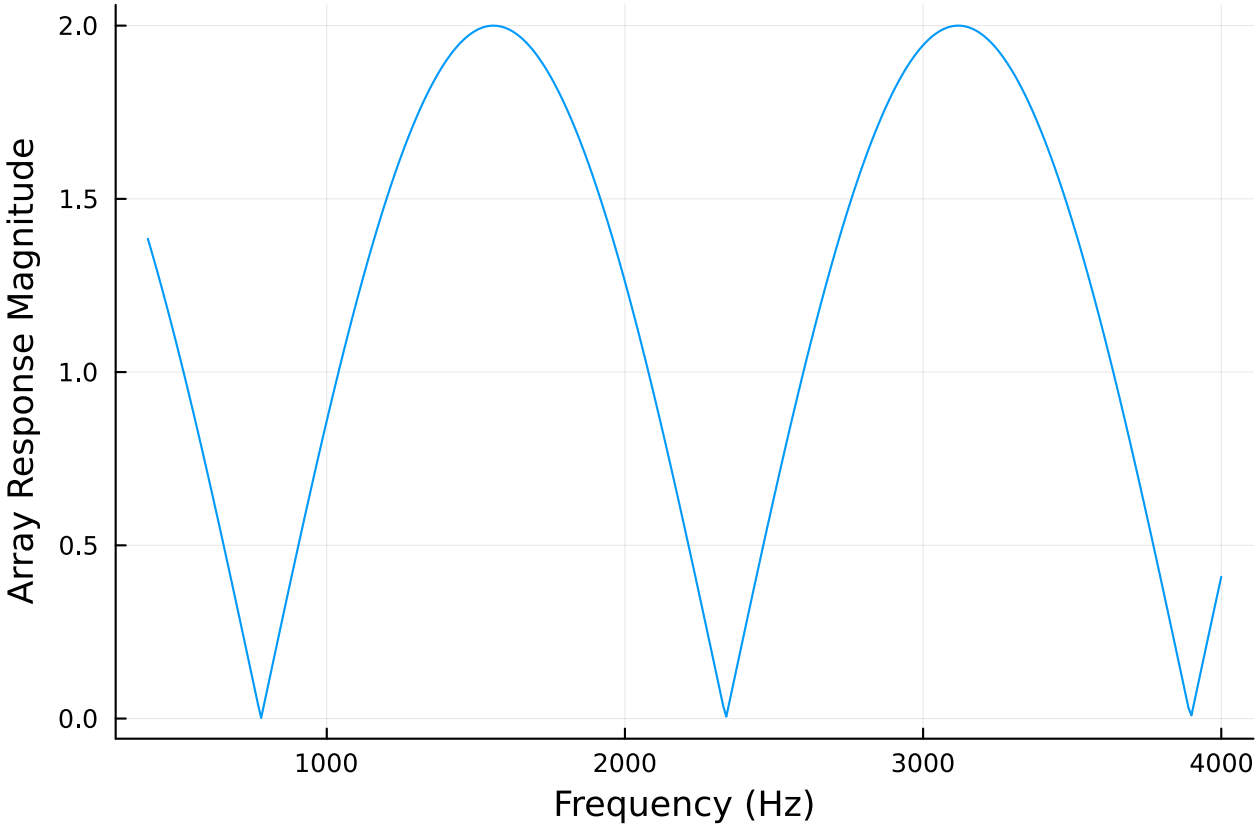



```
begin
    n_tri = collect(-10:10)
    tri_pulse = ss.tri(n_tri,4)
    # b_imp = fir_h.fir_remez_lpf(0.2,0.3,0.1,60,1)
    # Plots.plot(t_tri,tri_pulse,label="Tri pulse")
    Plots.plot(n_tri,tri_pulse,line=:stem, marker=:circle,label="tri pulse")
end
```




```
begin
    f_df = collect(0:1/1024:0.5)
    L_df = length(tri_pulse)
    H_tri = dsp_tools.freqz(tri_pulse,[1],f_df)
    Plots.plot(f_df,20*log10.(abs.(H_tri)),label="Tri Pulse with 7 Taps",ylim=
    (-60,20))
    Plots.plot!(xlabel="Normalized Frequency (ω/2π)",ylabel="Gain (dB)")
    Plots.plot!(title="Frequency Response Magnitude " * L"|H(e^{j\omega})|_{dB}" )
end
```

Two-Element Microphone Array




```
begin
    c_s = 343.0
    d_space = 22.0/100
    θ_arrive = π*0.0/180
    f_null = collect(400:10:4000)
    τ_d = d_space/c_s*cos(θ_arrive)
    R_mag = abs.(1 .+ exp.(1im*2π*f_null*τ_d))
    Plots.plot(f_null,R_mag,leg=false)
    Plots.plot!(xlabel="Frequency (Hz)",ylabel="Array Response Magnitude")
end
```

Mic1/Mic2 Range Difference with Source at r from Center

$$D_{r2r1} = \frac{1}{2}\left(\sqrt{d^2 + 4dr\cos(\theta)\cos(\phi) + 4r^2} - \sqrt{d^2 - 4dr\cos(\theta)\cos(\phi) + 4r^2}\right)$$

When $r \gg d$ the rays arrive as a plane wave parallel to each other:

$$D_{r2r1} = d\cos(\theta)\cos(\phi)$$

```
md"#### Mic1/Mic2 Range Difference with Source at $r$ from Center

$$D_{r2r1} = \frac{1}{2} \left(\sqrt{d^2+4 d r \cos (\theta ) \cos (\phi )+4 r^2}-\sqrt{d^2-4 d r \cos (\theta ) \cos (\phi )+4 r^2}\right)$$
When $r \gg d$ the rays arrive as a plane wave parallel to each other:

$$D_{r2r1} = d \cos (\theta ) \cos (\phi )$$
"
```