Sample Rate Reduction for Hydrophone Data

Paul McGill
July 30, 2020

Calculating decimation factors

Our goal is to decimate the 256 kHz hydrophone data down to 2 kHz, a factor of 128. This will require first low-pass filtering the signal to limit its bandwidth, followed by downsampling (i.e. throwing out 127 out of every 128 samples). Richard Lyons describes the process in [1], section 10.1.

Instead of decimating by a factor D of 128 in one stage, it's more efficient to use two stages, D1 and D2, resulting in much shorter filters. The optimum factor for the first stage D1 is defined in Lyons as:

In[1]:=
$$\text{dopt}[D\_, F\_] := 2\,D\,\frac{1 - \sqrt{D\,F\,/\,(2 - F)}}{2 - F\,(D + 1)}$$

where D = D1 x D2 is the total decimation factor, and F = Δf/fstop is the ratio of the final filter transition region width over the stopband frequency. The stopband frequency fstop is 1 kHz, the Nyquist rate of the final 2 kHz sample rate. If we set Δf to 100 Hz, which is 10% of the final bandwidth, then F = 100 Hz/1 kHz = 0.1.

In[2]:=
```
dopt[128, 0.1]
```

Out[2]=
```
37.4733
```

We can now set D1 equal to the closest integer submultiple of D,

In[3]:=
```
Divisors[128]
```

Out[3]=
```
{1, 2, 4, 8, 16, 32, 64, 128}
```

which is D1 = 32.

In[4]:=
```
d1 = 32; d2 = 128 / d1
```

Out[4]=
```
4
```

Calculating filter parameters

Now we must find the FIR filter lengths and frequencies for each of the two stages. First define the

original sample rate going into the first-stage filter, and the sample rate going into the second-stage filter.

In[5]:=
```
sr1 = 256 000; sr2 = sr1 / d1
```

Out[5]=
```
8000
```

We'll need the stopband frequency of the second stage, which is the Nyquist rate after the second stage.

In[6]:=
$$\texttt{f2stop} = \frac{\texttt{sr2} / \texttt{d2}}{2}$$

Out[6]=
```
1000
```

For the first stage, we want to start filtering at 90% of the final 1 kHz bandwidth.

In[7]:=
```
f1cut = 0.9 f2stop
```

Out[7]=
```
900.
```

We need to set the stopband frequency of the first stage (f1stop) low enough to prevent aliasing in the second-stage output, but as high as possible to make the transition band wide, thus requiring a shorter filter. The stopband frequency of the first stage can be higher than the Nyquist rate after the first stage decimation, because we can allow aliasing in the band between f2stop (1 kHz) and 7 kHz, which is the sample rate after the first stage (8 kHz) minus the Nyquist after the second stage (1 kHz). For an explanation of this see Lyons Fig. 7-5.

In[8]:=
```
f1stop = sr2 - f2stop
```

Out[8]=
```
7000
```

Find the transition bandwidth of the first-stage filter, in radians. Note that this filter is applied to the 256 kHz data, so the transition bandwidth is calculated with respect to the Nyquist rate of 128 kHz (sr1/2).

In[9]:=
$$\texttt{tbw1} = \frac{\texttt{f1stop} - \texttt{f1cut}}{(\texttt{sr1} / 2)}\ \texttt{Pi}$$

Out[9]=
```
0.149717
```

Find the minimum length of the Blackman filter kernel for that transition bandwidth. Use the table on page 9 of [2].

In[10]:=
$$\texttt{m1} = \texttt{Ceiling}\left[\frac{12\ \texttt{Pi}}{\texttt{tbw1}}\right]$$

Out[10]=
```
252
```

For the second stage, we start filtering at the same frequency as the first stage.

In[11]:=
```
f2cut = f1cut
```

Out[11]=
```
900.
```

Find the transition bandwidth of the second-stage filter, in radians. Note that this filter is applied to the

8 kHz output of the first stage, so the Nyquist rate is 4 kHz.

In[12]:=
```
sr2 / 2
```

Out[12]=
```
4000
```

In[13]:=
```
tbw2 = (f2stop - f2cut) / (sr2 / 2) Pi
```

Out[13]=
```
0.0785398
```

Find the minimum length of the Blackman filter kernel for that transition bandwidth.

In[14]:=
```
m2 = Ceiling[12 Pi / tbw2]
```

Out[14]=
```
480
```

In summary, here are the specifications for the two stages. The cutoff frequency and transition bandwidth are
specified in radians over a range of 0 to Pi, with Pi being the Nyquist frequency.

In[15]:=
```
Grid[{{"D1", "Fc1", "Tbw1", "M1", "D2", "Fc2", "Tbw2", "M2"},
  {d1, f1cut 2 Pi / sr1, tbw1, m1, d2, f2cut 2 Pi / sr2, tbw2, m2}}, Frame → All,
 Dividers → {{True, True, True, True, Thickness[3], True, True, True},
   {True, Thickness[3], True}}]
```

Out[15]=

D1	Fc1	Tbw1	M1	D2	Fc2	Tbw2	M2
32	0.0220893	0.149717	252	4	0.706858	0.0785398	480

Calculating filter kernels

Now design filter kernels for the two stages of decimation. We want to use windowed sinc low-pass filters because they're very stable, linear-phase, and easy to calculate. Although Mathematica has many filter design functions, windowed sinc is not one of them and so we must write one. Steven Smith describes the filter design process in [3], chapter 16. We'll use a Blackman window, which has linear phase, 0.02 % ripple in the passband, and provides 74 dB of attenuation in the stopband. The filter cutoff frequency ω is specified to be between 0 and Pi representing 0 Hz to the Nyquist frequency. The length of the kernel produced is m+1 elements.

In[16]:=
```
windowedSincFilterKernel[ω_, m_] := Module[{kern, win},
  (* calculate sinc kernel *)
  kern = Table[Sinc[ω x], {x, -m / 2, m / 2}];
  (* calculate window *)
  win = Array[BlackmanWindow, m + 1, {-0.5, 0.5}];
  (* calculate windowed sinc kernel and normalize for 0 dB gain at DC *)
  kern win / Total[kern win]
 ] (* end Module *)
```

Note that a Python implementation of windowed sinc filters can be found in [4].

Calculate the decimation filter kernels for the two stages of decimation and look at their responses.

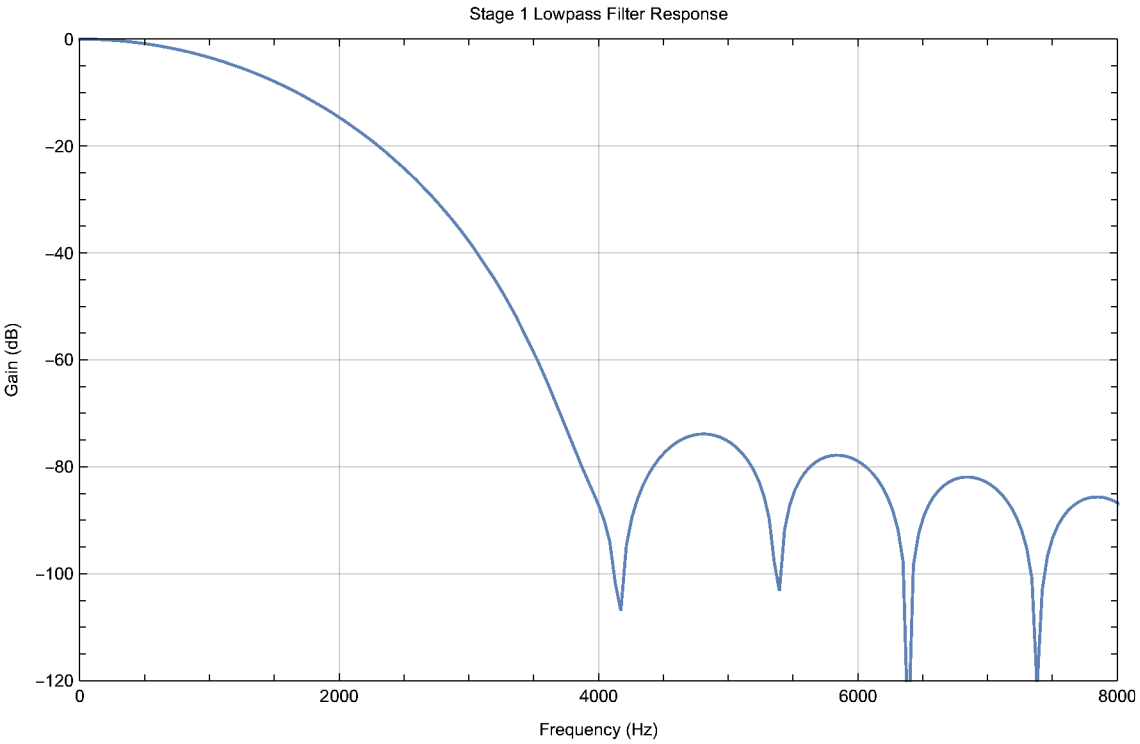
In[17]:=
```
{f1cut, f1stop, m1}
```

Out[17]=
```
{900., 7000, 252}
```

In[18]:=
```
h1 = windowedSincFilterKernel[2 Pi f1cut / sr1, m1];
```

In[19]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h1, 2 Pi f / sr1]]], {f, 0, sr1 / 2},
 PlotRange → {{0, sr2}, {-120, 0}}, Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 1 Lowpass Filter Response"}}]
```

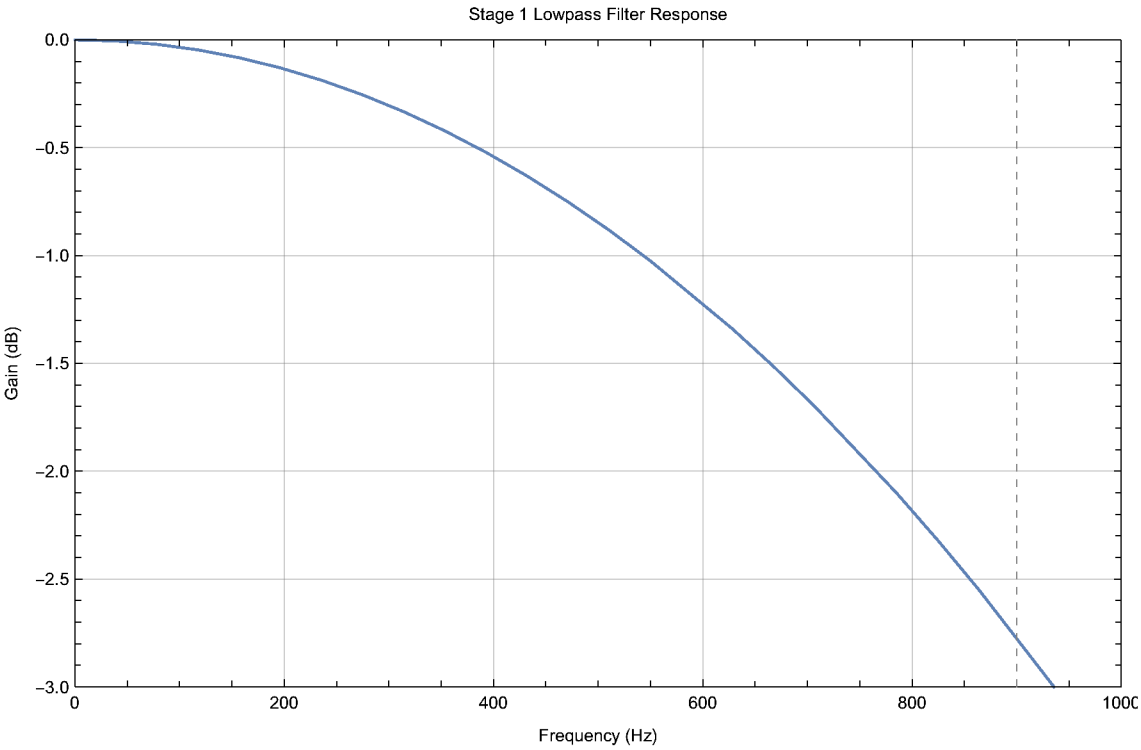
Out[19]=



We'd like to have very little attenuation below 900 Hz in order to preserve signal amplitudes in the final passband. We can zoom in to the upper left portion of the filter response to evaluate this better.

In[20]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h1, 2 Pi f / sr1]]], {f, 0, sr1 / 2},
 PlotRange → {{0, 1000}, {-3, 0}}, Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 1 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{900, -3}, {900, 0}}]}]
```

Out[20]=



We're about 2.75 dB down at 900 Hz. This makes sense, since the cutoff frequency is the "-3 dB" point, but it's a bit too much attenuation in our band of interest. We can raise the cutoff frequency of the filter as long as we still have at least 74 dB of attenuation at 7 kHz.

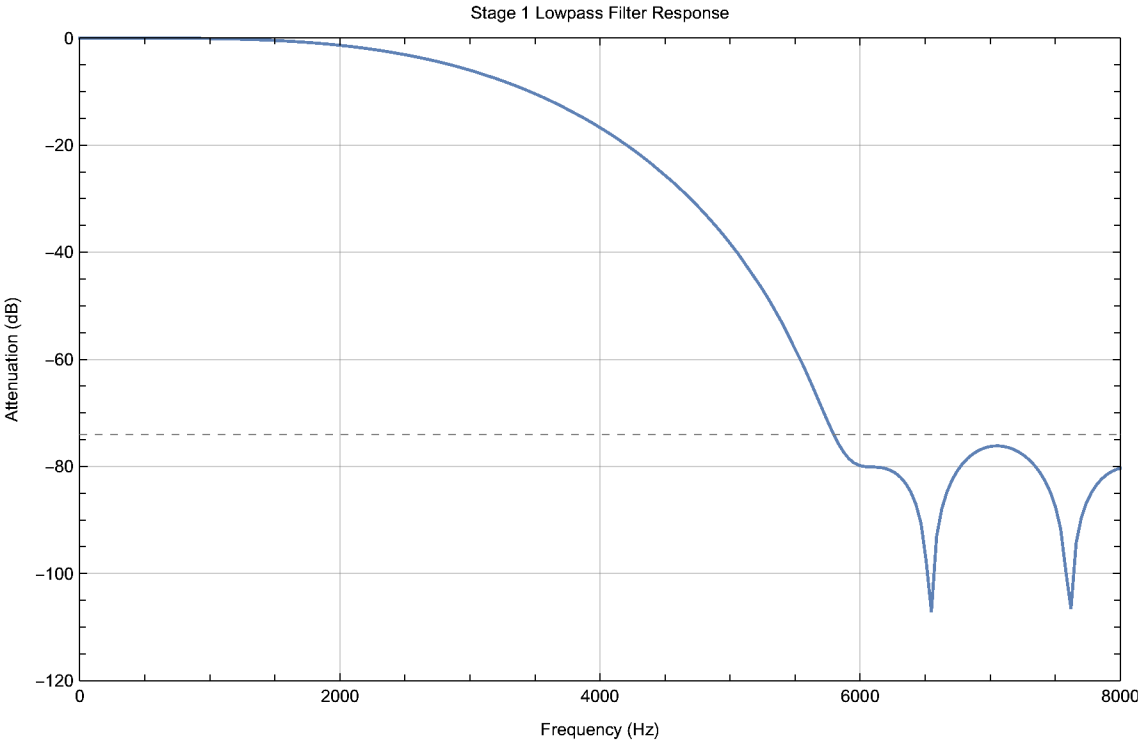
In[21]:=
```
f1cut = 3000
```

Out[21]=
```
3000
```

In[22]:=
```
h1 = windowedSincFilterKernel[2 Pi f1cut / sr1, m1];
```

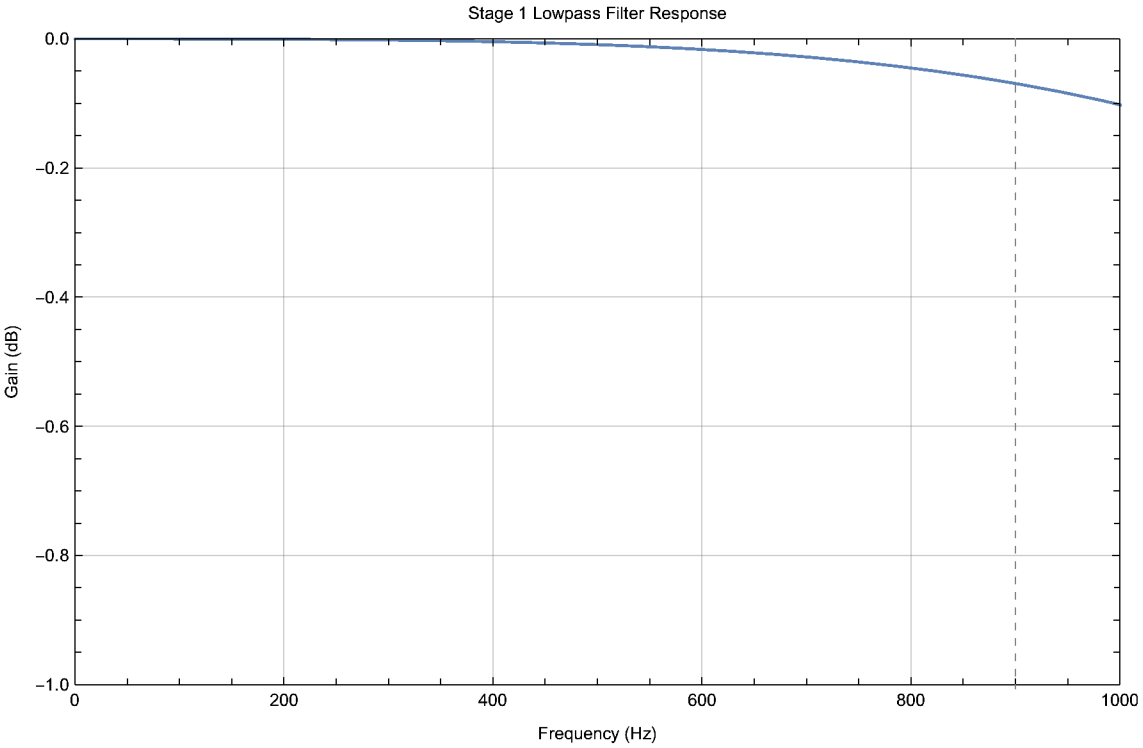
In[23]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h1, 2 Pi f / sr1]]],
 {f, 0, sr1 / 2}, PlotRange → {{0, sr2}, {-120, 0}}, Frame → True,
 GridLines → Automatic, FrameLabel → {{"Attenuation (dB)", None},
   {"Frequency (Hz)", "Stage 1 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{0, -74}, {8000, -74}}]}]
```

Out[23]=



In[24]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h1, 2 Pi f / sr1]]], {f, 0, sr1 / 2},
 PlotRange → {{0, 1000}, {-1, 0}}, Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 1 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{900, -3}, {900, 0}}]}]
```

Out[24]=



Now we have very little attenuation anywhere in the final passband. This means we probably could've used a shorter FIR filter but we'll leave it as is for now.

Design the second stage filter.

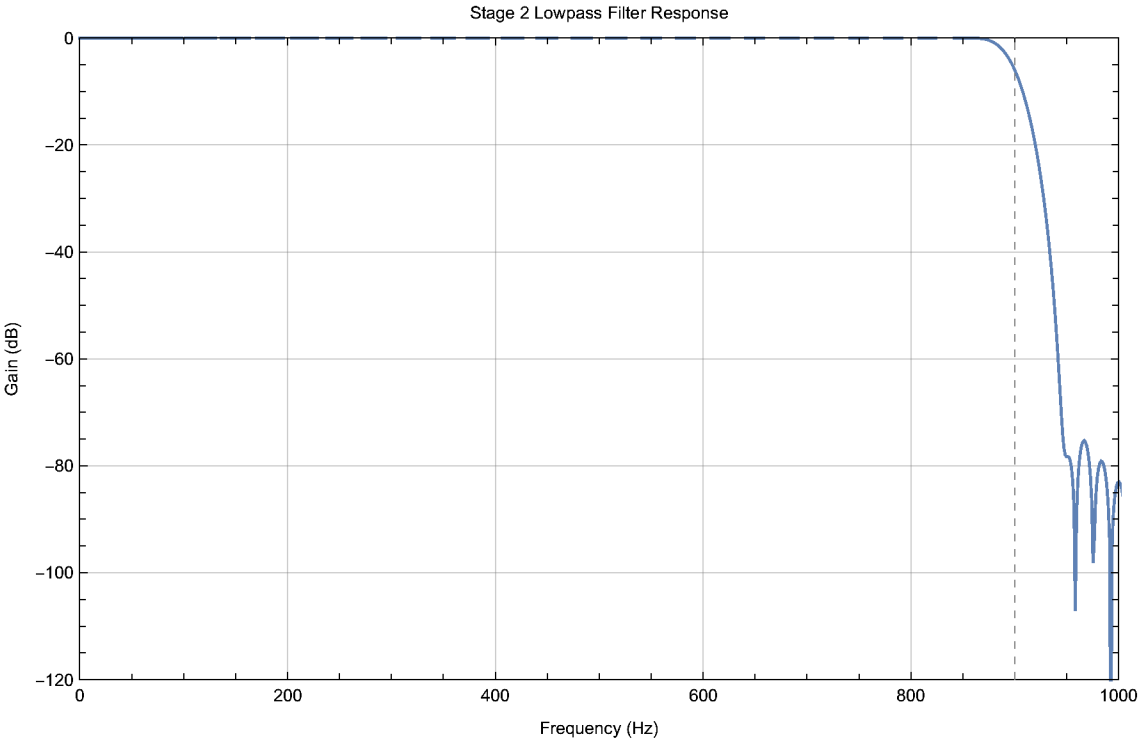
In[25]:=
```
{f2cut, f2stop, m2}
```

Out[25]=
```
{900., 1000, 480}
```

In[26]:=
```
h2 = windowedSincFilterKernel[2 Pi f2cut / sr2, m2];
```

In[27]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h2, 2 Pi f / sr2]]],
 {f, 0, sr2 / 2}, PlotRange → {{0, f2stop}, {-120, 0}},
 Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 2 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{900, -120}, {900, 0}}]}]
```

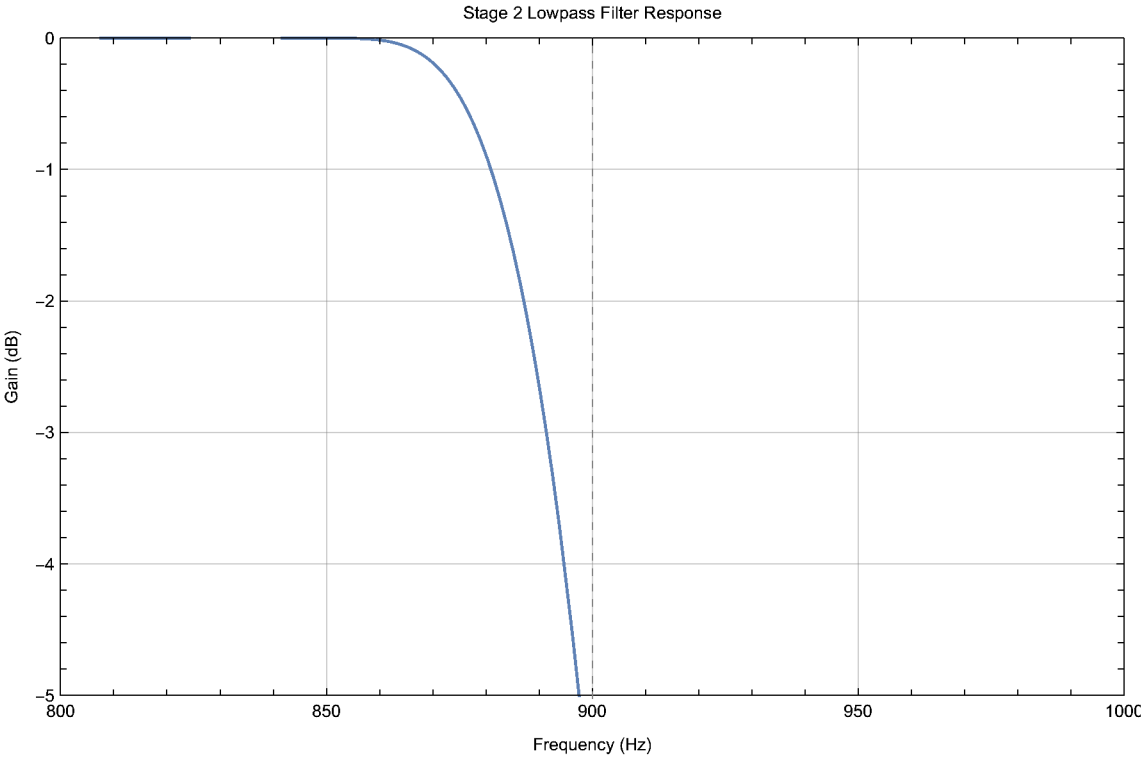
Out[27]=



Once again we'll zoom into the upper portion of the graph to verify that there's not much attenuation in the passband.

In[28]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h2, 2 Pi f / sr2]]],
 {f, 0, sr2 / 2}, PlotRange → {{800, f2stop}, {-5, 0}},
 Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 2 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{900, -120}, {900, 0}}]}]
```

Out[28]=



Again the attenuation in the passband is too much and we can adjust the cutoff frequency to be higher.

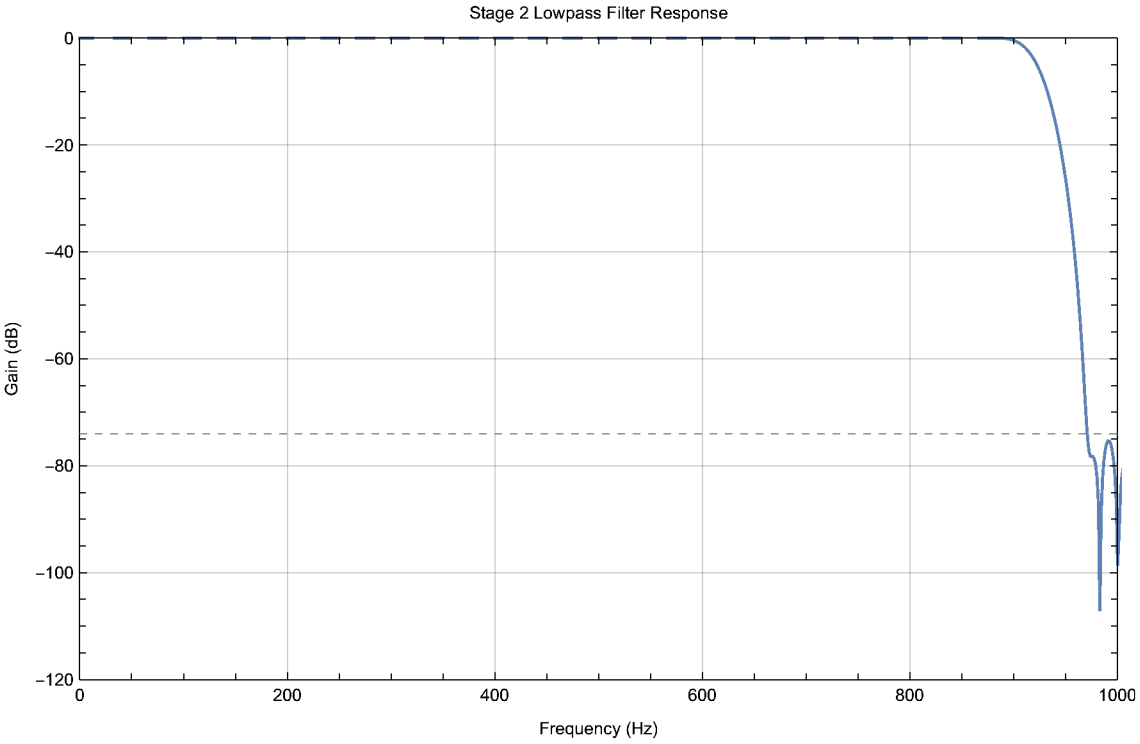
In[29]:=
```
f2cut = 925
```

Out[29]= 925

In[30]:=
```
h2 = windowedSincFilterKernel[2 Pi f2cut / sr2, m2];
```

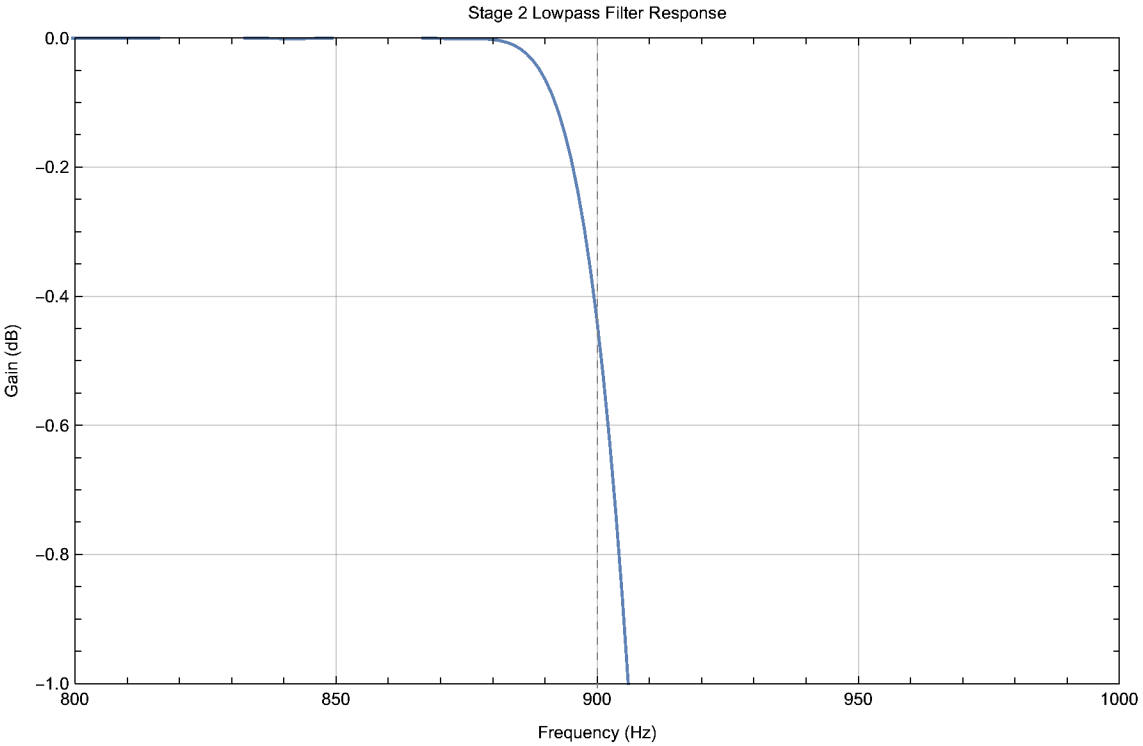
In[31]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h2, 2 Pi f / sr2]]],
 {f, 0, sr2 / 2}, PlotRange → {{0, f2stop}, {-120, 0}},
 Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 2 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{0, -74}, {1000, -74}}]}]
```

Out[31]=



In[32]:=
```
Plot[20 Log[10, Abs[ListFourierSequenceTransform[h2, 2 Pi f / sr2]]],
 {f, 0, sr2 / 2}, PlotRange → {{800, f2stop}, {-1, 0}},
 Frame → True, GridLines → Automatic, FrameLabel →
  {{"Gain (dB)", None}, {"Frequency (Hz)", "Stage 2 Lowpass Filter Response"}},
 Epilog → {Gray, Dashed, Line[{{900, -120}, {900, 0}}]}]
```

Out[32]=



Here are the modified filter parameters.

In[33]:=
```
Grid[{{"D1", "Fc1", "Tbw1", "M1", "D2", "Tbw2", "Fc2", "M2"},
  {d1, f1cut 2 Pi / sr1 // N, tbw1, m1, d2, f2cut 2 Pi / sr2 // N, tbw2, m2}},
 Frame → All, Dividers → {{True, True, True, True, Thickness[3], True, True, True},
   {True, Thickness[3], True}}]
```

Out[33]=

D1	Fc1	Tbw1	M1	D2	Tbw2	Fc2	M2
32	0.0736311	0.149717	252	4	0.726493	0.0785398	480

Filtering hydrophone data

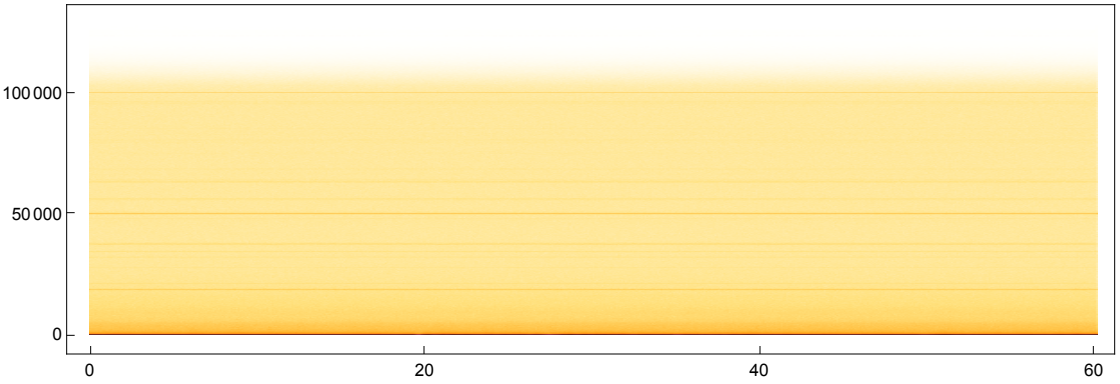
Now we can apply this decimation to data from the hydrophone to produce 2 kHz data. Import the second minute of a ten-minute data file that contains whale calls.

In[34]:=
```
whales = Take[Import["/Users/mcgill/Documents/My\ Projects/Passive\
        Acoustic\ Monitoring/MARS_20171003_200022.wav", "Data"][[1]],
    {60
      sr1,
     120
      sr1}];
```

Look at both the spectrogram and periodogram of the data. Note that we must amplify the data by 80 dB to see it on the spectrogram. Also note the attenuation of frequencies above 100 kHz due to the antialiasing filter in the hydrophone.

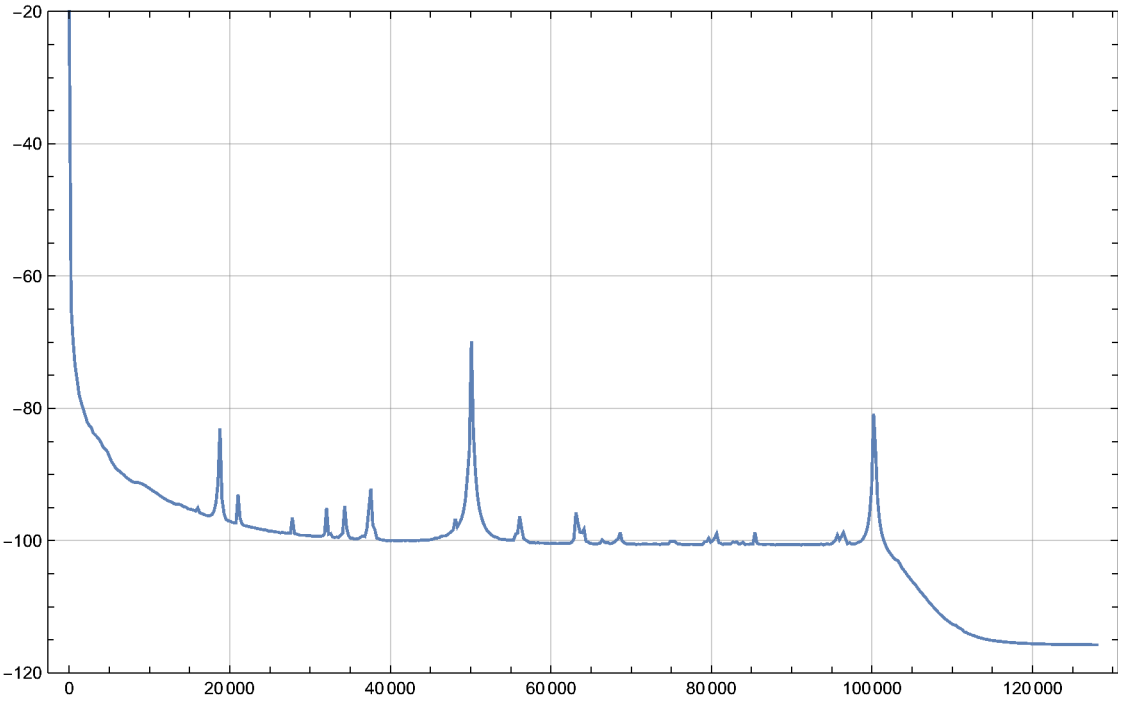
In[35]:=
```
Spectrogram[1 × 10^4 whales, SampleRate → 256 000, PlotRange → All]
```

Out[35]=



In[36]:=
```
Periodogram[whales, 1024, SampleRate → 256 000,
 PlotRange → {All, {-120, -20}}, Frame -> True, GridLines → Automatic]
```

Out[36]=



Filter and downsample the data, with the first stage producing 8 kHz data and the second stage produc-

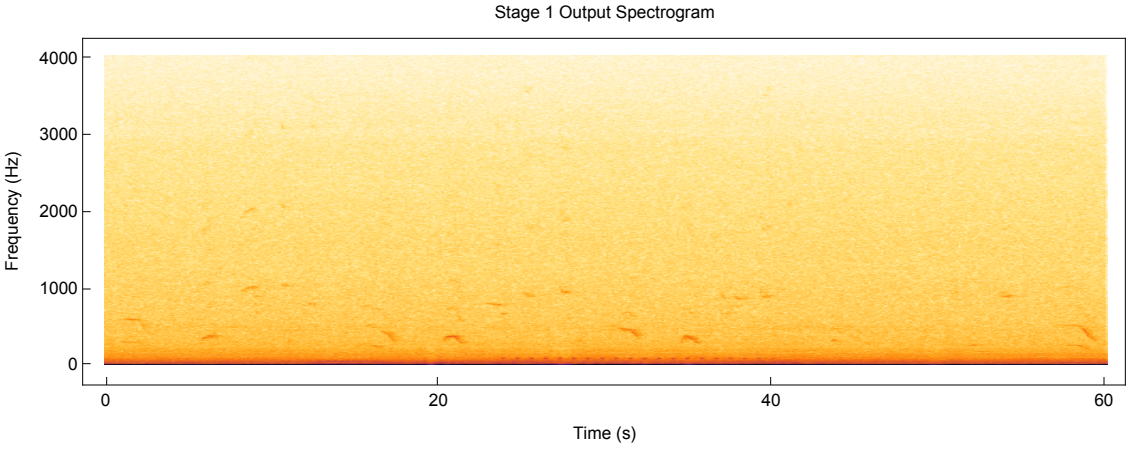
ing 2 kHz data.

In[37]:=
```
whales8kHz =
  Downsample[
   ListConvolve[h1, whales],
   d1];
```

In[38]:=
```
whales2kHz =
  Downsample[
   ListConvolve[h2, whales8kHz],
   d2];
```

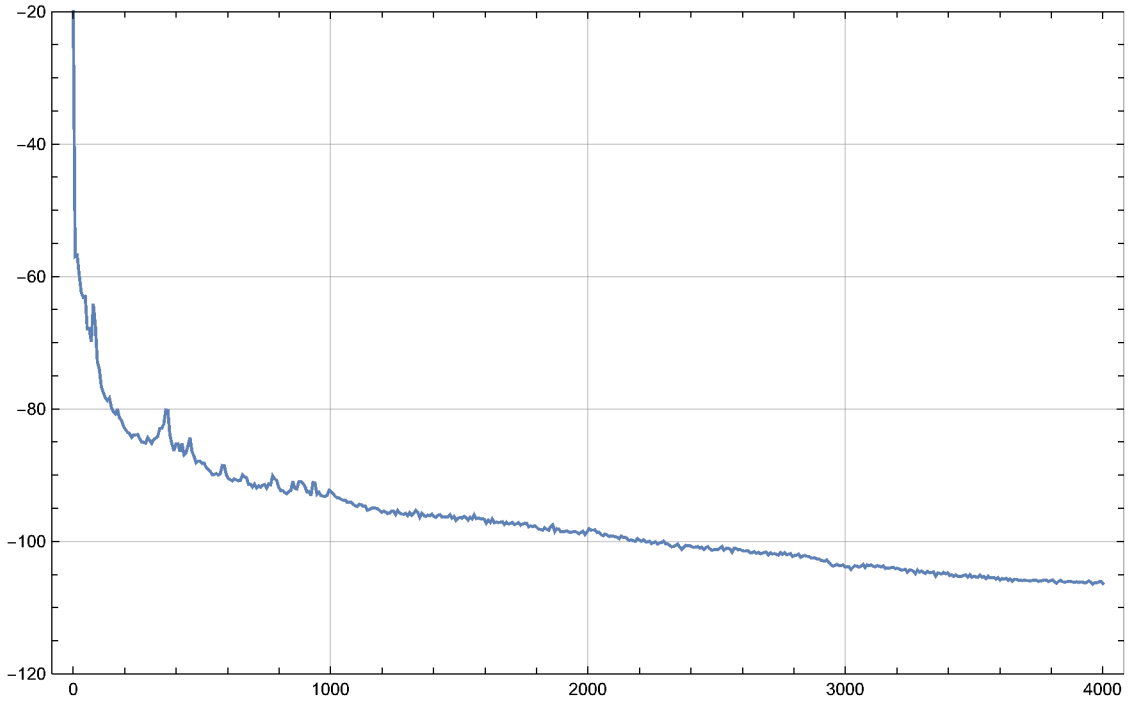
In[39]:=
```
Spectrogram[5 × 10^4 whales8kHz, 1024, SampleRate → 8000, PlotRange → All, Frame -> True,
 FrameLabel → {{"Frequency (Hz)", ""}, {"Time (s)", "Stage 1 Output Spectrogram"}}]
```

Out[39]=



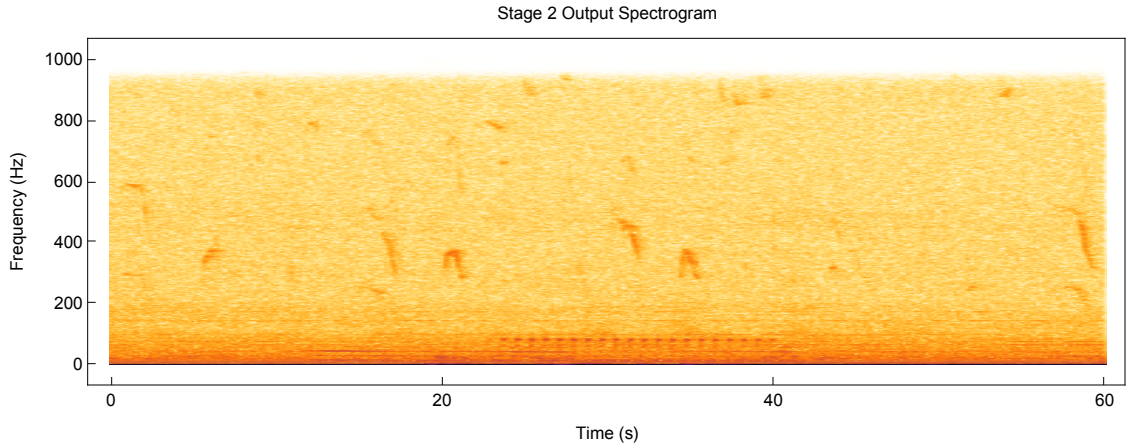
In[40]:=
```
Periodogram[whales8kHz, 1024, SampleRate → 8000,
 PlotRange → {All, {-120, -20}}, Frame -> True, GridLines → Automatic]
```

Out[40]=



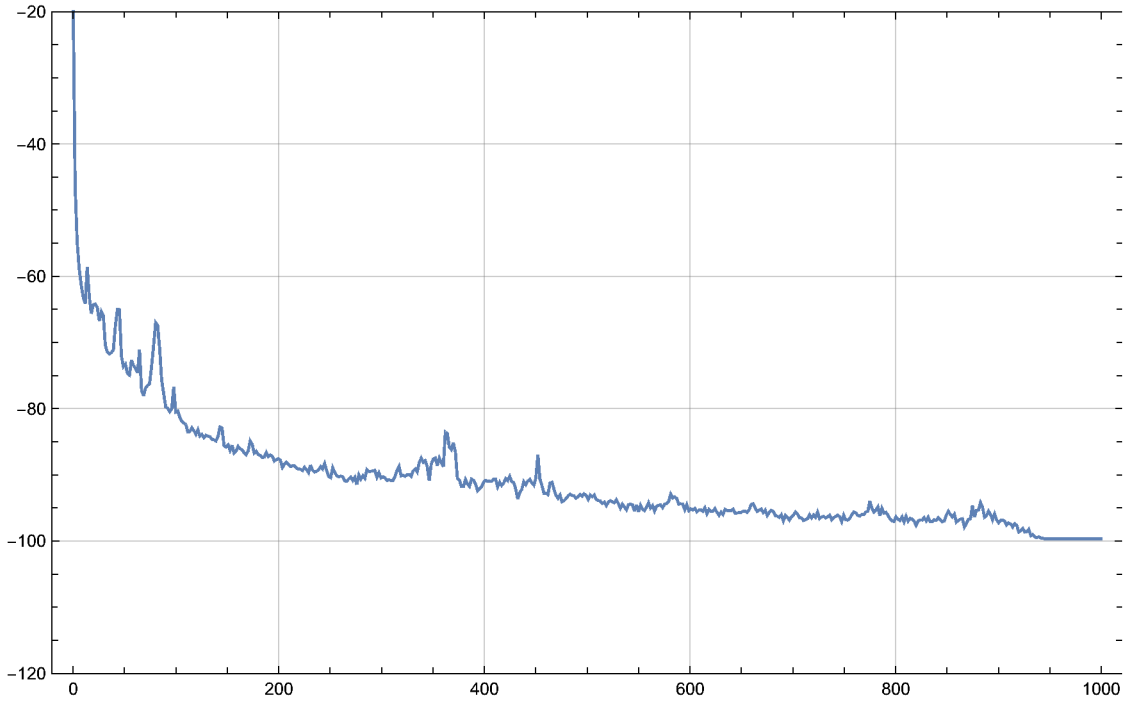
In[41]:=
```
Spectrogram[5 × 10^4 whales2kHz, 1024, SampleRate → 2000, PlotRange → All, Frame -> True,
 FrameLabel → {{"Frequency (Hz)", ""}, {"Time (s)", "Stage 2 Output Spectrogram"}}]
```

Out[41]=



In[42]:=
```
Periodogram[whales2kHz, 1024, SampleRate → 2000,
 PlotRange → {All, {-120, -20}}, Frame -> True, GridLines → Automatic]
```

Out[42]=



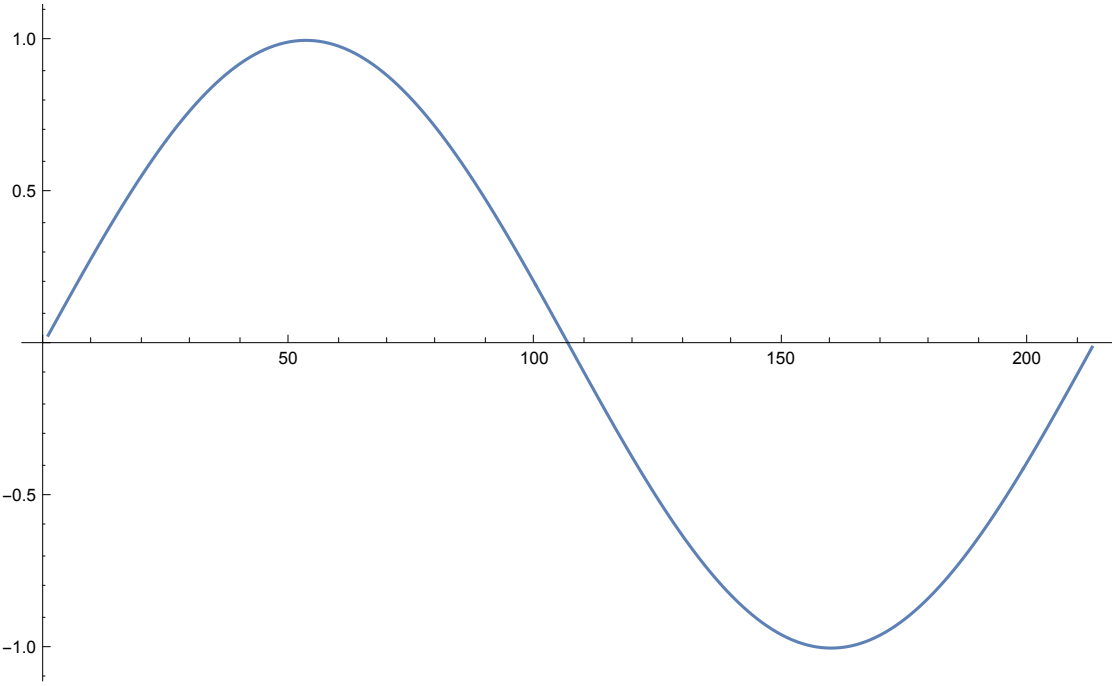
Verifying the results

The downsampled data clearly show the whale calls and the attenuation above 900 Hz, but how do we know that the low-pass filters are properly preventing aliasing from strong signals just above the passband? To test this create a 1200 Hz tone at 256 ksps that lasts one minute.

In[43]:=
```
tone = Table[Sin[2 Pi x 1200 / sr1], {x, 1, Length[whales]}];
```

In[44]:=
```
ListPlot[tone[[ ;; 213]], Joined → True]
```

Out[44]=



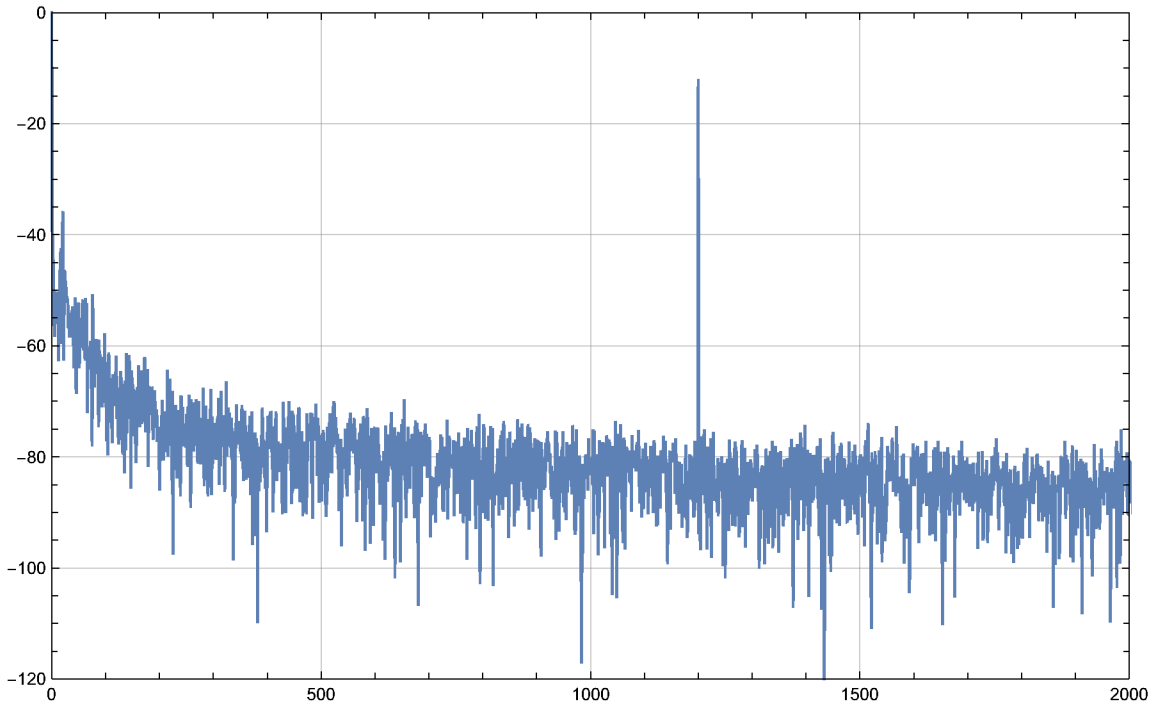
Attenuate the tone by 60 dB and add it to the whale data.

In[45]:=
```
whaletone = whales + 10^-3 tone;
```

Look at the spectrum of the combined signal.

In[46]:=
```
Periodogram[whaletone[[ ;; 256 000]], SampleRate → 256 000,
 PlotRange → {{0, 2000}, {-120, 0}}, Frame -> True, GridLines → Automatic]
```

Out[46]=



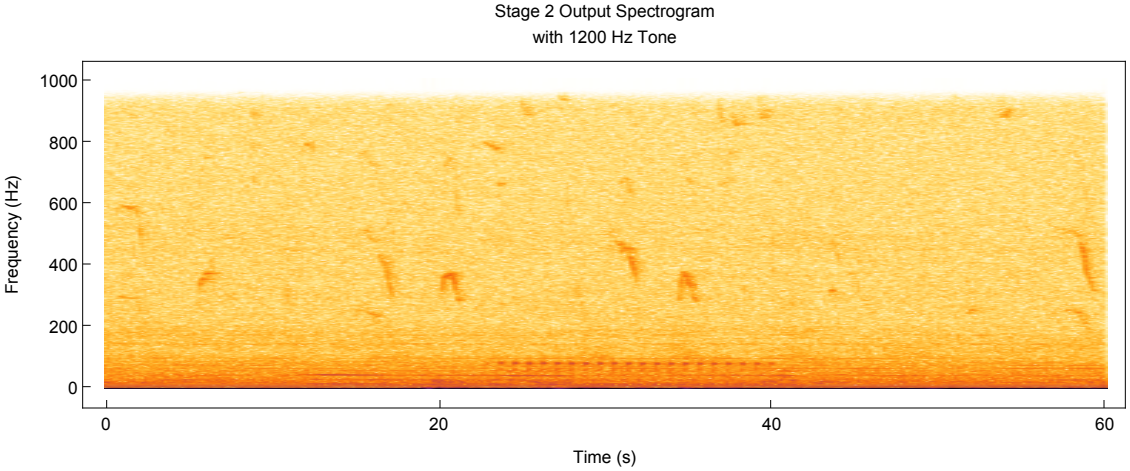
The tone is very strong at about 70 db louder than the natural signals we see. Now do the same analysis on the data containing the tone and look at the resulting spectrum.

In[47]:=
```
whaletone8kHz =
  Downsample[
   ListConvolve[h1, whaletone],
   d1];
```

In[48]:=
```
whaletone2kHz =
  Downsample[
   ListConvolve[h2, whaletone8kHz],
   d2];
```

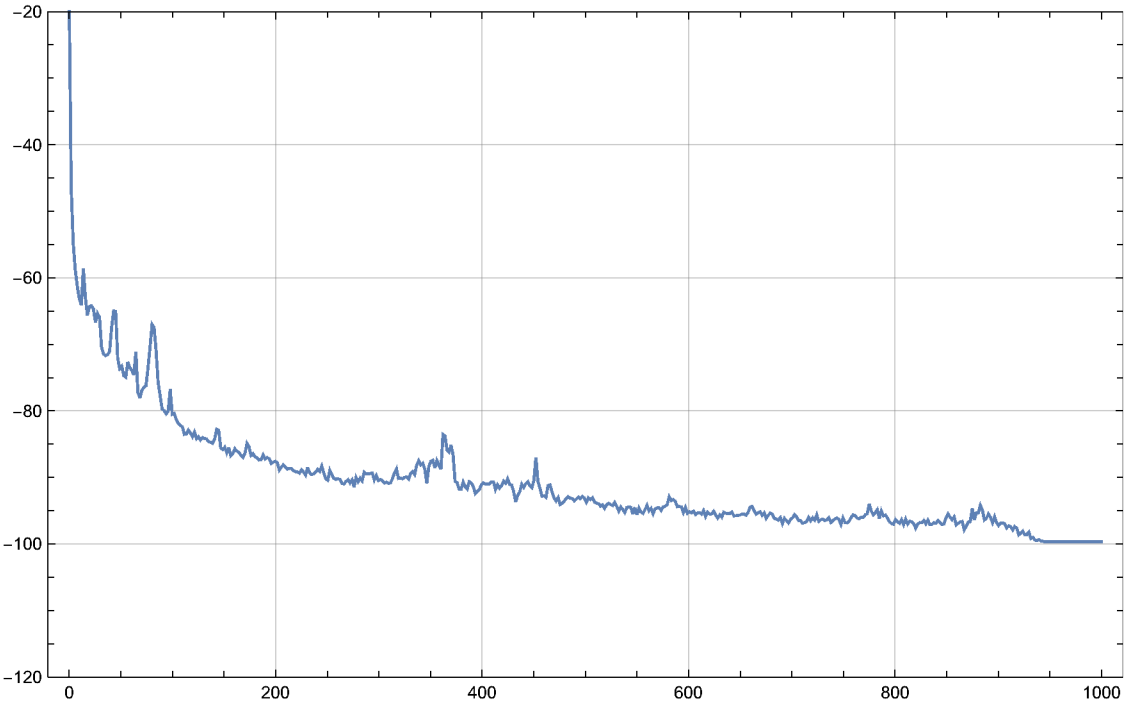
In[49]:=
```
Spectrogram[5 × 10^4 whaletone2kHz, 1024, SampleRate → 2000,
 PlotRange → All, Frame -> True, FrameLabel → {{"Frequency (Hz)", ""},
  {"Time (s)", "Stage 2 Output Spectrogram\rwith 1200 Hz Tone"}}]
```

Out[49]=



In[50]:=
```
Periodogram[whaletone2kHz, 1024, SampleRate → 2000,
 PlotRange → {All, {-120, -20}}, Frame -> True, GridLines → Automatic]
```

Out[50]=



The spectrogram shows no hint of a tone at 800 Hz, which is where the 1200 Hz tone would appear if it had not been sufficiently filtered.

Implementation notes

The analysis above doesn't address three issues that might be important in a practical implementation:

Filter delays

Although most implementations of windowed sinc filters are linear phase, they're not zero phase, and therefor will introduce a delay in the output signal equal to one-half the filter length. If exact timing is important for subsequent analysis, then the output samples should be shifted earlier in time accordingly. See [3], chapter 19, page 328 for an explanation of filter phase response.

The delay due to the first-stage filter is half the filter length times the time between samples at the original sample rate, as described in [1], section 5.8.

In[62]:=
$$\text{delay1} = \frac{\text{Length[h1]} - 1}{2} \frac{1}{\text{sr1}} \text{ // N}$$

Out[62]= 0.000492188

The delay due to the second-stage filter is half the filter length times the time between samples after the first-stage decimation.

In[63]:=
$$\text{delay2} = \frac{\text{Length[h2]} - 1}{2} \frac{1}{\text{sr2}} \text{ // N}$$

Out[63]= 0.03

The total delay is the sum of the first- and second-stage delays.

In[64]:=
$$\text{totalDelay} = \text{delay1} + \text{delay2}$$

Out[64]= 0.0304922

This delay represents a certain number of samples at the final decimated sample rate (2000 Hz in the examples above). The final output should be shifted forward in time this many samples to adjust for filter delays.

In[68]:=
$$\text{Round}\left[\text{totalDelay} \frac{\text{sr2}}{\text{d2}}\right]$$

Out[68]= 61

End effects during block processing

As with any digital filtering operation, the effects at the beginning and end of each file (i.e. block) must be considered. In the analysis above, I didn't worry about this and there is no overhang between the filter kernel and either end of the data. To process long, continuous segments of data in multiple files, the overlap-add method should be used to ensure that no data are lost at the beginning and end of each file. A description of overlap-add can be found in [3], chapter 18.

Filter efficiency

The filtering operations in the analysis above are very inefficient. For instance in the case of decimation by a factor of 32, every output value of the filter is calculated in the filtering step. This is immediately

followed by the downsampling step, where 31 out of 32 filter output values are discarded. It's much more efficient to calculate only every 32nd output value, which is possible because FIR filters don't depend on past filter outputs. This requires additional bookkeeping overhead that may not save time for small decimation factors.

References

[1] Lyons, R. G. (2004). *Understanding digital signal processing* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.
[2] Notes on FIR filter design using window functions. (n.d.). Retrieved March 6, 2018, from http://-course.ece.cmu.edu/~ece396/lectures/L19/WindowFIRDesign.pdf
[3] Smith, S. W. (1997). *The scientist and engineers guide to digital signal processing.* San Diego (California): California Technical Publishing. (accessible for free at http://www.dspguide.com/pdfbook.htm)
[4] How to Create a Simple Low-Pass Filter. (n.d.). Retrieved March 06, 2018, from https://tomroelandts.-com/articles/how-to-create-a-simple-low-pass-filter