7

ADC Laboratory

7.1 Purpose 7-2

7.2 Background 7-2

7.2.1 Downsampling 7-2

7.2.2 Kaiser filter 7-4

7.2.3 Upsampling 7-7

Upsampling a simple sinusoidal input 7-7

Upsampling speech 7-7

Some questions 7-9

7.2.4 Resampling 7-9

Some questions 7-9

7.3 Assignment 7-9

7.1 Purpose

The purpose of this laboratory is to design and implement a routine that will resample (upsample and/or downsample) a sequence.

7.2 Background

To prepare for this exercise, review Chapter 6. There are three cases we wish to consider: downsample, upsampling and resampling.

7.2.1 Downsampling

Consider the case in which you wish to downsample an input sequence, $x[n]$ by a factor of D. Why can't you just form the output sequence, $y[n]$, by keeping only every D^{th} point of $x[n]$? Follow along with the following example to understand why.

1. First, download and listen to a snippet of speech, $x[n]$:

```
>> [x, fs] = audioread('seashell.wav');
>> soundsc(x, fs);
```

This speech has been sampled at $f_S = 11.025\,\text{kHz}$ after having been properly anti-alias filtered by the A/D converter (at what frequency?). Figure 7-1 shows the spectrogram of speech:

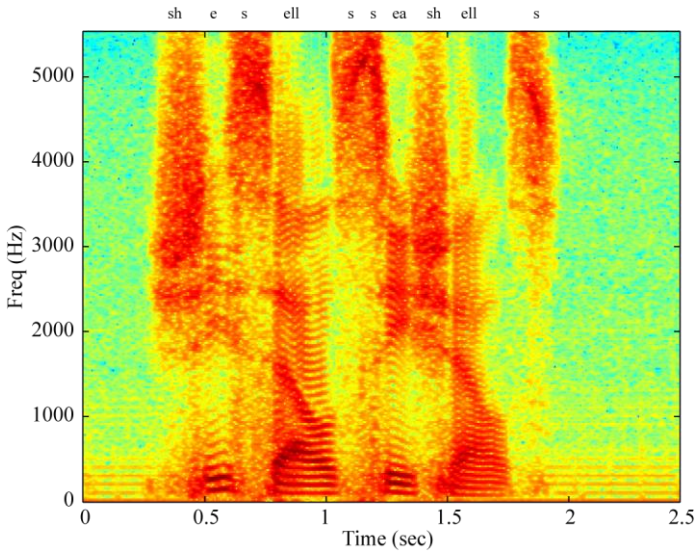

Figure 7-1: Spectrogram of "She sells seashells"

The spectrogram is basically a plot of the energy in the speech as a function of frequency and time. More exactly, what you are looking at is the computed as follows:

- Take the sequence, $x[n]$, and break it up into a series of overlapping frames. Each frame is 512 points long (about 46.4 ms), and each frame overlaps with previous one by 448 points (about 5.8 ms).

- For each frame, m, compute the DFT, $X_m[k]$,using Matlab's `fft` routine. This gives us, for each frame, a 512 point sequence, $X_m[k], 0 \le k < 512$ that corresponds to the Fourier transform, $X_m(\omega)$ for that frame of speech, with ω being 512 points spaced between 0 and 2π , $0 \le \omega < 2\pi$. Note that the 512th point doesn't correspond to 2π , but rather $2\pi \cdot 511/512$
- Keep only the 257 points, $X_m[k], 0 \le k \le 256$ that correspond to $X_m(\omega)$ with frequencies, $0 \le \omega \le \pi$ (note: this includes π). These discrete-time frequencies correspond to the continuous-time frequencies between 0 Hz and half the sampling rate, or $0 \le f \le 5512.5$ Hz.
- For each frame of speech, compute the magnitude of $X_m(\omega)$ on a log (dB) scale.
- Plot the spectrogram as a three-dimensional plot. For each frame of spectrogram, the magnitude, $|X_m(\omega)|$, is plotted as a thin vertical slice centered at $m*5.8$ msecs on the *x*-axis. The *y*-axis has frequencies 0-5512.5 Hz. The colors of the plot indicate the magnitude, $|X_m(\omega)|$, at each frequency. The hotter colors such as red indicate the highest magnitudes; the cooler colors, such as green and blue are the lower magnitudes.
- Above the plot you can see the transcript of the utterance "She sells seashells". You can see that the consonants, like 'sh' and 's' have a good deal of energy above 3 kHz, while the vowels, /e/, have energy at lower frequencies.

2. Now, simply decimate, which means throwing out every other sample of $x[n]$ to form $y[n] = x[2n]$, and playback the result at 5.5 kHz. If you've done it right (or should I say, "wrong"), it should sound like the file `seadown_bad.wav`, that is to say, terrible! What is going on? Aliasing is what! You are hearing the high frequencies of the 's' and 'sh' being aliased to lower frequencies. Figure 7-2 shows the spectrogram of $y[n]$.

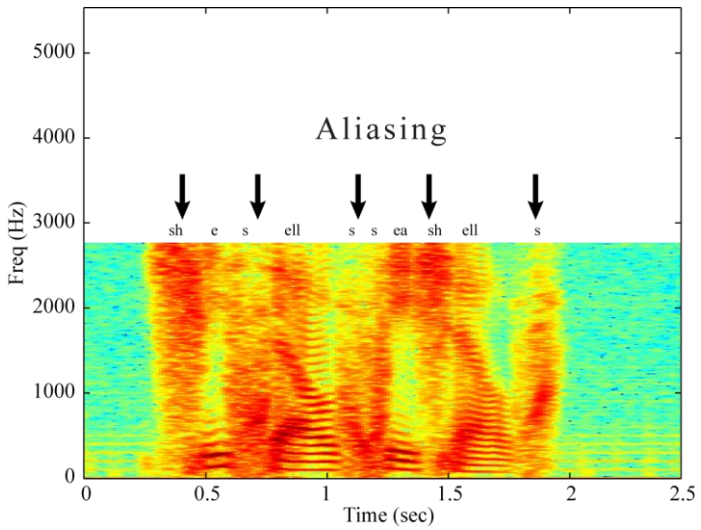

Figure 7-2: "Seashell" with aliasing

If you look at Figure 7-2 and compare it with Figure 7-1, you can see a few things:

- The spectral frequencies of Figure 7-2 only go from 0-2756 Hz.

- During the times of the consonants (indicated by the arrows) there is clearly an excess of spectral energy at low frequencies in Figure 7-2 that was not in Figure 7-1. What is going on??? Aliasing! All the frequencies of the spectrogram above $f_s/4$ are being 'folded over' to lower frequencies.

  For example, if you look at the final 's' in the word 'seashells', you can see that the energy in the consonant, which used to extend from about 3500-5500 Hz in Figure 7-1, has been folded over and now goes from 2000Hz to 0 Hz. Note that the piece of the spectrogram at 5500 Hz in Figure 7-1 maps to 0 Hz in Figure 7-2.

3. Now, compare what happens if we properly downsample $x[n]$. In order to do this, use Matlab's `resample` command:

```
>> z = resample(x, 1, 2);
>> soundsc(z, fs/2);
```

If you did it right, it should sound like `seadown.wav`. It sounds a little muffled, but at least the signal isn't aliased anymore. Figure 7-3 shows the spectrogram of the correctly downsampled signal, $z[n]$:

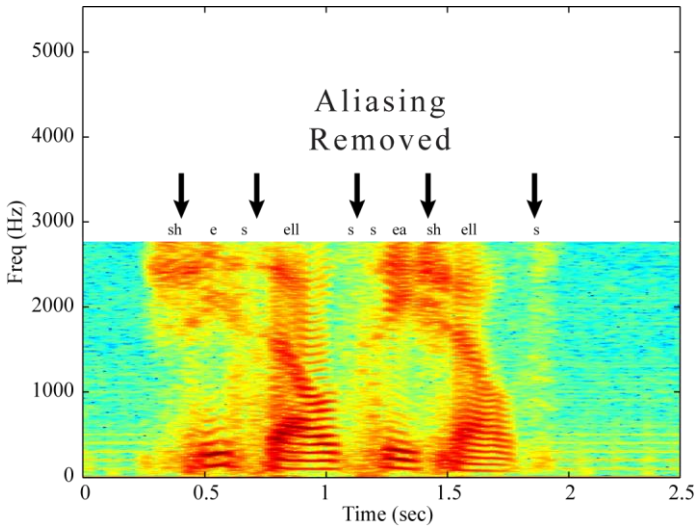

Figure 7-3: "Seashell" with aliasing removed

As you can see, the aliasing is no longer a problem. The Matlab `resample` command properly filters $x[n]$ with a well- designed discrete-time filter, $h[n]$, before it downsamples it.

7.2.2 Kaiser filter

In class we discussed the case where $h[n]$ was an ideal lowpass filter. Of course it is not possible to use an ideal lowpass filter in this laboratory. However, there are many practically realizable filters that can be used in this interpolation application. In our lab this week, we will design a sharp, practical lowpass filter, called a Kaiser filter using the Matlab `fir1` routine. This filter is basically a sinc function which has been multiplied by a window function (called a Kaiser window) to form a finite-length impulse response (FIR) filter. To create the impulse response of a kaiser filter with a length, N, and a cutoff frequency of `wc`, we let Matlab do all the work, as follows:

```
>> h = fir1(N-1, wc/pi, kaiser(N, 5));
```

Note that the second argument, `wc/pi` is just the cutoff frequency expressed as a fraction of π. This parameter can vary between 0 and 1 (corresponding to half the sample rate). Note also that the first argument to `fir1` is `N-1`, not `N`.

Figure 7-4 shows the effect of the length the filter's impulse response, N, on the frequency respnose of the filter.

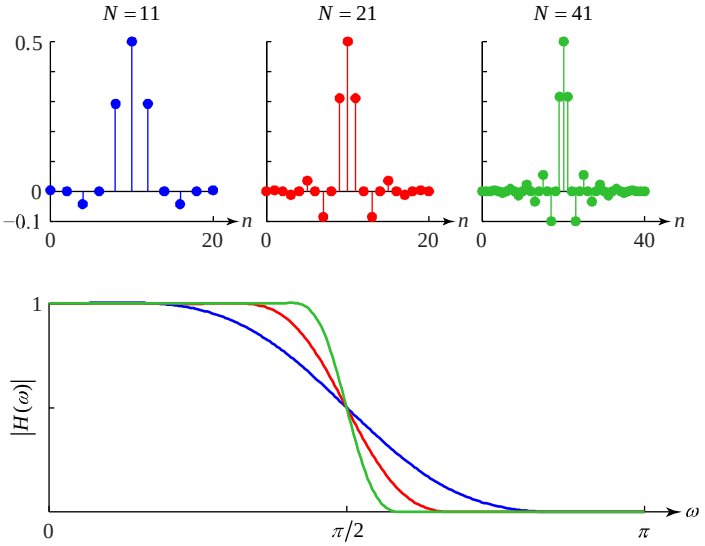

Figure 7-4: Kaiser filter as a function of length

The top panel of Figure 7-4 shows the impulse responses of three different Kaiser filters created with $\omega_c = \pi/2$. The bottom panel is the spectrum (i.e. the DTFT) of these filters. We've plotted only the positive frequencies. As you increase the order of the filter, the cutoff (i.e. the slope of the spectrum around $\pi/2$) gets sharper.

A few little details:

- In this week's lab, in order to match the way that Matlab's resample function works, we'll actually make N depend on the ω_c too using the following formula:

```
>> fn = wc/pi; % normalized frequency corresponds to 0 --> pi
>> N = round(1+20/fn); % order increases as fn decreases
>> h = fir1(N, fn, kaiser(N, 5));
```

The only reason to do this is so that when we try to plot the results of Matlab's resample function against our own, the results should match pretty well. You might think of making your own little

function that takes a value of ω_c as an argument and returns the appropriate Kaiser filter impulse response, $h[n]$. At this point, let's not worry a lot about filter design. That was the topic of another lab.

- The other thing you might notice about Matlab's resample function is this. When we downsample 'seashell' we see the following:

```
>> [x, fs] = wavread('seashell');
>> length(x)
ans =
       27560

>> y = resample(x, 1, 2);
>> length(y)
ans =
       13780
```

As you can see, the length of the downsampled sequence is exactly half the length of the original sequence. However, when you create your resample function, you there are two ways of filtering $h[n]$ with your $x[n]$, either using Matlab's `conv` or `filter` functions, and then downsampling:

```
>> y = conv(x, h);
```

or

```
>> y = filter(h, 1, x);
```

Depending on the function you use, you may find that this procedure leaves some extra points at the beginning and end of your resampled sequence. It will be up to you to figure out how many extra points there are (if any). If there are extra points, you need to get rid of them so that your resample function and Matlab's resample function will match as well as possible.

Some questions (please answer in writing):

- The spectrograms in Figure 7-2 and Figure 7-3 only go to 2756 Hz instead of 5512 Hz. Why is this?
- How should the bandwidth of the filter we used to get Figure 7-3 (i.e. ω_c) depend on D, the integer value of downsampling?
- The sound in Figure 7-3 is not aliased, but it does seem very muffled compared to the original signal. Why?
- Why couldn't we have used an ideal lowpass filter instead of the Kaiser filter?

7.2.3 Upsampling

Upsampling a simple sinusoidal input

Figure 7-5 shows the process of upsampling a sinusoidal input, $x[n] = \sin \pi n/2$, by a factor of $U = 4$.

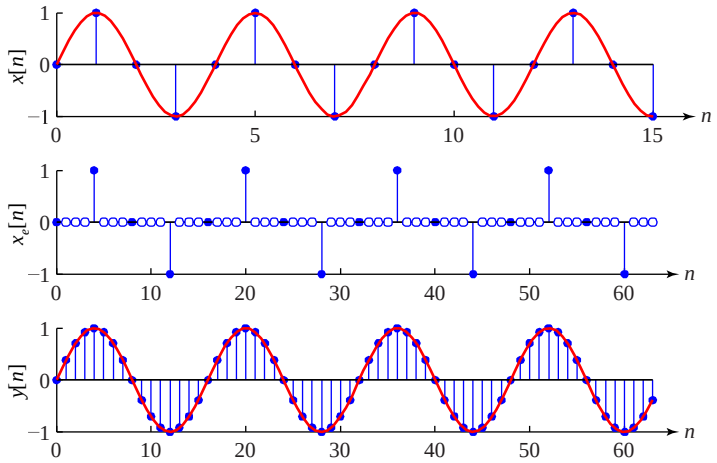

Figure 7-5: Upsampling a cosine by a factor of four

- The top panel of Figure 7-5 shows a continuous sine wave (red curve) of frequency, f_C, sampled at a frequency of $f_S = 4 f_C$, that is, at 4 samples per cycle. The resulting sequence is $x[n] = \sin \pi n/2$.
- The middle panel shows the expanded discrete-time waveform, $x_e[n]$, obtained by inserting $U - 1$ zeros "between" each pair of points in $x[n]$. This is easily done in Matlab without using a `for` loop. We'll use the fact that Matlab can index into an array in sophisticated ways. Specifically, consider the following:

```
>> x = [1 5 4 2 6];
>> y = zeros(1, 10);
>> y(1:2:10) = x
y =
     1 0 5 0 4 0 2 0 6 0
```

- The bottom panel of Figure 7-5 shows $y[n]$, the upsampled (interpolated) output waveform derived from filtering $x_e[n]$ by $h[n]$, the impulse response of a properly designed lowpass filter.

Upsampling speech

Now consider upsampling $x[n]$, a speech segment of 'seashell', by a factor of $U = 2$.

- Create a new sequence, $y[n]$, by interpolating one zero between each pair of points of $x[n]$.
- Listen to $y[n]$. What sample rate do you have to use with `soundsc`? Why? If you did it right, it should sound like `seaup_bad.wav`. Can you interpret what you hear? It sounds very 'tinny', as if there is a lot of extraneous energy at high frequencies. Figure 7-6 shows what the spectrogram looks like:

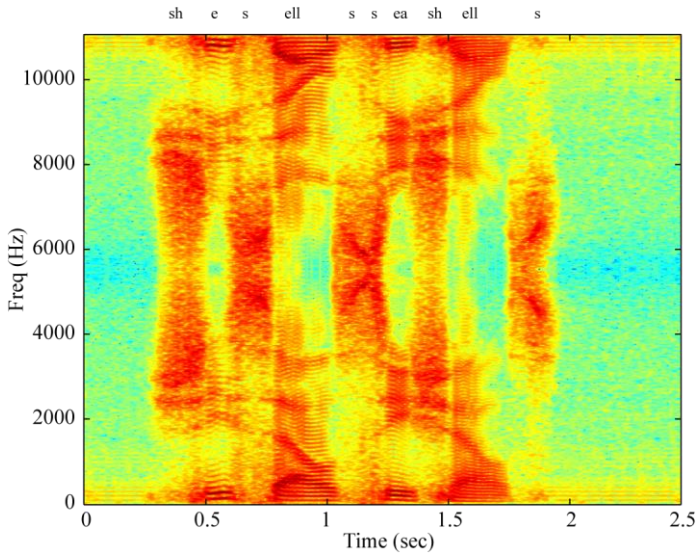

Figure 7-6: Upsampled speech segment before lowpass filtering

- Compare what happens if we properly upsample $x[n]$. In order to do this, use Matlab's `resample` command:

```
>> z = resample(x, 2, 1);
```

If you did it right, it should sound like `seaup.wav`, namely, just like the original wave file, except y twice as long. Figure 7-7 shows what the spectrogram of the upsampled signal looks like after proper lowpass filtering. Note that all the image frequencies have been filtered out.

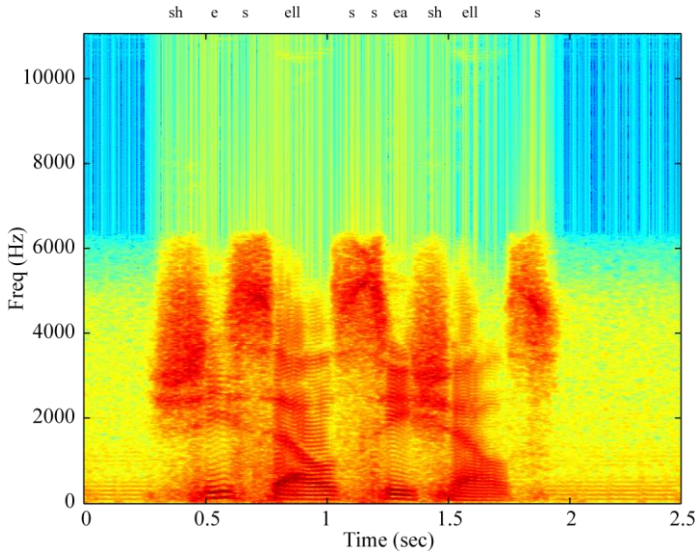

Figure 7-7: Upsampled speech segment after lowpass filtering

Some questions

- Can we use the same kind of Kaiser filter described in Section 7.2.2 for upsampling that we used in downsampling?
- How does the bandwidth of the upsampling filter (i.e., ω_C) depend on U ?
- Why does the spectrogram in Figure 7-6 extend to 11 kHz?
- Why does it appear as if, for the spectrogram in Figure 7-6, the data at frequencies above 5500 Hz is the mirror image of the data below 5500 Hz

7.2.4 Resampling

Now you are asked to resample an input sequence, $x[n]$, by a rational factor U/D. Suppose you obtained $x[n]$ by sampling a continuous-time signal, $x(t)$, at a rate of f_S. If you wish to produce an output sequence, $y[n]$, that is equivalent to having sampled $x(t)$ at a rate of $1.5 f_S = 3 f_s/2$, you could upsample $x[n]$ by a factor of 3 and then downsample the result by a factor of 2 to form $y[n]$. In general, to resample by a rational factor of U/D, we could

1. Upsample $x[n]$ by U, as in Section 7.2.3 (i.e. expand the sequence and lowpass filter it) to form sequence, $w[n]$; then,
2. Downsample $w[n]$ by D, as in 7.2.1 (i.e. lowpass filter the sequence and then downsample), to form sequence $y[n]$.

Some questions

- Why is this procedure wasteful?
- Do you really need two lowpass filtering steps, one in the upsample, one in the downsample?
- Could we first downsample and then upsample? What would the drawbacks of this approach be?

You should understand why we can perform a general resampling by a rational factor of U/D, where U and D are integers, by the following procedure:

1. If $U > 1$, make a new sequence, $x_e[n]$ by inserting $U-1$ zeros after each point in $x[n]$;
2. Filter $x_e[n]$ (or $x[n]$ if $U = 1$) using an FIR lowpass filter of the appropriate bandwidth, ω_C, to form a sequence, $w[n]$. Important question: What is the relation between the required filter bandwidth, ω_C, U and D?
3. If $D > 1$, downsample $w[n]$ by a factor of D to form output sequence, $y[n]$.

7.3 Assignment

1. Answer in writing all the questions in the "Some questions" sections of the Downsampling and Upsampling links, above.
2. Write a Matlab function, named `resamp`, which has the following first lines:

```
function y = resamp(x, r)
% RESAMP Resample an input sequence x by a factor of r
%          to produce an output sequence y by a combination
%          of upsampling and downsampling.
%          For example,
%             y=resamp(x,1.5);
%          will upsample x by 3 and downsample by 2.
```

- The array, `x`, is just a plain-vanilla array, not one of our Matlab sequence structures.
- You may want to look at the Matlab function rat, which tries to express a rational number (such as the resampling parameter, `r`) as the ratio of integers.
- You may use the Matlab functions `fir1`, `filter` and/or `kaiser`, to make your Kaiser filter only.
- Obviously, you may not use the Matlab `resample`, `decimate`, or `interpolate` functions. However, you should feel free to use these functions to check your `resamp` function. You should read about making the Kaiser filter in Section 7.2.2.
- You will want to make sure that the bandwidth and gain of your Kaiser filter are correct, given the value of your resampling parameter, `r`.
- You may use Matlab's `conv` routine if you wish, rather than `filter`, but you may find it adds extra points to the output array.
- Test your `resamp` against Matlab's `resample` function. Since Matlab's `resample` function may use a different lowpass filter, we can't simply compare point by point. However, we can look at the output of the two functions for a simple sinusoidal input to see that it does the right thing. The program `test_resamp.m` is designed to do just that. What `test_resamp` does is to take an input sequence and plot the first few points resampled by a given factor using both Matlab's `resample` function and your `test_resamp` function. To use `test_resamp`, just put it in the directory with your `resamp` function, and run it. For example, to test upsampling of a little sin sequence by a factor of two, just do this:

```
>> x = -sin(2 * pi * (0:10) / 4);
>> test_resamp(x, 2);
```

The result is shown in Figure 7-8. The output of your `resamp` function (in blue) are compared with Matlab's `resample` function (in red). In this case, the perfectly superimpose.

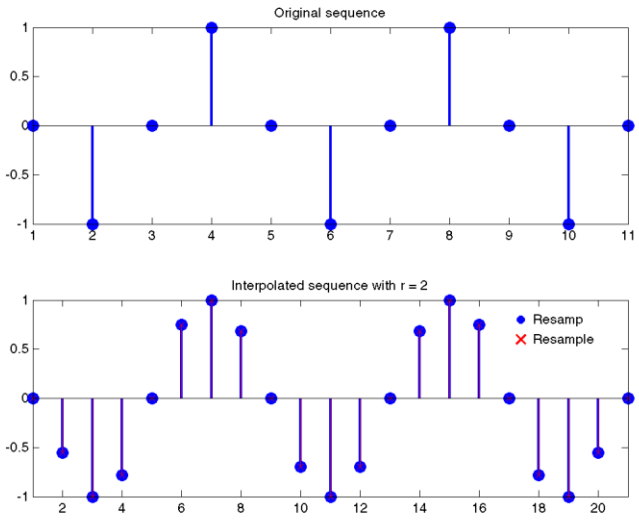

Figure 7-8: Output of test_resamp program

For your assignment, download `lab7_2024.zip`, unpack it the same directory as your `resamp` function and run `lab7_2024`. Publish it to a .pdf file when you're satisfied with the output.