Lab 4 – Sample Rate Conversion

Write a C program to perform rational sample rate conversion.

- The program should accept command line arguments for *U*, *D* the file to be converted, and a filter impulse response file.

```
// Up Sampling

#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define IOBUFFSIZE 2048

typedef struct
{
        int ndim; /* Number of Dimensions; audio=1 */
        int nchan; /* Number of Channels; monaural=1 stereo=2 */
        int dim0; /* Length of First Dimension; audio=L L=length */
        int dim1; /* Length of Second Dimension; audio=Fs Fs=sample rate */
        int dim2; /* Length of Third Dimension; audio=null */
} dsp_file_header;

int main(int argc,char *argv[])
{
        // Open Files
        FILE *fx,*fy,*fh;
        if(NULL == (fx = fopen("input.bin","rb")))
        {
                printf("ERROR: Cannot open input.bin for input.\n");
                return 0;
        }
        if(NULL == (fy = fopen("output.bin","wb")))
        {
                printf("ERROR: Cannot open output.bin for output.\n");
                return 0;
        }
        if(NULL == (fh = fopen("impulse.bin","rb")))
        {
                printf("ERROR: Cannot open impulse file impulse.bin.\n");
                return 0;
        }

        // Read In & Write Out Headers
        int Lx,Lh,Ly,U,L,M;
        dsp_file_header headx,heady,headh;
        fread(&headx,sizeof(dsp_file_header),1,fx); // Read in Input Header
        fread(&headh,sizeof(dsp_file_header),1,fh); // Read in Impulse Header
        Lx = headx.dim0; // Length of Input
        Lh = headh.dim0; // Length of Impulse Response
        U = headh.dim2;         // Up Sample Multiplier
        L = Lh;                         // Length of Impulse Response
        M = ceil(L/U);          // Length of Impulse Response (Modified)
        memcpy(&heady,&headx,sizeof(dsp_file_header)); // Copy Input Header to Output Header
        Ly = heady.dim0; // Length of Output
        fwrite(&heady,sizeof(dsp_file_header),1,fy); // Write out Output Header

    // Error Checking
        if(headx.nchan > 1)
        {
```

```
			printf("ERROR: This program only processes single channel signals.\n");
			return 0;
		}

		// Memory Allocation
		float *h = (float*)calloc(L,sizeof(float));		// Dynamic Array for Impulse Response
		float *g = (float*)calloc(L,sizeof(float));		// Dynamic Array for Circular Buffer
		float x[IOBUFFSIZE], y[IOBUFFSIZE];				// Static Arrays for I/O

		// Read Impulse Response (natural order in file; reverse in memory)
		fread(h,sizeof(float),Lh,fh);

		// Processing
		float t;			// Variable for accumulating convolution result
		int xlen,ylen=0;// Indexes for input and output buffers
		int i;				// Index for input data buffer
		int j;				// Index for up sampling loop
		int k=0;			// Index for circular data buffer
		int m;				// Convolution loop index for filter coefficients
		int n;				// Convolution loop index for circular data buffer
		xlen = fread(x,sizeof(float),IOBUFFSIZE,fx); // Read in first chunk of input samples
		while (xlen > 0)
		{ // While there are samples to be processed, keep processing
			for (i=0; i<xlen; i++)
			{ // Process each of the input samples
				k = (k+M-1) % M;// Update circular index of filter circular data buffer
				g[k] = x[i];		// Put each sample into the filter circular data buffer
				for (j=0; j<U; j++)
				{ // Loop over the up sampled outputs
					for (t=0.0, m=0, n=0; n<M; n++, m+=U)
					{ // Convolution loop
						t += h[m+j]*g[(k+n) % M]; // Multiply and accumulate into local variable
					} // End convolution loop
					y[ylen] = t;	// Save result into output buffer
					ylen++;			// Increment the index for the output buffer
					if (ylen==IOBUFFSIZE)
					{ // If output buffer is full, then save it to output file
						fwrite(y,sizeof(float),ylen,fy); // Write the output buffer
						ylen = 0;		// Reset the inex for the output buffer
					}
				} // End loop over up sampled outputs
			} // End loop over inputs samples
			xlen = fread(x,sizeof(float),IOBUFFSIZE,fx); // Read in next chunk of input samples
		}
		if(ylen>0) // Finish writing the last chunk of output samples
		{ // If output buffer is full, then save it to output file
			fwrite(y,sizeof(float),ylen,fy); // Write the output buffer
			ylen = 0; // Reset the index for the output buffer
		}

		// Numer of I/O samples
		printf("Number input samples = %d\n",Lx);
		printf("Number output samples = %d\n",Ly);

		// Close files and free memory
		fclose(fx);
		fclose(fy);
		fclose(fh);
		free(g);
		free(h);

		system("pause>nul");
		return 1;
}
```

```c
// Down Sampling

#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define IOBUFFSIZE 2048

typedef struct
{
	int ndim; /* Number of Dimensions; audio=1 */
	int nchan; /* Number of Channels; monaural=1 stereo=2 */
	int dim0; /* Length of First Dimension; audio=L L=length */
	int dim1; /* Length of Second Dimension; audio=Fs Fs=sample rate */
	int dim2; /* Length of Third Dimension; audio=null */
} dsp_file_header;

int main(int argc,char *argv[])
{
	// Open Files
	FILE *fx,*fy,*fh;
	if(NULL == (fx = fopen("input.bin","rb")))
	{
		printf("ERROR: Cannot open input.bin for input.\n");
		return 0;
	}
	if(NULL == (fy = fopen("output.bin","wb")))
	{
		printf("ERROR: Cannot open output.bin for output.\n");
		return 0;
	}
	if(NULL == (fh = fopen("impulse.bin","rb")))
	{
		printf("ERROR: Cannot open impulse file impulse.bin.\n");
		return 0;
	}

	// Read In & Write Out Headers
	int Lx,Lh,Ly,D,L;
	dsp_file_header headx,heady,headh;
	fread(&headx,sizeof(dsp_file_header),1,fx); // Read in Input Header
	fread(&headh,sizeof(dsp_file_header),1,fh); // Read in Impulse Header
	Lx = headx.dim0;		// Length of Input
	Lh = headh.dim0;		// Length of Impulse Response
	D = headh.dim2;			// Down Sample Divisor
	L = Lh;					// Length of Impulse Response
	memcpy(&heady,&headx,sizeof(dsp_file_header)); // Copy Input Header to Output Header
	Ly = heady.dim0;		// Length of Output
	fwrite(&heady,sizeof(dsp_file_header),1,fy); // Write out Output Header

	// Error Checking
	if(headx.nchan > 1)
	{
		printf("ERROR: This program only processes single channel signals.\n");
		return 0;
	}

	// Memory Allocation
	float *h = (float*)calloc(L,sizeof(float));	// Dynamic Array for Impulse Response
	float *g = (float*)calloc(L,sizeof(float));	// Dynamic Array for Circular Buffer
	float x[IOBUFFSIZE],y[IOBUFFSIZE];				// Static Arrays for I/O

	// Read Impulse Response (natural order in file; reverse in memory)
```

```c
	fread(h,sizeof(float),Lh,fh);

	// Processing
	float t;		// Variable for accumulating convolution result
	int xlen, ylen=0;	// Indexes for input and output buffers
	int i;		// Index for input data buffer
	int l=1;// Down sampling counter
	int k=0;// Index for circular data buffer
	int n;		// Convolution loop index for filter coefficients and circular data buffer
	xlen = fread(x,sizeof(float),IOBUFFSIZE,fx); // Read in first chunk of input samples
	while(xlen>0)
	{ // While there are samples to be processed, keep processing
		for(i=0; i<xlen; i++)
		{ // Process each of the input samples
			k = (k+L-1) % L;	// Update circular index of filter circular data buffer
			g[k] = x[i];			// Put each sample into the filter circular data buffer
			l = (l+D-1) % D;	// Update downsampling index
			if(l==0)
			{ // Down sampling condition: Only compute convolution result when needed
				l = D;				// Reset the down sampling counter
				for(t=0.0,n=0; n<L; n++)
				{ // Convolution loop
					t += h[n]*g[(k+n) % L]; // Multiply and accumulate into local variable
				} // End convolution loop
				y[ylen] = t;		// Save result into output buffer
				ylen++;				// Increment the index for the output buffer
				if(ylen==IOBUFFSIZE)
				{ // If output buffer is full, then save it to output file
					fwrite(y,sizeof(float),ylen,fy); // Write the output buffer
					ylen = 0;			// Reset the inex for the output buffer
				}
			} // End down sampling condition
		} // End loop over input samples
		xlen = fread(x,sizeof(float),IOBUFFSIZE,fx); // Read in next chunk of input samples
	} // Finish writing the last chunk of output samples
	if(ylen>0)
	{ // If output buffer is full, then save it to output file
		fwrite(y,sizeof(float),ylen,fy); // Write the output buffer
		ylen = 0;		// Reset the index for the output buffer
	}

	// Numer of I/O samples
	printf("Number input samples = %d\n",Lx);
	printf("Number output samples = %d\n",Ly);

	// Close files and free memory
	fclose(fx);
	fclose(fy);
	fclose(fh);
	free(g);
	free(h);

	system("pause>nul");
	return 1;
}
```

Design the filters in MATLAB.

```matlab
U = 0; % Up Sample Multiplier
D = 0; % Down Sample Divisor
N = max([U,D]);
fpass = 0.9/(2*N);
fstop = 1.1/(2*N);
f1 = (fstop + fpass)/2;
f2 = (fstop - fpass)/2;
L = 100;
```

```
n = [-L:L].';
h = (1/N)*sinc(2*f1*n).*sinc(2*f2*n);
win = hamming(2*L+1);
hw = h.*win;

NFFT = 2^14;
freq = [0:NFFT-1]/NFFT;
subplot(211);
plot(freq,20*log10(abs(fft([h hw],NFFT))),'LineWidth',2);
hold on;
ax = axis();
plot(fpass*[1 1],ax(3:4),'r');
plot(1-fpass*[1 1],ax(3:4),'r');
plot(fstop*[1 1],ax(3:4),'r');
plot(1-fstop*[1 1],ax(3:4),'r');
hold off;
grid on;
ylim([-100 10]);
subplot(212);
plot(freq,abs(fft([h hw],NFFT)),'LineWidth',2);
hold on;
ax = axis();
plot(fpass*[1 1],ax(3:4),'r');
plot(1-fpass*[1 1],ax(3:4),'r');
plot(fstop*[1 1],ax(3:4),'r');
plot(1-fstop*[1 1],ax(3:4),'r');
hold off;
grid on;

file_name = sprintf('lpf_U%d_D%d.bin',U,D);
fid = fopen(file_name,'wb');
fwrite(fid,[1 1 length(h) 1 N],'int');
fwrite(fid,h,'float');
fclose(fid);
```

Use the file `galway11_mono_45sec.wav` to perform the following processing steps:

- Up sample by $U = 2$, use $f_{pass} = 0.9/(2U)$ and $f_{stop} = 1.1/(2U)$
  - What are the input and output sample rates?

```
Input Samples, Number of = 496125               Output Samples, Number of = 496125*2 = 992250
Input Length (seconds) = 45                     Output Length (seconds) = 45
Input Sample Rate (sam/s) = 11025               Output Sample Rate (sam/s) = 22050
```

- Down sample by $D = 5$, use $f_{pass} = 0.9/(2D)$ and $f_{stop} = 1.1/(2D)$
  - What are the input and output samples rates?

```
Input Samples, Number of = 496125               Output Samples, Number of = 496125/5 = 99225
Input Length (seconds) = 45                     Output Length (seconds) = 45
Input Sample Rate (sam/s) = 11025               Output Sample Rate (sam/s) = 2205
```

- Perform a $U/D = 2/5$ sample rate conversion
  - What are the input and output sample rates?
  - What f_{pass} and f_{stop} did you use?

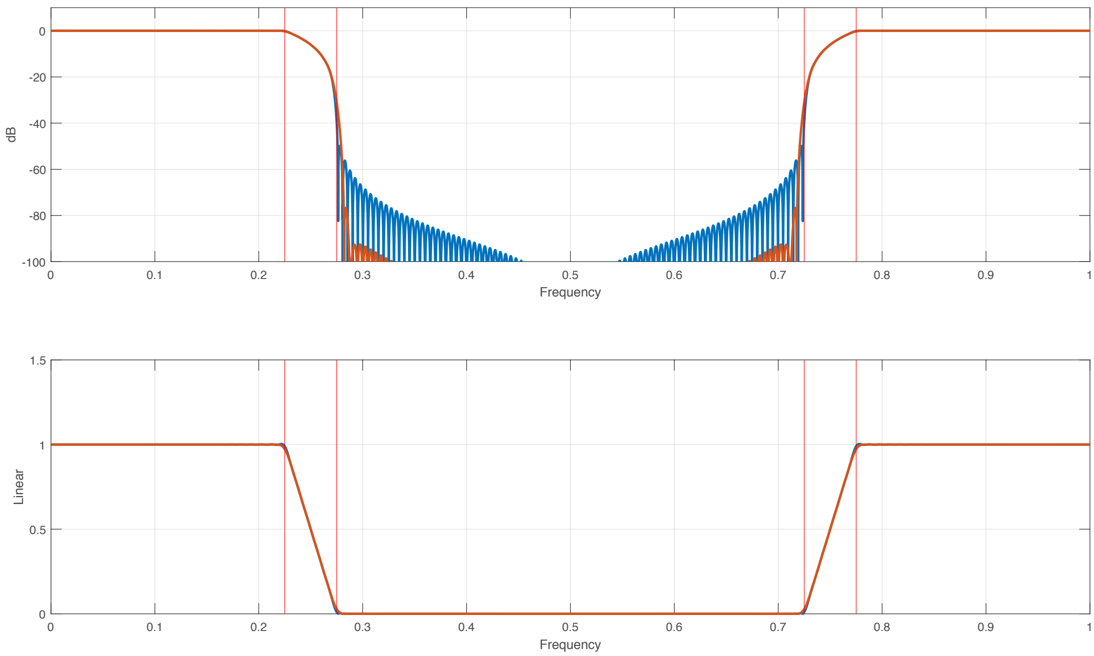
```
Input Samples, Number of = 496125               Output Samples, Number of = (496125*2)/5 = 198450
Input Length (seconds) = 45                     Output Length (seconds) = 45
Input Sample Rate (sam/s) = 11025               Output Sample Rate (sam/s) = 4410
```

$f_{pass} = 0.9/(2D)$ and $f_{stop} = 1.1/(2D)$

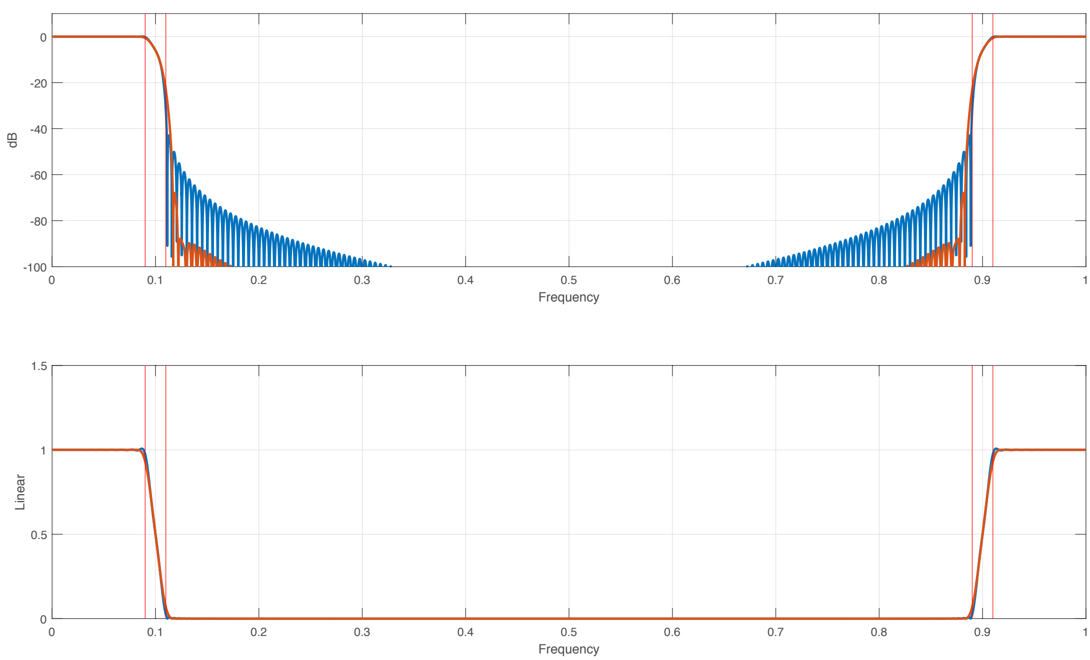
- In each of these cases, choose a sufficiently long filter so that the stop band attenuation is greater than 40 dB.
  - How long was the filter in each case?
  - Plot the magnitude response (both linear and dB scales).

Filter Length = 201

$U = 2$

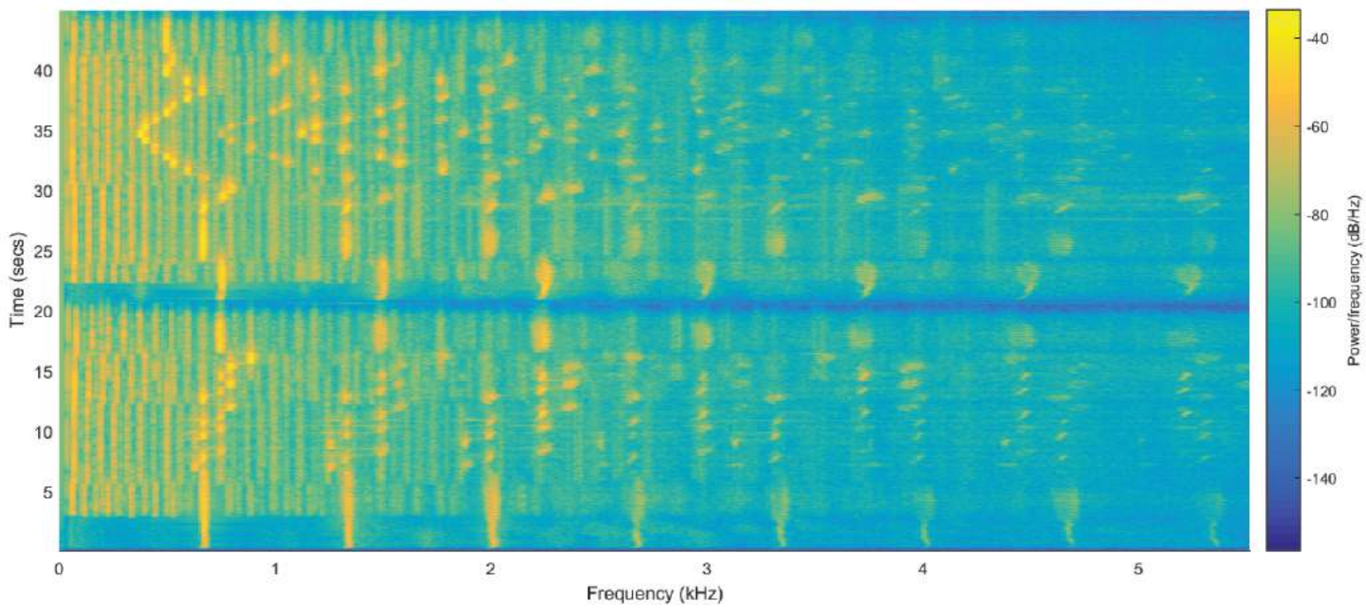


$D = 5$ or $U/D = 2/5$

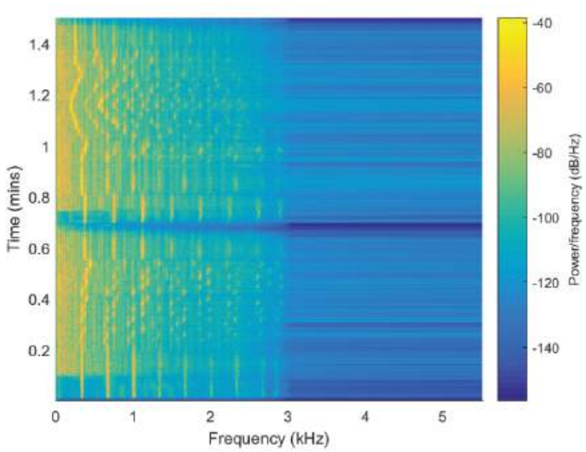


- In each of these cases, plot spectrograms of the signal before and after conversion.
  - Comment on what you see in the output spectrogram and how it can be explained based on the sample rate conversion operation.
  - Compare the input and output spectrograms.
  - Make sure to use the correct sample rates for the spectrograms.

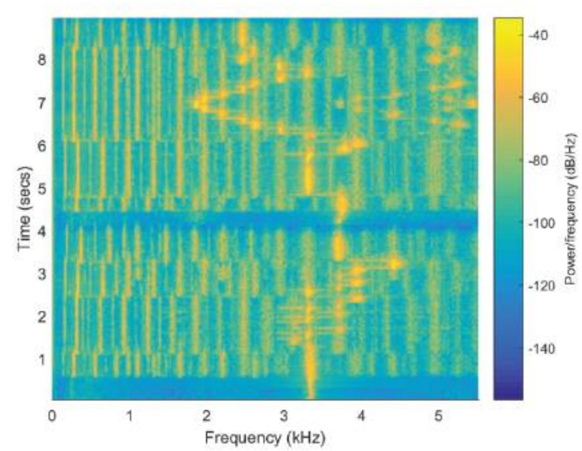
`galway11_mono_45sec.wav`



U = 2

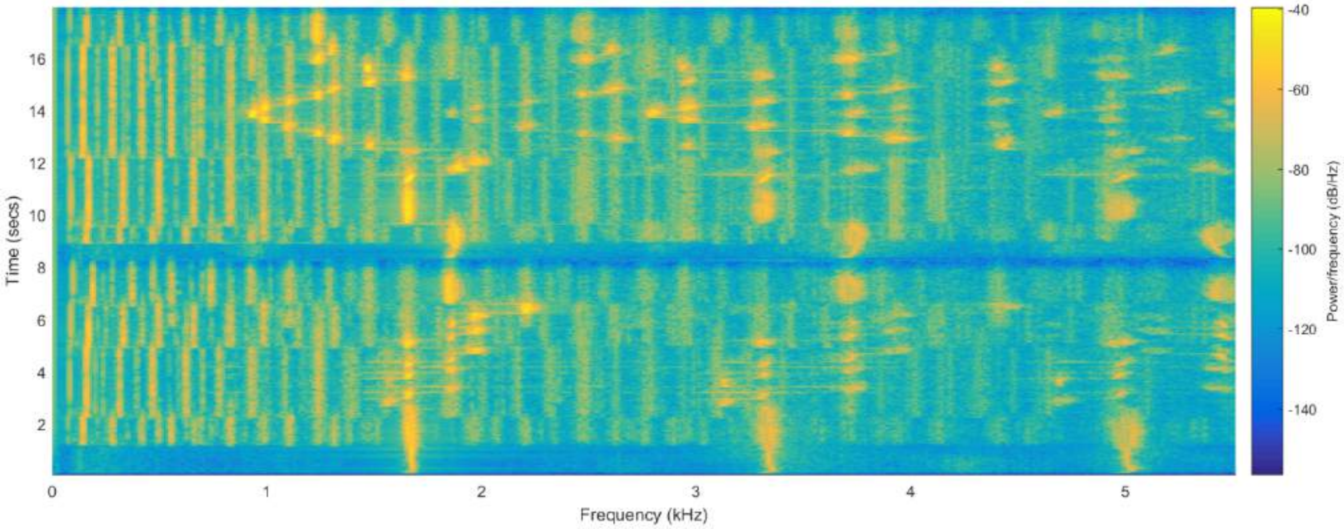


D = 5



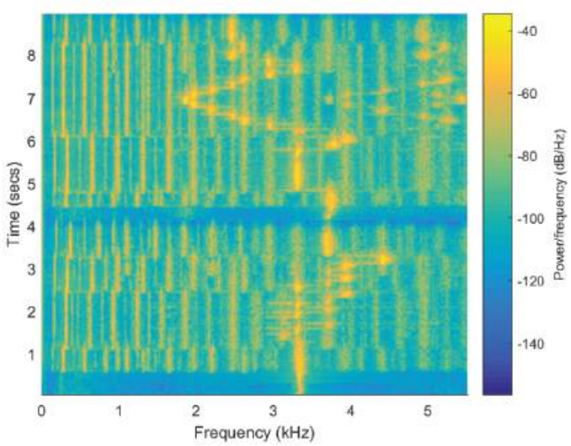
The output spectrograms show that with upsampling, the input was scaled from right to left horizontally. While with downsampling, the input was scaled from left to right horizontally. Thus being consistent with each respective form of resampling where upsampling increases the number of samples while downsampling reduces the number of samples.

U/*D* = 2/5

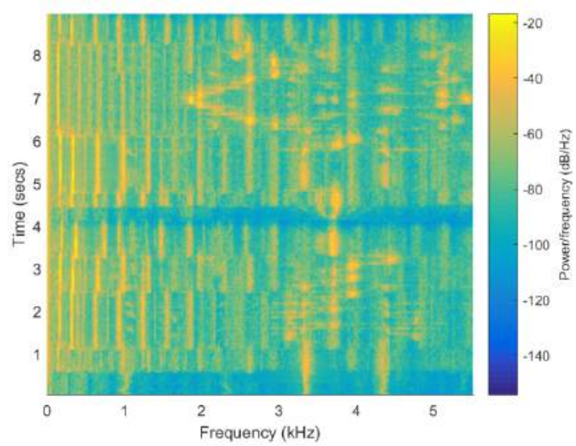


- Down sample the signal by *D* = 5 without any anti-aliasing filtering.
  - Listen to the input and output and compare to the case in which anti-aliasing filtering is used.
  - Comment on what you hear (what does aliasing sound like?)
  - Compare spectrograms of the downsampled signal with and without anti-aliasing filtering.
  - Comment on what you see.

D = 5 with anti-aliasing filter



D = 5 without anti-aliasing filter



After listening to the output with the anti-aliasing filter, the output volume sounded much higher and with much less audible detail than the input.

Comparing the spectrograms, the anti-aliasing filter shows the higher frequencies being filtered out more clearly than without the filter. Visually this is apparent since the spectrogram with the anti-aliasing filter shows more blue color than the spectrogram without the anti-aliasing filter.