Shifting the Pitch of Audio in the Frequency Domain

Introduction

This application will shift the pitch of an audio file by scaling its frequency spectrum.

Our simple approach will multiply each frequency by the same amount. For a piano or violin note, the harmonics will still be an integer multiple of the pitch with the same balance between amplitudes (baring some digital aliasing effects). This preserves the *timbre* of the sound.

In more detail:

- A small WAV file is imported (by default, a violin note played at B5 with vibrato).
- The audio is converted to the frequency domain. Each spectral point corresponds to a frequency of 0 Hz to fs/2 Hz in in steps of fs/N (where fs is the sampling rate and N is the number of samples).
- A new vector of frequencies is generated by multiplying the original frequencies by $2^{n\_semitones/12}$ (where n_semitones is the number of semitones, and can be positive or negative)
- New spectral data is generated by interpolating the original frequencies and spectral points against the new frequencies.
- The new spectral data is inverted to the time domain, and converted to audio.

If your original audio is a piano note at F4 (a pitch of 349.2 Hz), you can convert it to an A4 note by scaling the frequencies by 4 semitones (a pitch of $349.2\ \text{Hz}^{4/12} = 440.0$ Hz).

The pitch-shifted audio requires data that nearly always falls between the bins of the unaltered spectral data, but is approximated with linear interpolation. This slightly distorts the pitch-shifted audio.

More complex methods of pitch shifting use overlapping slices of audio analyzed with short time Fourier transforms (as employed in phase vocoders).

This loudspeaker component is needed to play the audio files

Load Packages

```
> restart:
  with(AudioTools):
  with(SignalProcessing):
  with(ArrayTools):
  with(CurveFitting):
```

Import and Prepare the Original Audio

By default, the application analyzes one of the standard WAV file distributed with Maple - a violin note played at B5 with vibrato.

```
> original_audio := Read( FileTools:-JoinPath( [ kernelopts( datadir
  ), "audio", "ViolinThreePosVibrato.wav" ] ) )
```

$$original\_audio := \begin{bmatrix} \text{"Sample Rate"} & 44100 \\ \text{"File Format"} & PCM \\ \text{"File Bit Depth"} & 16 \\ \text{"Channels"} & 2 \\ \text{"Samples/Channel"} & 64724 \\ \text{"Duration"} & 1.46766\,s \end{bmatrix} \quad (3.1)$$

Sampling rate

```
> fs := attributes( original_audio )[ 1 ];
```

$$fs := 44100 \quad (3.2)$$

If the audio has two channels, only analyze the first channel

```
> if numelems( Dimensions( original_audio ) ) = 2 then
     original_audio := original_audio[ .., 1 ];
  end if
```

$$original\_audio := \begin{bmatrix} \text{"Sample Rate"} & 44100 \\ \text{"File Format"} & PCM \\ \text{"File Bit Depth"} & 16 \\ \text{"Channels"} & 1 \\ \text{"Samples/Channel"} & 64724 \\ \text{"Duration"} & 1.46766\,s \end{bmatrix} \quad (3.3)$$

Number of samples

```
> N := numelems( original_audio )
```

$$N := 64724 \quad (3.4)$$

Truncate the audio file to an even number of samples (this makes the code simpler, and we won't

notice a missing sample)

```
> if N mod 2 <> 0 then
     original_audio := original_audio[ .. -2 ]:
     N := N - 1:
  end if:
```

Play the audio file

```
> Play(original_audio)
```

Shift the Pitch in the Frequency Domain

Transform to the frequency domain

```
> data_fft := Vector[ column ]( FFT( original_audio ) );
```

Warning, size of Array must be a power of two greater than two, using DFT instead and suppressing this warning for the remainder of this session

$$data\_fft := \begin{bmatrix} -0.00987586383130373 - 5.182164244728210^{-18}I \\ 0.000625781743063166 - 0.000473814805431979I \\ 0.000579352433631191 + 0.0000261065024135195I \\ 0.000730737434269617 - 0.00239721405215204I \\ 0.00123029748665954 - 0.00101121175200478I \\ 0.00131475034878267 + 0.00182985380240968I \\ 0.000844604211737920 + 0.000807019515718205I \\ -0.00238169344347167 + 0.00459234344714289I \\ 0.00315195578329923 - 0.00110063506315321I \\ -0.00225110137443608 - 0.00147775225177824I \\ \vdots \end{bmatrix} \quad \text{(4.1)}$$

64724 element Vector[column]

Extract the first and second half of the frequency spectrum. The second half is a mirror image of the first half, and is reversed so frequencies increase in the same direction as the first half.

```
> first_half_spectrum  := data_fft[ 1 .. N / 2 ]:
  second_half_spectrum := Reverse( data_fft[ N / 2 + 1 .. -1 ] ):
```

Frequencies in the spectral data of the original audio

```
> freq_orig := Vector( [ seq( i * fs / N, i = 1 .. N / 2) ],
  datatype = float[ 8 ] );
```

$$freq\_orig := \begin{bmatrix} 0.681354675236388 \\ 1.36270935047278 \\ 2.04406402570917 \\ 2.72541870094555 \\ 3.40677337618194 \\ 4.08812805141833 \\ 4.76948272665472 \\ 5.45083740189111 \\ 6.13219207712750 \\ 6.81354675236388 \\ \vdots \end{bmatrix} \quad \textbf{(4.2)}$$

32362 element Vector[column]

Let's now scale the frequencies up or down by a number of semitones. The first step is to generate a new vector of scaled frequencies.

n_semitones is positive to increase pitch, and negative to decrease pitch (an octave is 12 semitones, and is a doubling or halving of the frequency). Increasing the pitch by 2 semitones will change the B5 violin note to a C#6.

```
> n_semitones := 2:
  freq_scaled := freq_orig * 2. ^ ( -n_semitones / 12 );
```

$$freq\_scaled := \begin{bmatrix} 0.607018006739540 \\ 1.21403601347908 \\ 1.82105402021862 \\ 2.42807202695816 \\ 3.03509003369770 \\ 3.64210804043724 \\ 4.24912604717678 \\ 4.85614405391632 \\ 5.46316206065586 \\ 6.07018006739540 \\ \vdots \end{bmatrix} \quad \textbf{(4.3)}$$

32362 element Vector[column]

Now make a vector of new spectral data that contains interpolated values of the original spectral data.

Specifically, we need to interpolate the value of [freq_orig, data_fft] at

- freq_shift[1] to find the spectral value at the 1st index
- freq_shift[2] to find the spectral value at the 2nd index
- etc

```
> first_half_spectrum_scaled  :=  ArrayInterpolation( freq_orig,
  simplify( Re~( first_half_spectrum  ) ), freq_scaled, method =
  linear ) + I * ArrayInterpolation( freq_orig, simplify( Im~(
  first_half_spectrum  ) ), freq_scaled, method = linear ):

  second_half_spectrum_scaled :=  ArrayInterpolation( freq_orig,
  simplify( Re~( second_half_spectrum ) ), freq_scaled, method =
  linear ) + I * ArrayInterpolation( freq_orig, simplify( Im~(
  second_half_spectrum ) ), freq_scaled, method = linear ):
```

Combine the two halves of the scaled spectral data

```
> pitch_shifted_spectrum := Concatenate(1,
  first_half_spectrum_scaled, Reverse( second_half_spectrum_scaled )
  ):
```

Compare the Original and Pitch-Shifted Audio

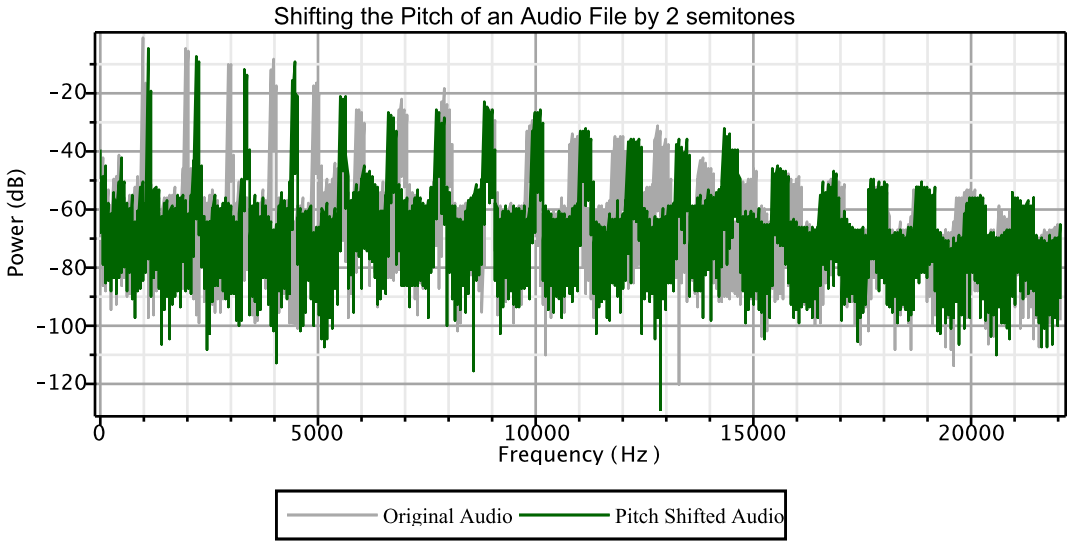
Now convert the interpolated spectral data back to the time domain and create an audio file

```
> pitch_shifted_time_domain := Re~( InverseFFT(
  pitch_shifted_spectrum ) ):
  pitch_shifted_audio := Create( pitch_shifted_time_domain, rate =
  fs )
```

$$pitch\_shifted\_audio := \begin{bmatrix} \text{"Sample Rate"} & 44100 \\ \text{"File Format"} & PCM \\ \text{"File Bit Depth"} & 16 \\ \text{"Channels"} & 1 \\ \text{"Samples/Channel"} & 64724 \\ \text{"Duration"} & 1.46766\,s \end{bmatrix} \quad \textbf{(5.1)}$$

Periodograms of the original and pitch-shifted audio

```
> plots:-display(
   Periodogram( original_audio,      color = "DarkGrey", thickness =
  0, legend = "Original Audio" )
  ,Periodogram( pitch_shifted_audio, color = "DarkGreen",  thickness
  = 0, legend = "Pitch Shifted Audio" )
  ,title = cat( "Shifting the Pitch of an Audio File by ",
  n_semitones, " semitones"), titlefont = [ Arial, 14 ], legendstyle
  = [ font = [ Arial ] ]
  ,size = [800, 400] )
```



Play the original audio file again

```
> Play( Normalize( original_audio, offset = remove, amplitude = 0.99
  ) )
```

Now play the pitch shifted audio

```
> Play( Normalize( pitch_shifted_audio, amplitude = 0.99 ) )
```

For the violin note analyzed by this application, you can even create a duet of violins by playing the original and pitch-shifted audio in either channel of a stereo sound file.

```
> duet := Create( << original_audio, pitch_shifted_audio >> ^ %T,
  rate = fs)
```

$$duet := \begin{bmatrix} \text{"Sample Rate"} & 44100 \\ \text{"File Format"} & PCM \\ \text{"File Bit Depth"} & 16 \\ \text{"Channels"} & 2 \\ \text{"Samples/Channel"} & 64724 \\ \text{"Duration"} & 1.46766\,s \end{bmatrix} \quad \textbf{(5.2)}$$

```
> Play( Normalize( duet, amplitude = 0.99 ) )
```