A Mathematical Approach to Digital Sound Production, and Musical Interpolation

Our Focus

- Use of DFT for
  - 1) Re-sampling/scaling of audio tracks
  - 2) Interpolation of musical notes
  - 3) Graphical Equalizer
- Generation of synthetic audio data
- Digital audio mixing

Sound

- Properties of sound waves:

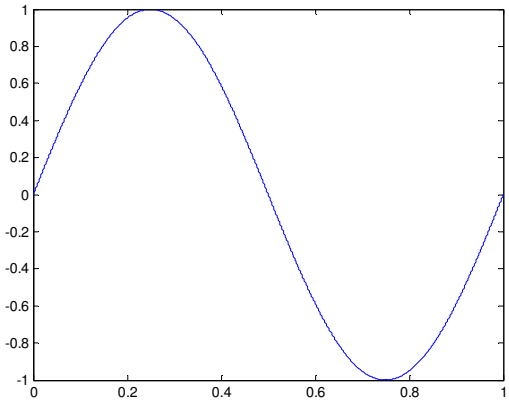
- Frequency, wavelength, period, and amplitude
- $y(t) = a(t)\sin(f(t)\, x(t))$
  - $a(t) \in [0,1]$
  - f(t) = frequency Hz = song(t)
  - x(t) = frequency rads/sec = $2\pi t$
- Superposition
  - Let $w_i(t)$ be a sequence of n wave functions, then

$s_n(t) = w_1(t) + w_2(t) + \ldots + w_n(t)$ is the resulting wave when all n waves occur at the same time.

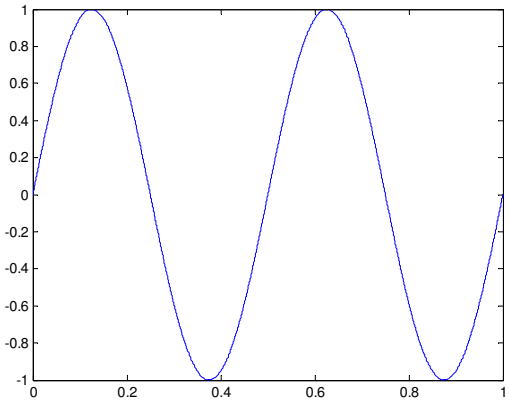
Superposition

- Two waveforms and the resulting superposition

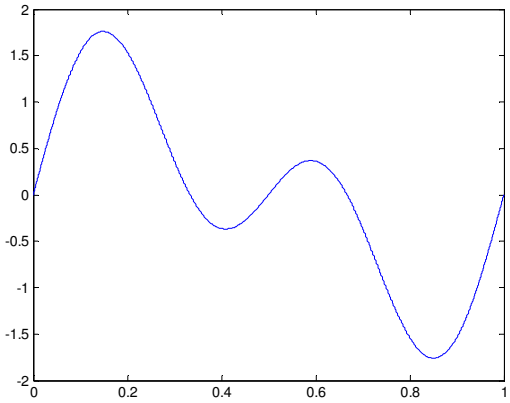
w_1 : 1 Hz



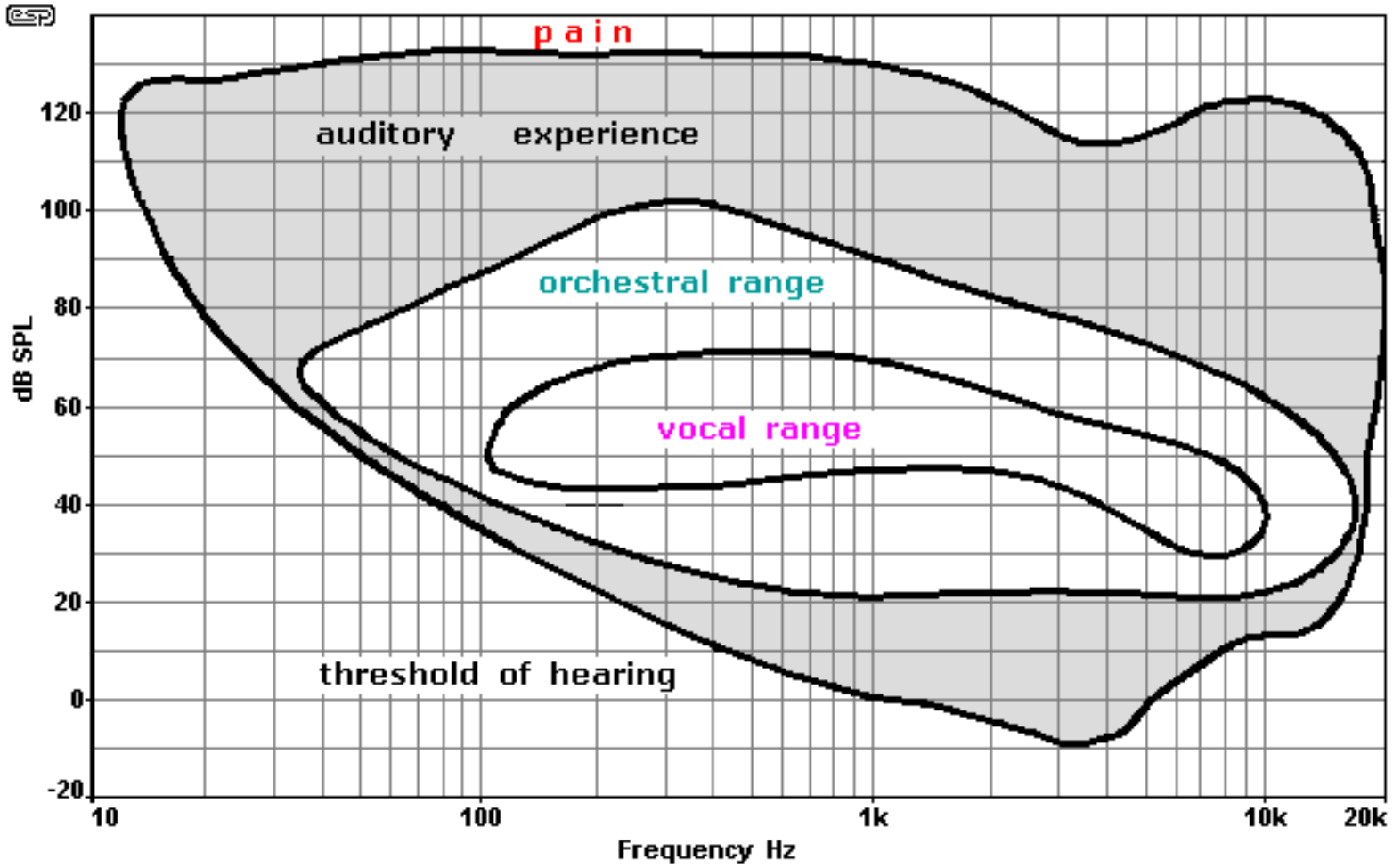
w_2 : 2 Hz



$s_2 = w_1 + w_2$

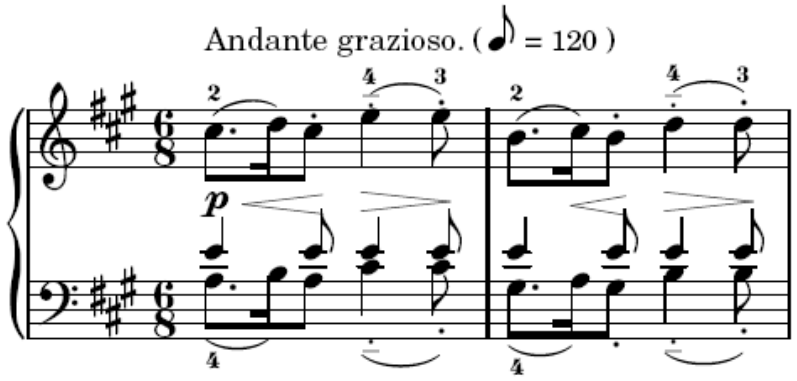


Human Hearing



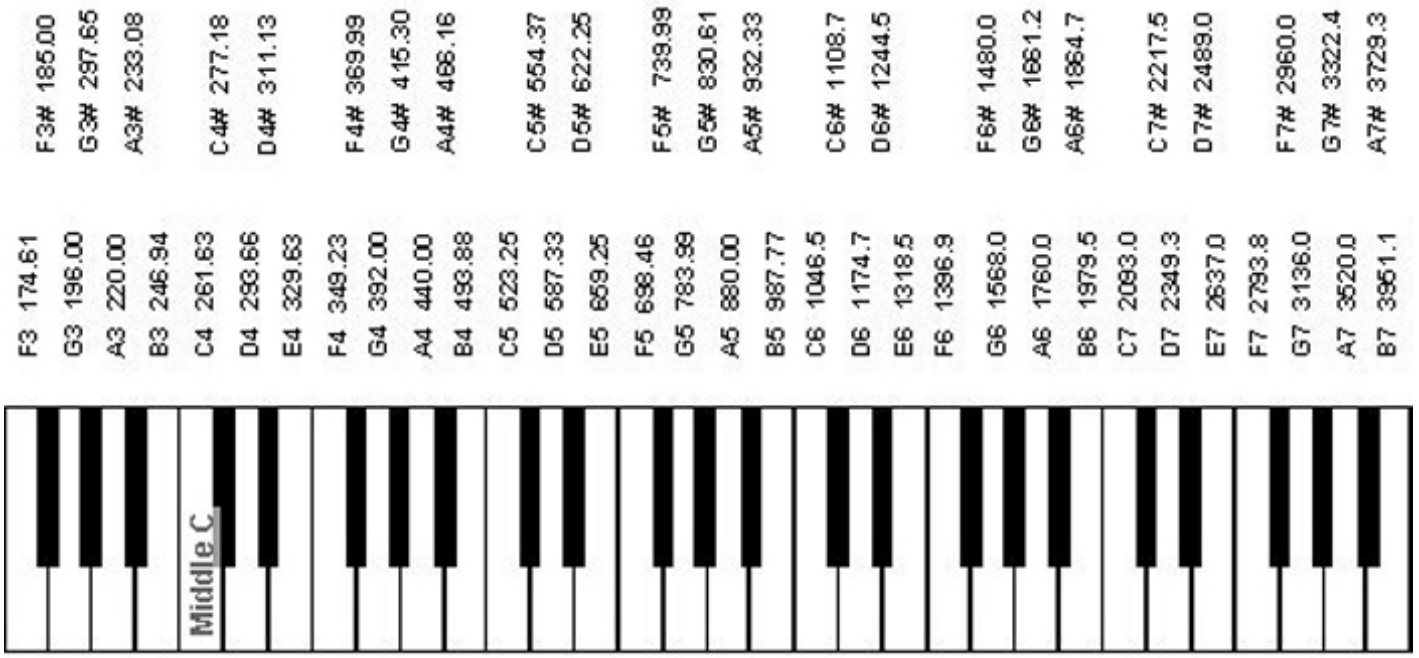
Music

- Duration of note determined by time signature, tempo, note type
- Time signature: *N*/*D*
  - There are *N* notes of type *D* per bar
  - *D* is a power of two
- Tempo measured in beats per minute or BPM.



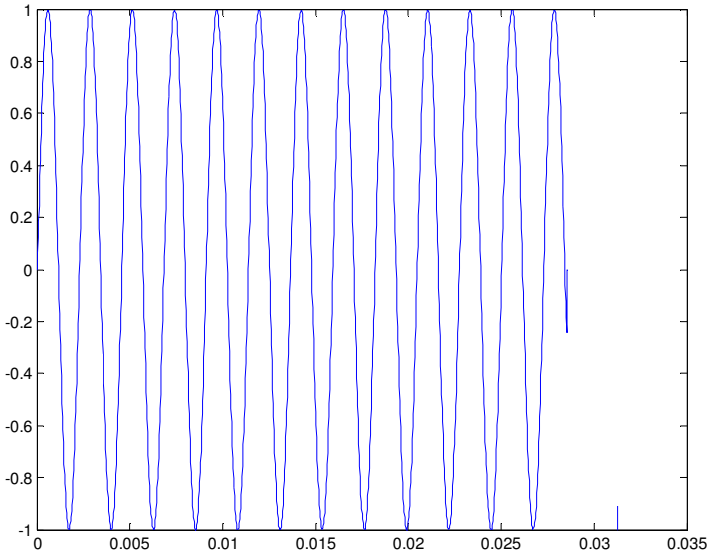
Piano Note Frequencies

- Unit of frequency: hertz (Hz)
- Base unit of hertz: 2pi rad/s, one complete cycle in one second
- E.g. the A4 note is 440 Hz or 440*2*pi rad/s = 440 sine periods



Note Frequencies

- Each note is approximated by a sine wave
- Piano frequency range of 27.5 Hz to 4186 Hz
- Humans generally able to hear between 20 Hz and 20,000 Hz



0.03125 seconds of a 440 Hz wave, an idealized A4 note

Musical Note Durations

- If a given musical piece is in time signature N/D and at a tempo of B beats per min. then one D note lasts $60/B$ seconds.
  - N has little or no bearing on what we are doing here
- Since notes are powers of 2, the above information is enough to determine the duration of all other note types.
  - Suppose $D = 4$ (a quarter note) and $B = 60$ BPM; then a quarter note lasts 1 second, a half note lasts 2 seconds and an eighth note lasts .5 seconds, and so on.

Digital Sound

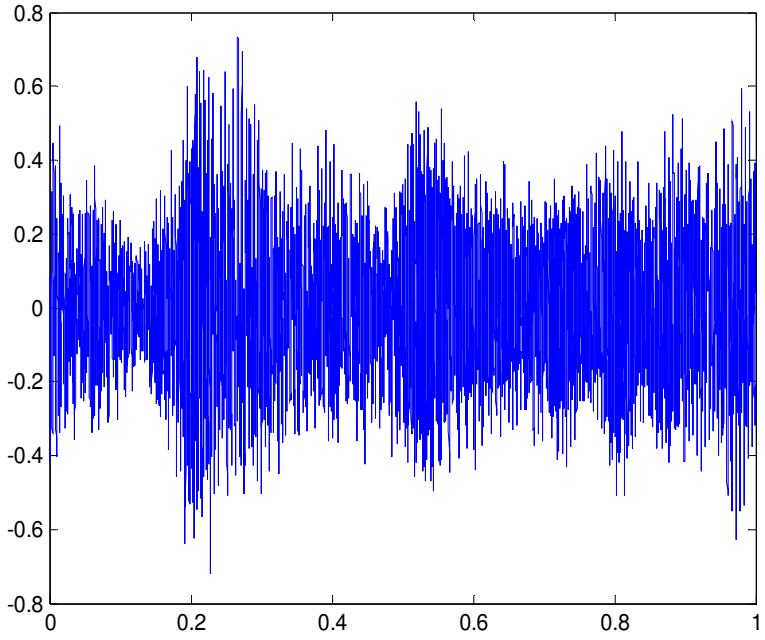
- Analog to digital conversion takes place at a certain "sampling rate". CD audio uses a rate of 44.1 kHz for instance.
  - The number of samples per second defines time increments (Δt)
- Each sample has a certain level of accuracy given by the number of bits per sample (16 for CD audio).
  - The number of bits per sample define amplitude increments (Δy)
  - For a k bit sample, the possible numeric range is 0 to 2^k-1 but the range is half devoted to negative amplitudes.

Digital Sound Amplitude

- In analog systems, amplitude is measured in dB, but in digital systems, the amplitude is a percentage of the maximum for the device
  - In analog systems recordings of amplitudes in excess of the input device's capabilities manifest as distortion in the recording.
  - In digital systems, excess amplitude leads to "clipping"
- Clipping means the amplitude went beyond the representable range and is left at its maximum or min value depending on where it went out of range.
  - For instance, if one continues to scale a digital representation of a sine wave eventually the result is a square wave.

Digital Sound

- In matlab sound can be played with the command:
  - `sound(vec, r, b)`
  - takes *vec* a vector of samples (each in the range [-1,1]
  - the sample rate *r*
  - bit resolution *b*
  - For CD quality *r*=44.1 kHz and b = 16.
- A 1 second sound at rate *r* is represented by a vector with length *r*



1 second of Vivaldi at 44 kHz, 16bit:
`sound(vivaldi,44*1024,16)`

Interpolating Digital Sound

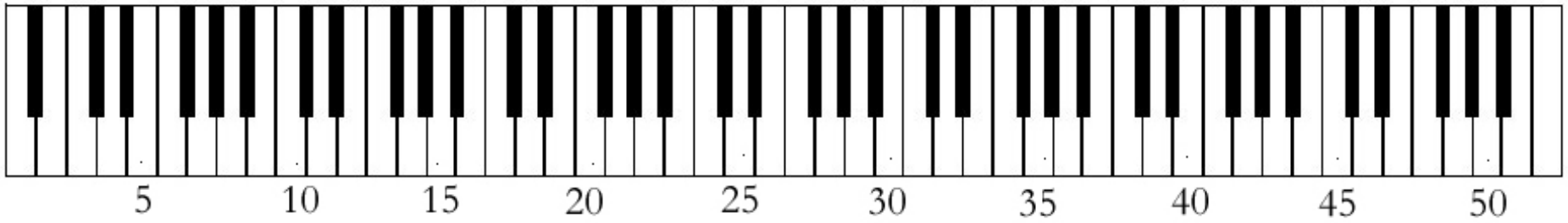
- Why do it
  - Record or produce sound at sample rate *r* to be played on audio systems of sample rate *kr* where *k* normally a power of 2 or 1/(power of two).
  - Slow down or speed up an audio sample
    - E.g. Re-sample a recording of an A4 piano note to be any other note
      - certain error introduced here
      - The length of the new note is shorter or longer
- How to do it:
  - Interpolation with DFT
    - Re-sampling an audio file from rate *r* to 2*r:*
      `dftinterp(audio_file,2*r)↰`
    - Re-sample note A4 to note A5:
      `dftinterp(note_a4,r/2)↰`
      (done with modified dftinterp)

Musical Interpolation

- Let $sg = [n_1\ n_2\ \ldots\ n_k]$ be a k note song at some tempo and time signature and let each n_i *be a* D note.
- Each n_i is a natural number in the range of 1 to 50, mapping to a scale on a standard piano. Let the notes be in some key, say C major.
- Using `round(dftinterp(`*sg*`,2*`*k*`))` we get a new song with twice as many notes and the new notes all fall between the original notes
- Let the notes in the new song be $2D$ notes then the two songs have the same duration.

Musical Interpolation

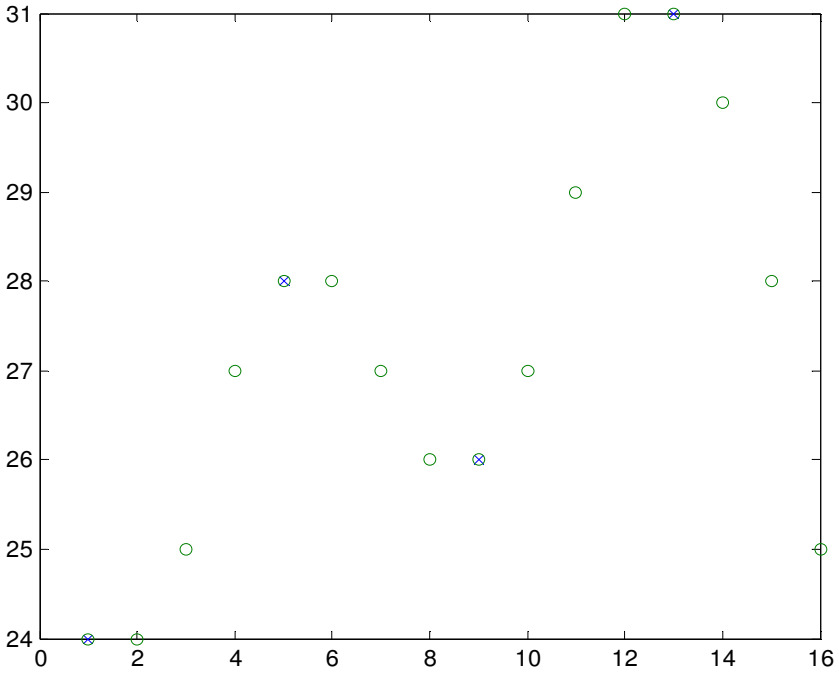
- The standard piano has 88 keys.
- All the white keys are in the scale of C major.
- Map a song in C major from piano index to C major scale index



Above, indexes into the C major scale

Musical Interpolation

- Example using *sg* = [24 28 26 31]
  - `scale_song(sg,2)` (make twice as many notes)
  - In the graph nodes with 'x' are original notes, 'o' are added notes



Here a 1 bar song (in 4/4) is expanded to 4 bars OR could be interpreted as a one bar accompaniment.

Notice that the 'x' nodes also have circles around them

Audio Production

- Goals of audio production:
  - Mix multiple audio tracks into one "final" track
  - Control properties of each audio track so that the amplitudes of selected frequency ranges can be controlled
  - Maximize amplitudes
  - Tweak amplitudes of all tracks so that no clipping occurs in the final mix

Audio Mixing

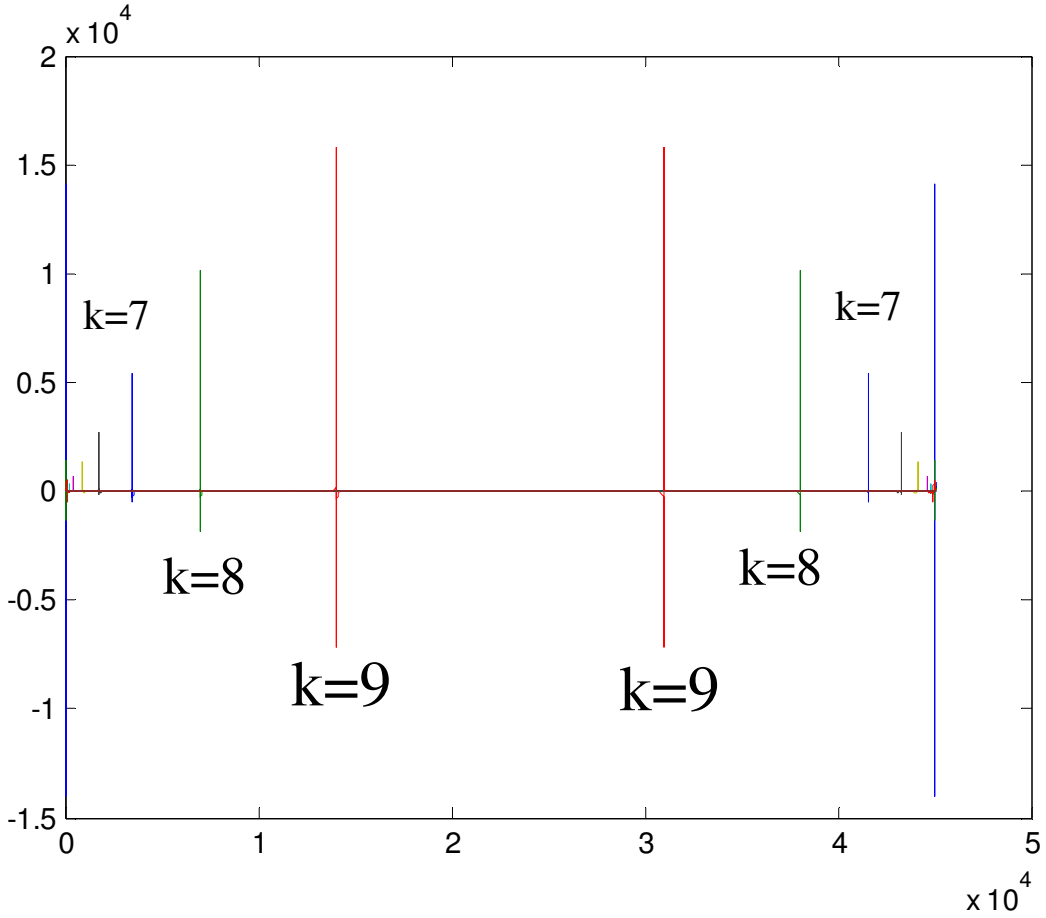
- Using the principle of superposition, audio tracks can be added to create a new track wherein both original sounds can be heard
- In order to avoid clipping tracks can be averaged
  - E.g. $s = (s_1 + s_2)/2$
- If some tracks should be louder than others, a weighted average can be used:
  - E.g. $s = .3s_1 + .7s_2$
- Any track can be made louder or softer by scaling by a constant

Audio Frequency Equalization

- Most people are familiar with the idea of increasing the bass and treble of audio
- Bass is a general term referring to the low frequency portion of the audio, treble refers to the high frequencies
- For high quality audio production, use higher granularity; divide the frequency range into *n* ranges or “bands”.
- To edit the frequencies directly use a DCT on the audio data. The elements in the leftmost part of the DCT are low and high in the middle

Audio Frequency Equalization

- A graph of reference frequencies from $27.5*2^k$ Hz for k = 0 to 9



Audio Frequency Equalization

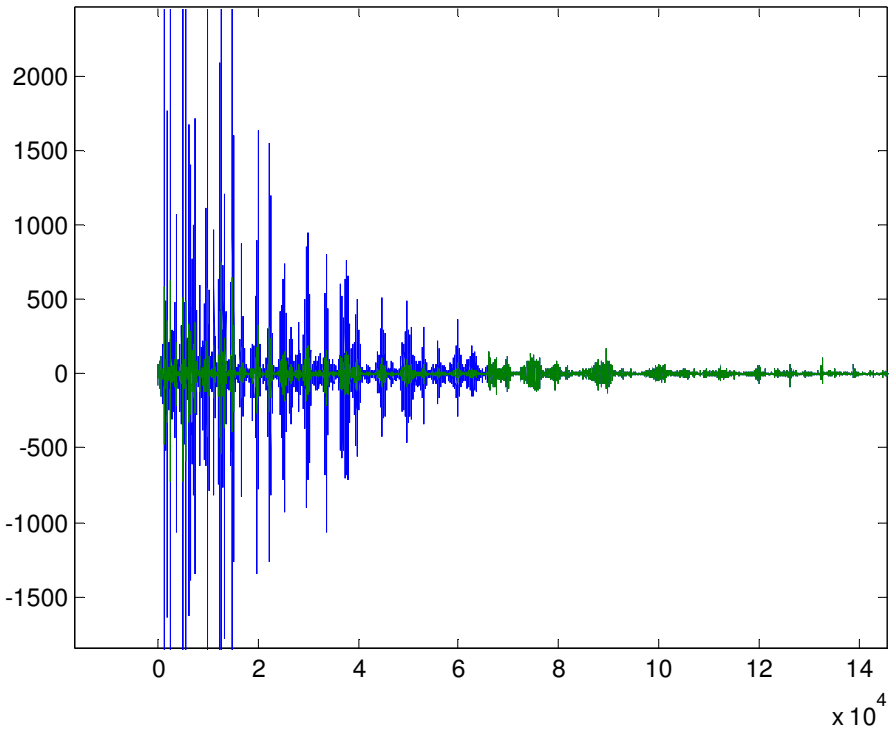
- The graph of reference frequencies was created by making sound files of sine waves at the frequencies 27.5 to $27.5*2^9$ Hz
- The low frequencies appear closest to the right and left edges of the graph and the high frequencies appear closer to the middle

  [Jesse Hansen, et al, University of Rhode Island]
- The graph shows redundant frequency information mirrored across the middle of the graph
- The height of the spike shows the amplitude of that frequency

Audio Frequency Equalization

- To equalize the sound in a slightly naive way we can scale the areas of the DCT down or up based on percentages into the graph.
- To equalize the lowest frequencies we might choose from .1% to 10% into the graph (doing the same for the mirror image on the other side).
- This is "naive" because it doesn't let us select frequency in Hertz, although some further calculations could be done to figure out with percentages map to which frequency ranges.

One half of the DFT graph for Vivaldi



Original frequency graph in blue

Altered frequency graph in green

Audio Frequency Equalization

- `vivaldi` is a high quality audio sample of a few seconds of Vivaldi's Four Seasons, recorded in DDD, uncompressed
- To create the graph in the previous slide:

```
vivaldi_eq =
multibandEQ(vivaldi, [.001 .1   .9 (1-.001)], [.2 .2], 44*1024);
```

- The second two arguments are vectors
  - The first vector specifies which percentage ranges to work on
  - The second specifies the scaling factor to apply
    - A factor of 1 in the second vector will have no effect on the DFT
- In the second vector we set the EQ to work on .1% to 10% of the range and 90% to 99% (the redundant area)
- The third vector contains scaling factors which associate directly with the ranges, in this case reducing the ranges to 20% of their value
- To make the EQ even nicer we could use an arbitrary discrete graph as a diagonal matrix to multiply the DCF by. The way we have it, the graph we use is a sequence of constant function.