CS 583– Computational Audio -- Fall, 2021

Wayne Snyder
Computer Science Department
Boston University

Lecture 11

DFT concluded:

Review of characteristics of the DFT

Interpreting spectra

Extracting musical information from spectra

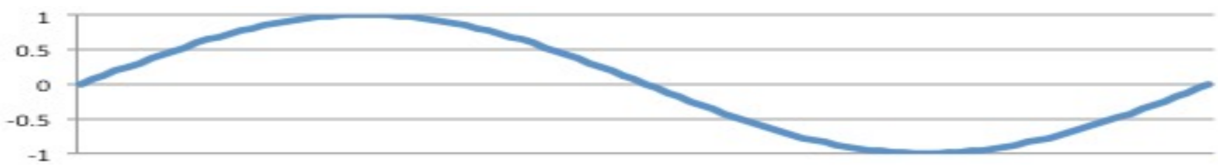
Inverse FFT

Fourier Transform Pairs

Computer Science

Recall: There is a tradeoff between

Temporal Resolution – What is the shortest musical event we can observe?
Spectral Resolution – How many frequencies can we measure?

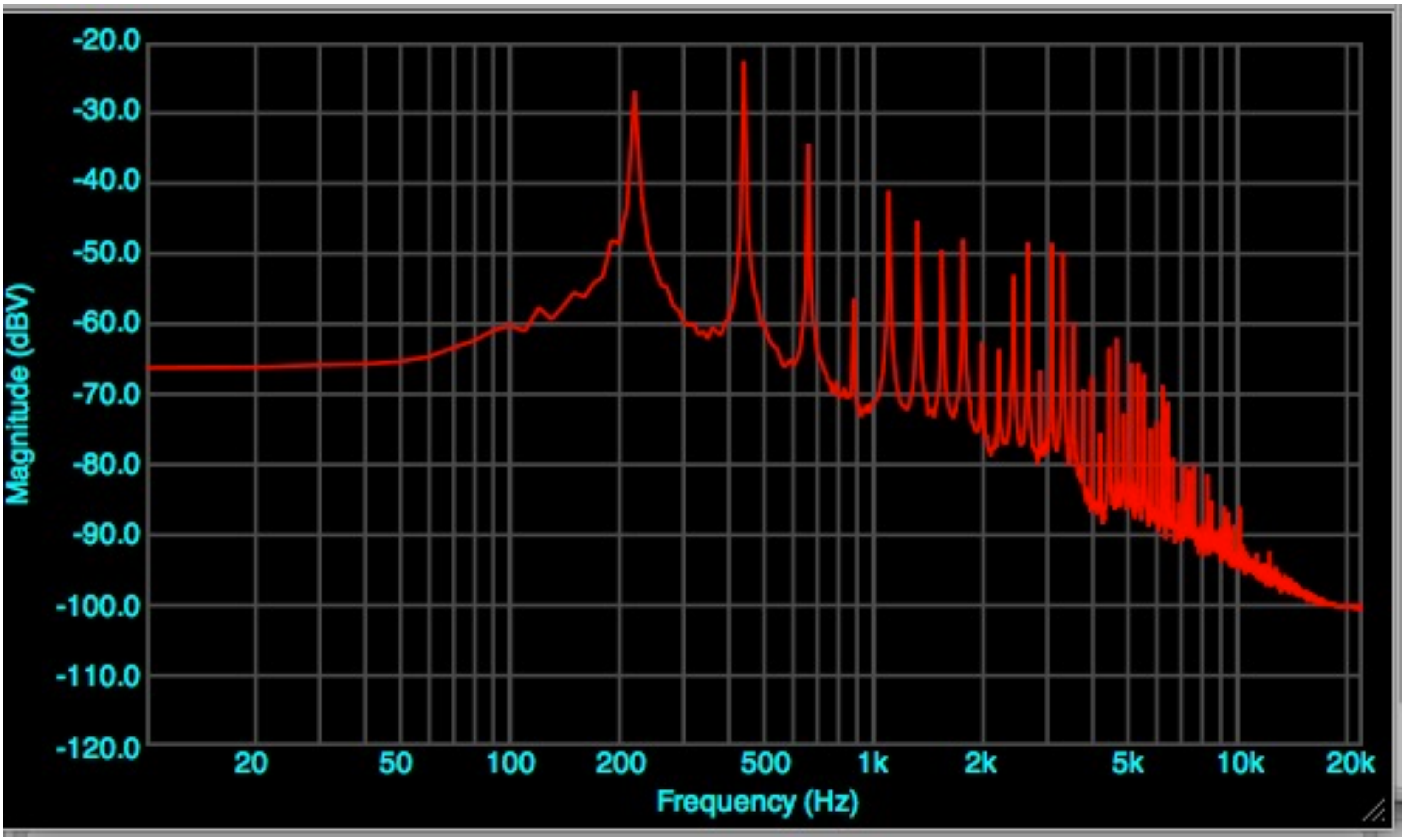


← Window of W Samples →

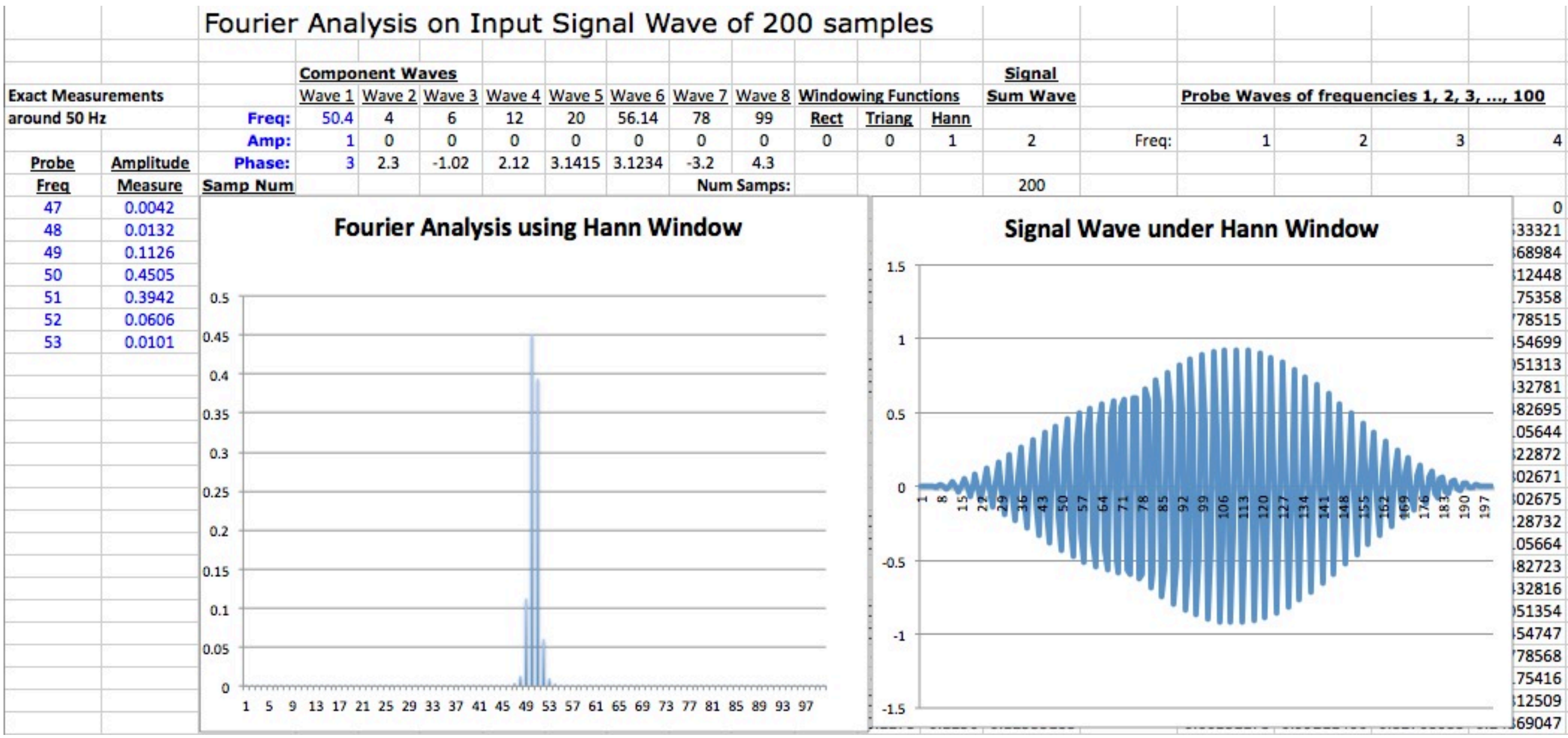
But then temporal and frequency resolution are in an inverse relationship:

W	Time Resolution	Frequency Resolution
64	0.0029	344.5312
128	0.0058	172.2656
256	0.0116	86.1328
512	0.0232	43.0664
1024	0.0464	21.5332
2048	0.0929	10.7666
4096	0.1858	5.3833
8192	0.3715	2.6917

But it is not clear that the only frequencies are multiples of the fundamental, and each “peak” is not a simple value, but a ”triangular mountain”:



Let's try two of these in our experiment on a non-integral frequency of 50.4 Hz, using the Rectangular (as before), the Triangular, and the Hann Windows:



Conclusions on windowing for the DFT:

(1) Window size determines frequency resolution: given a window size of W samples, with a fundamental frequency of f = SR / W, we can only probe for the integral frequencies (the harmonics of F):

0, f, 2*f, 3*f, …., k*f, ….., Nyquist Limit

Any other frequencies will be subject to the "picket fence" problem and only approximated.

(2) Non-integral frequencies cause "leakage" to adjacent integral frequencies; good windowing functions (e.g., Hann) mitigate leakage effects and provide reasonably accurate measurements of amplitude of components, after correction.

Understanding Spectra:

Things to keep in mind when interpreting scales in spectra (y-axis in instantaneous spectrum, z-axis (color) in spectrum)

- The squared spectrum ("power spectrum") corresponds more closely to human perception and is generally preferred; note that log scale obscures the difference between these two (since $\log(x^2) = 2 * \log(x)$)

Understanding Spectra:

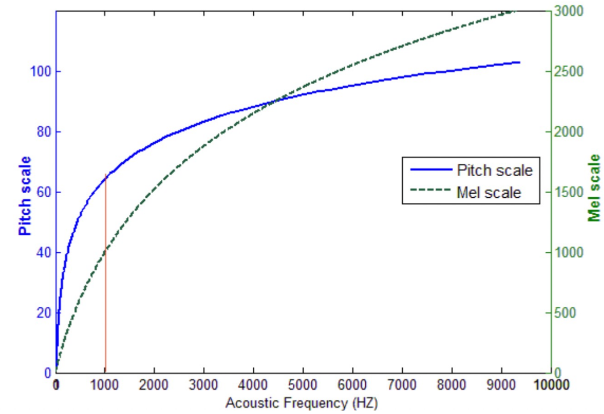
Things to keep in mind when interpreting the scales in spectra:

- Y-axis (z-axis on spectrogram): The squared spectrum ("power spectrum") corresponds more closely to human perception and is generally preferred; note that log scale obscures the difference between these two, since

$$\log(x^2) = 2 * \log(x)$$

- X-axis (y-axis on spectrogram): Log scale corresponds to music notation and the piano keyboard;

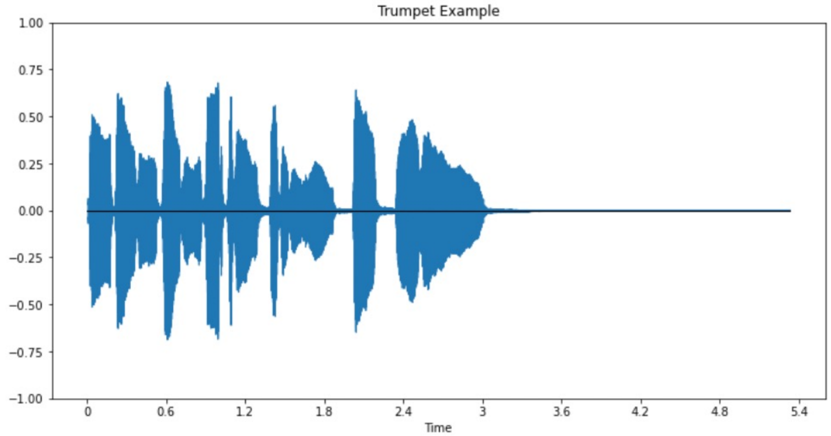
- But human perception of pitch corresponds better with the "Mel Scale":

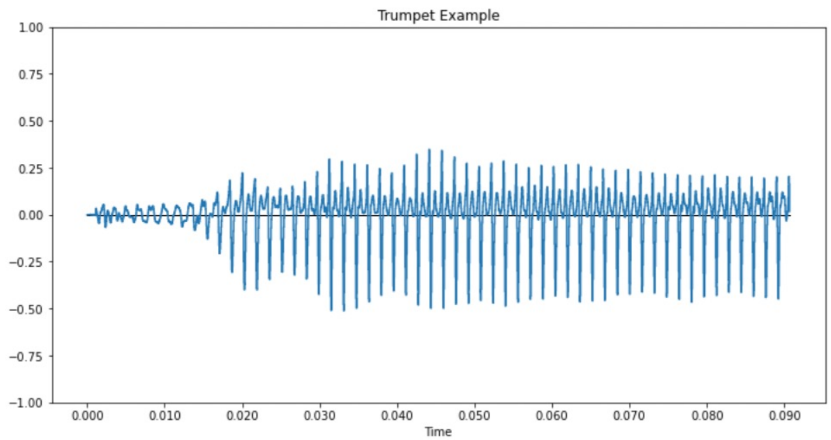


- Log scales in displaying frequency OR amplitude/power axis help with understanding but do NOT change the data (unless you make it so)

Example: Trumpet Example in Librosa

```
64]: y, sr = librosa.load(librosa.ex('trumpet'))

displaySignal(y,title='Trumpet Example')

displaySignal(y[:2000], title='Trumpet Example')

Audio(y,rate=SR)
```





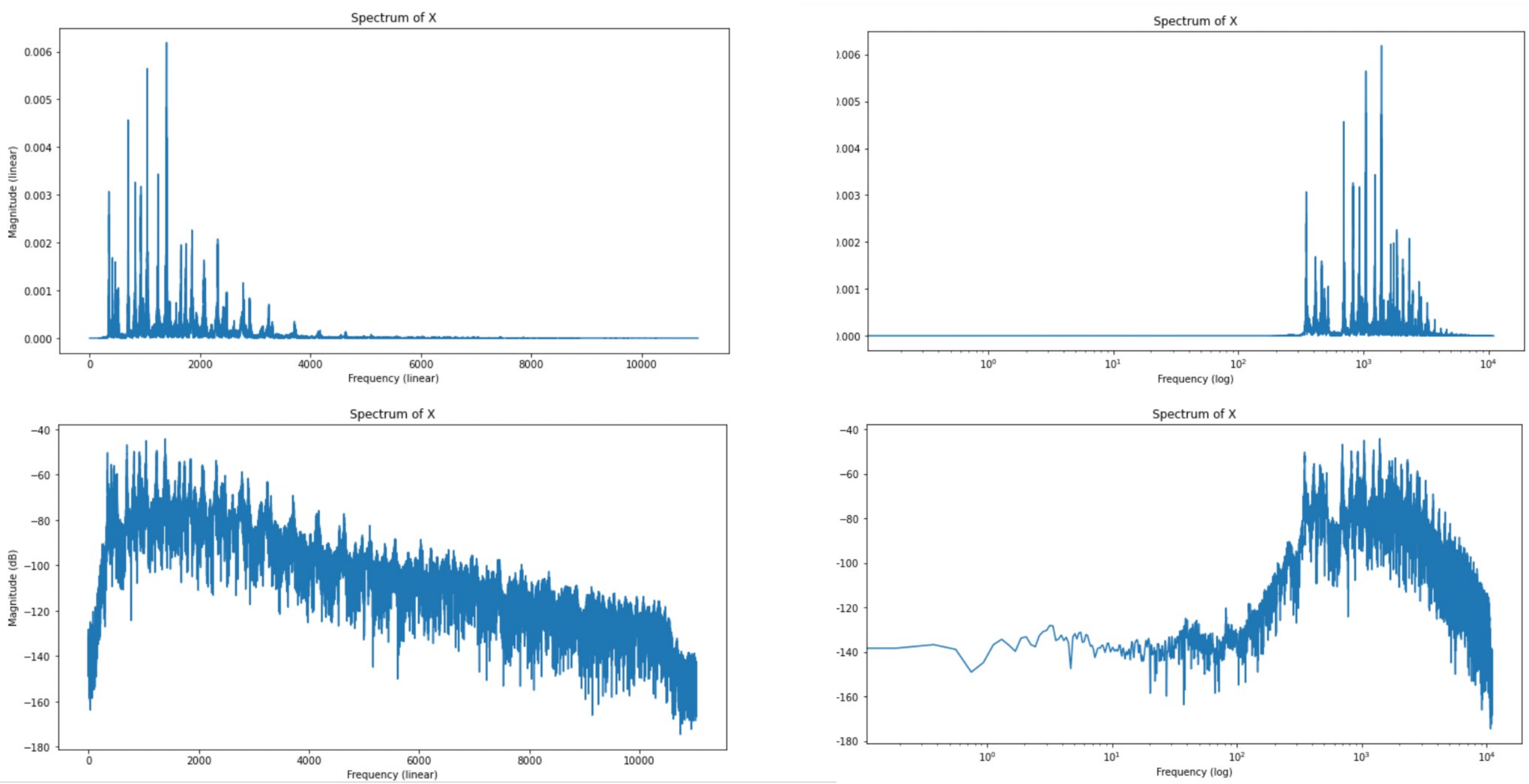
Digital Audio Fundamentals: The Discrete Fourier Transform

Computer Science

Spectrum of Trumpet: It only really makes sense to take the "instantaneous spectrum" of a short window of a signal; otherwise, the spectrum mixes up all the pitches AND includes timing information; here is the spectrum of the whole signal:

Magnitude

Log (dB)

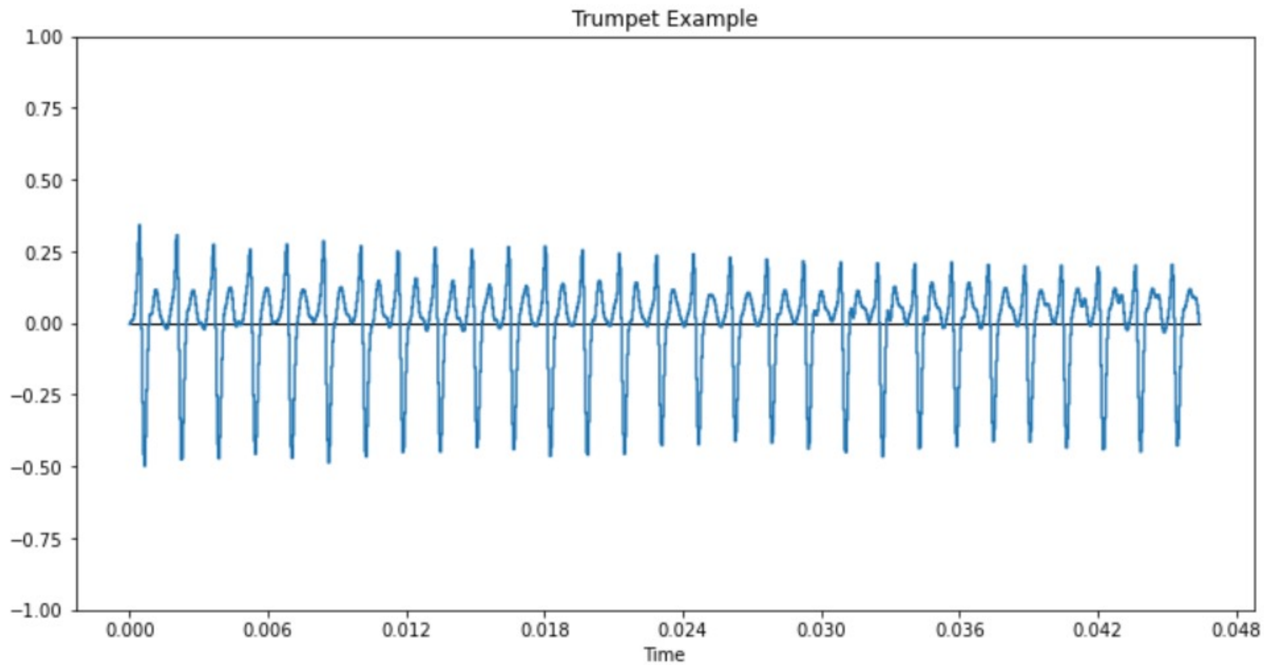


Linear Frequency

Log Frequency

Here is a small window, isolating the sound of a single note:

```
y, sr = librosa.load(librosa.ex('trumpet'))

y = y[1000:2024]

displaySignal(y,title='Trumpet Example')

displaySignal(y[:2000], title='Trumpet Example')

Audio(y,rate=SR)
```

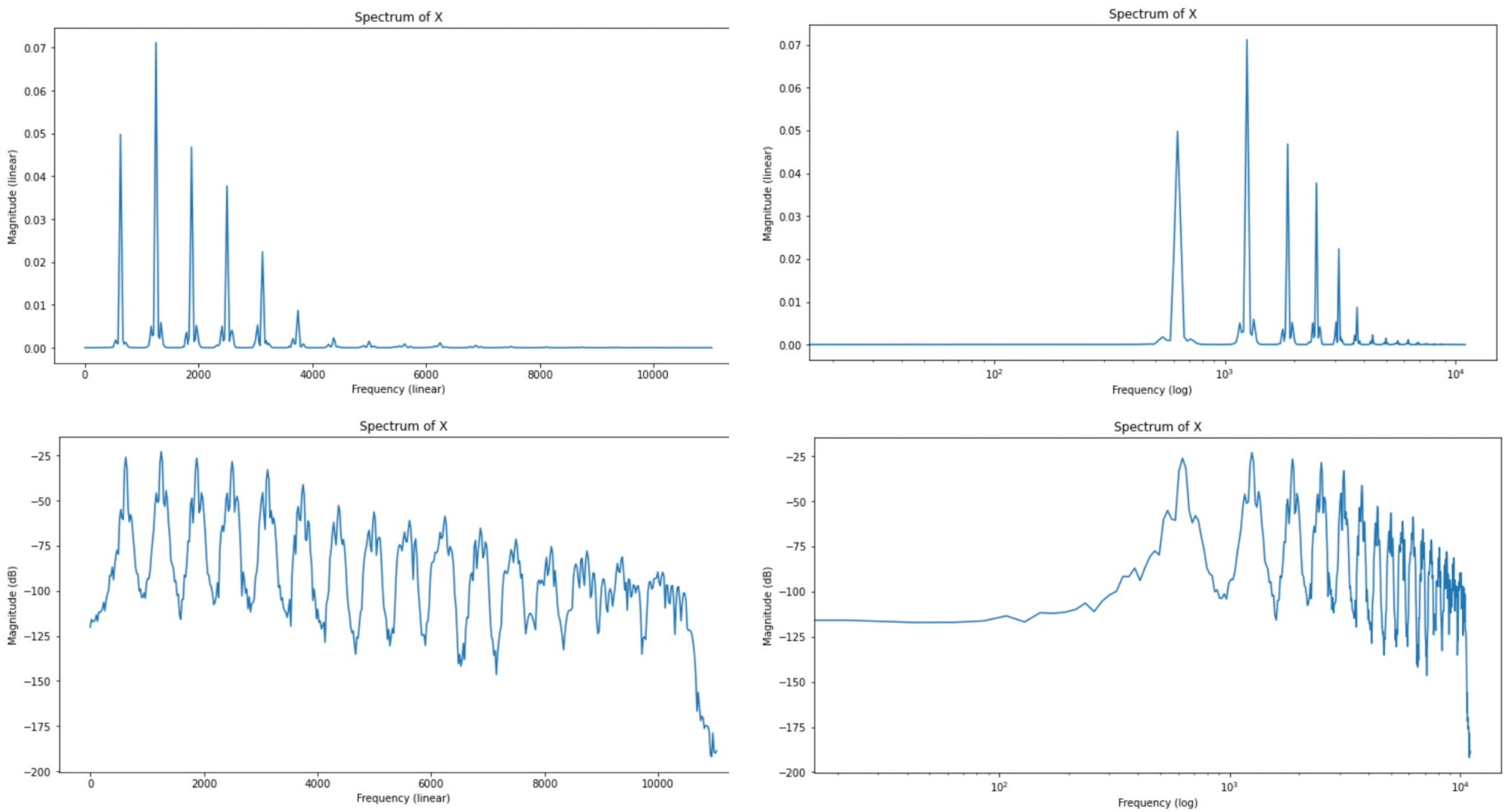


Digital Audio Fundamentals: The Discrete Fourier Transform

Here is a spectrum of the window:

Magnitude

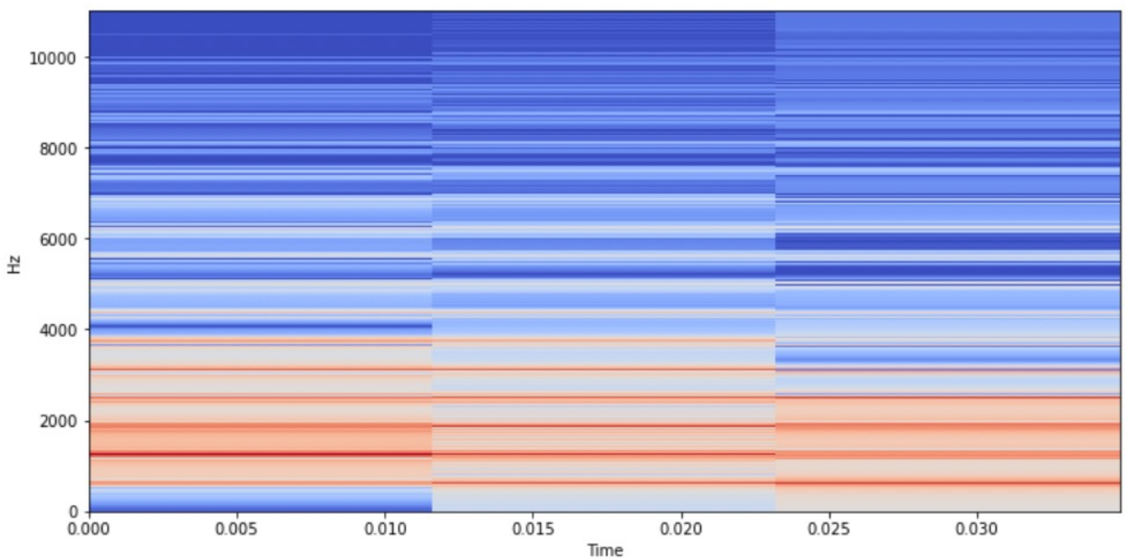
Log (dB)



Linear Frequency   Log Frequency

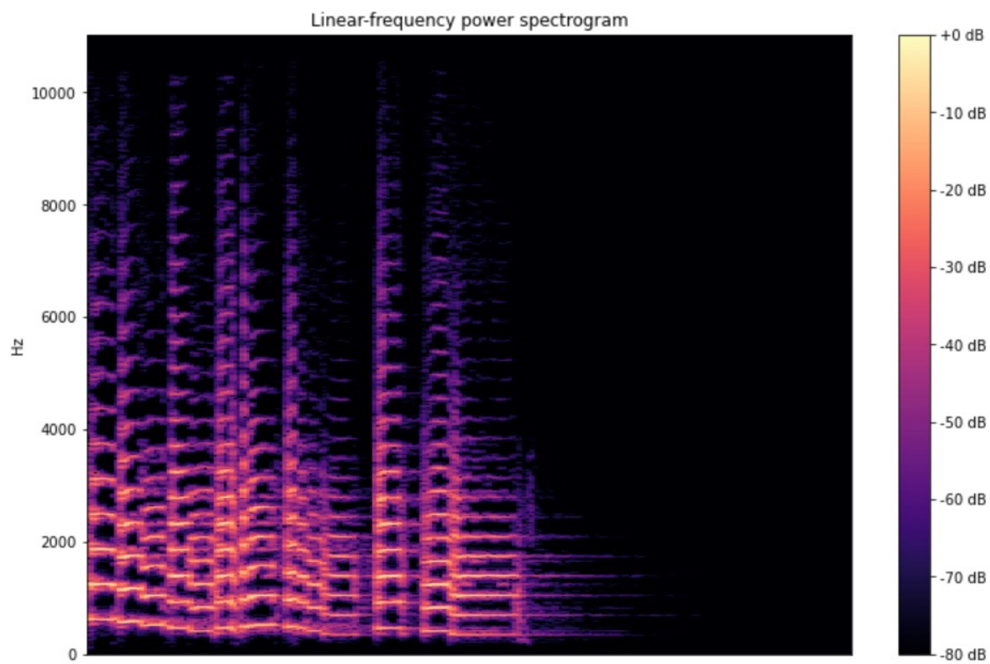
Ha, the spectrogram doesn't tell you much, even with a small hop length (=skip):

```
]: y, sr = librosa.load(librosa.ex('trumpet'))

   x = y[1000:2024]

   x = S = librosa.stft(x)
   Sdb = librosa.amplitude_to_db(abs(S))
   plt.figure(figsize=(12,6))
   librosa.display.specshow(Sdb, sr=sr, hop_length=256, x_axis='time', y_axis='hz')
   #plt.colorbar()
   plt.show()
```



There are a variety of ways to scale the frequency axis: Here is linear scale:

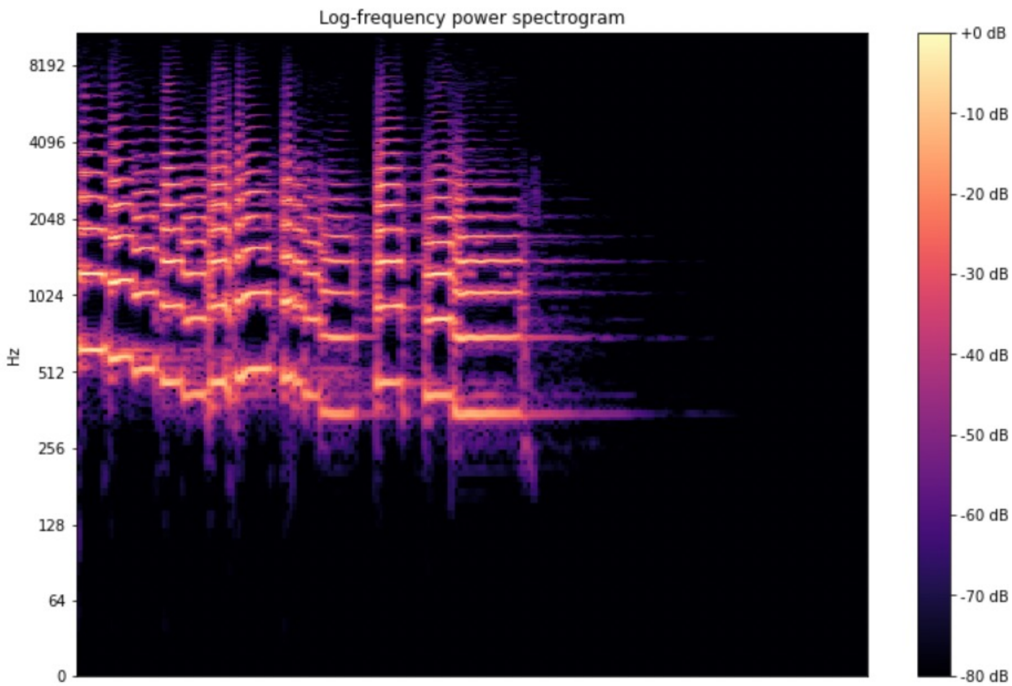
```
4]: y, sr = librosa.load(librosa.ex('trumpet'))
    plt.figure(figsize=(12, 8))
    D = librosa.amplitude_to_db(librosa.stft(y), ref=np.max)
    librosa.display.specshow(D, y_axis='linear')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Linear-frequency power spectrogram')

4]: Text(0.5, 1.0, 'Linear-frequency power spectrogram')
```



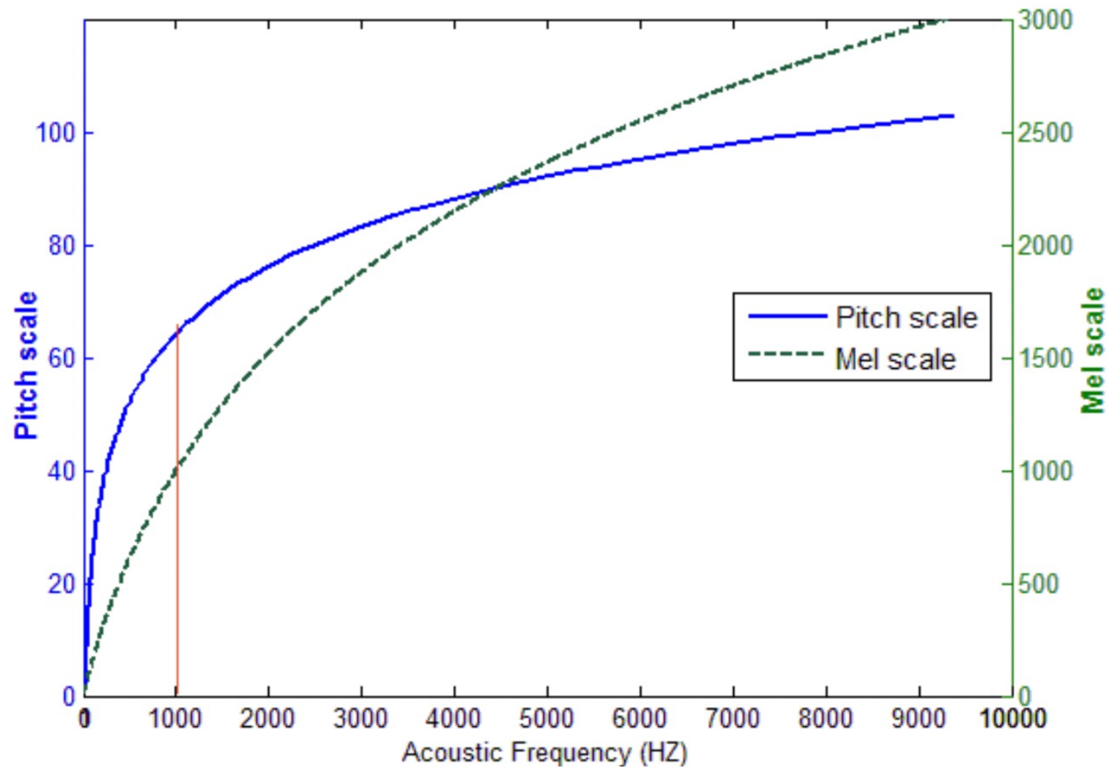
There are a variety of ways to scale the frequency axis: Here is log scale:

```
y, sr = librosa.load(librosa.ex('trumpet'))
plt.figure(figsize=(12, 8))
D = librosa.amplitude_to_db(librosa.stft(y), ref=np.max)
librosa.display.specshow(D, y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Log-frequency power spectrogram')
```

```
Text(0.5, 1.0, 'Log-frequency power spectrogram')
```



The mel spectrogram uses the mel (as in MELody) scale which corresponds more to the human perception of pitch:



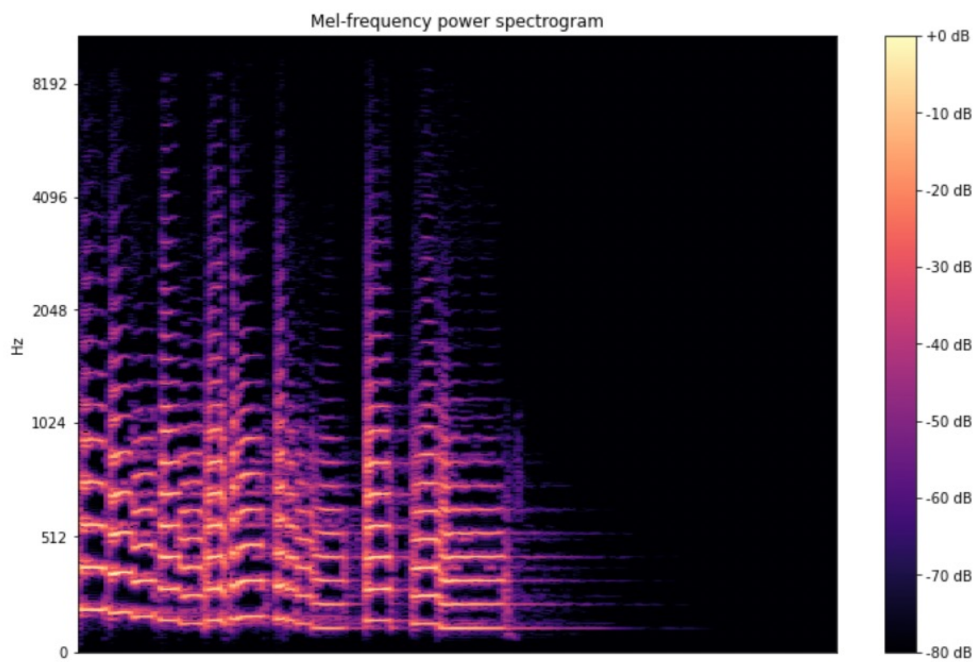
Digital Audio Fundamentals: The Discrete Fourier Transform

Computer Science

There are a variety of ways to scale the frequency axis: Here is mel scale:

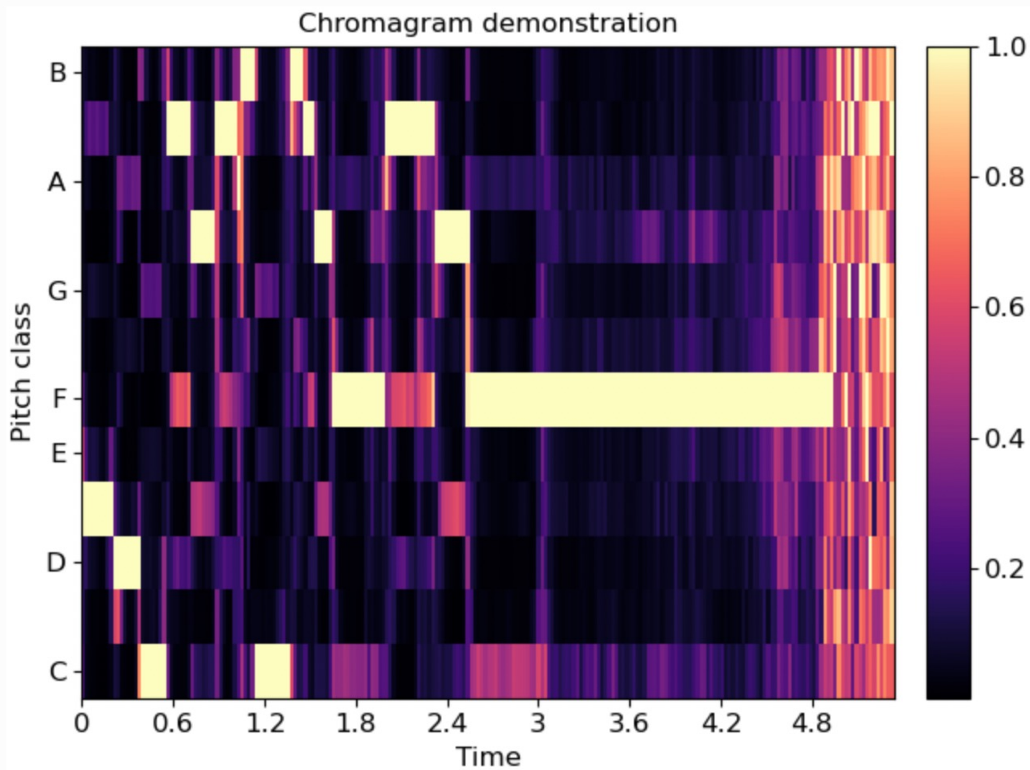
```
y, sr = librosa.load(librosa.ex('trumpet'))
plt.figure(figsize=(12, 8))
D = librosa.amplitude_to_db(librosa.stft(y), ref=np.max)
librosa.display.specshow(D, y_axis='mel')
plt.colorbar(format='%+2.0f dB')
plt.title('Mel-frequency power spectrogram')
```

```
Text(0.5, 1.0, 'Mel-frequency power spectrogram')
```



We can also combine all octaves into pitch classes uses a ”chroma”:

```
chroma = librosa.feature.chroma_cqt(y=y, sr=sr)
fig, ax = plt.subplots()
img = librosa.display.specshow(chroma, y_axis='chroma', x_axis='time', ax=ax)
ax.set(title='Chromagram demonstration')
fig.colorbar(img, ax=ax)
```



Digital Audio Fundamentals: The Inverse DFTR

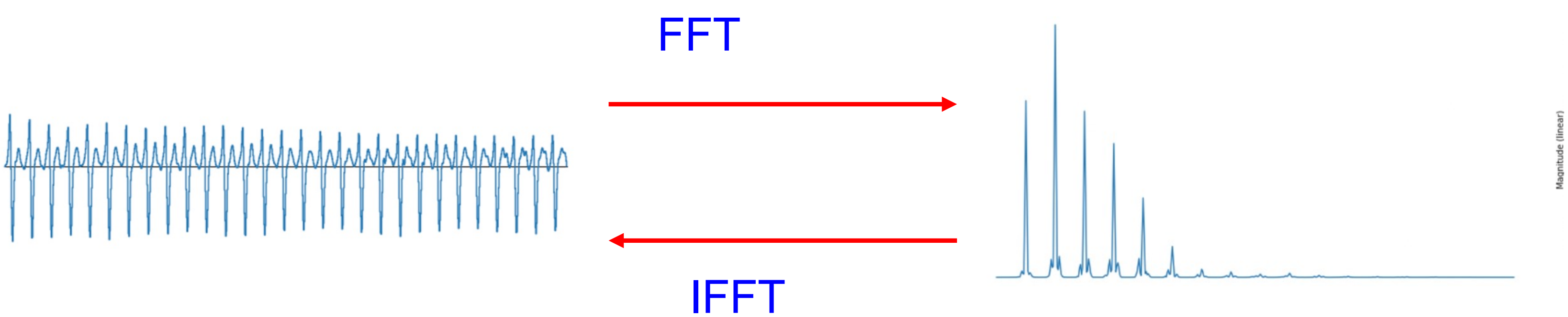
The Fourier Transforms come in pairs:

Discrete Fourier Transform:

signal (amplitude vs. time) -> spectrum (amplitude vs frequency)

Inverse DFT:

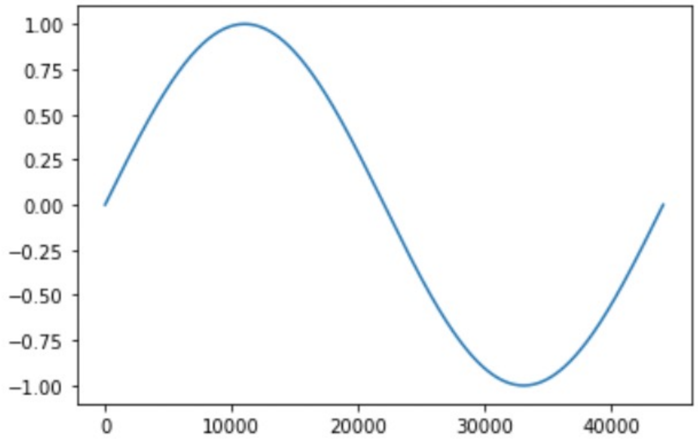
spectrum (amplitude vs frequency) -> signal (amplitude vs. time)



Digital Audio Fundamentals: The Inverse DFTR

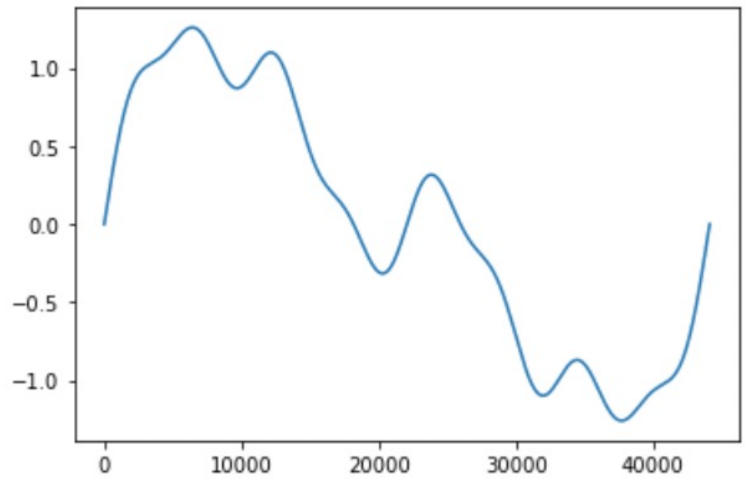
Computer Science

```
: # To create a signal of N samples, must input a spectrum array
  # of length N/2 + 1 amplitudes, with the kth frequency bin
  # representing the amplitude of frequency k*(SR/N)

  def realIFFT(S,A=None):
      S = np.array(S)
      lenX = 2*(len(S)-1)
      complex_S = lenX / 2 * -1.j * S
      X = np.fft.irfft(complex_S)
      if(A == None):
          return X
      else:
          return A * X / max(X)

  S = np.zeros(22050)
  S[1] = 1.0
  X = realIFFT(S)
  plt.plot(X)
  plt.show()
```



Digital Audio Fundamentals: The Inverse DFTR

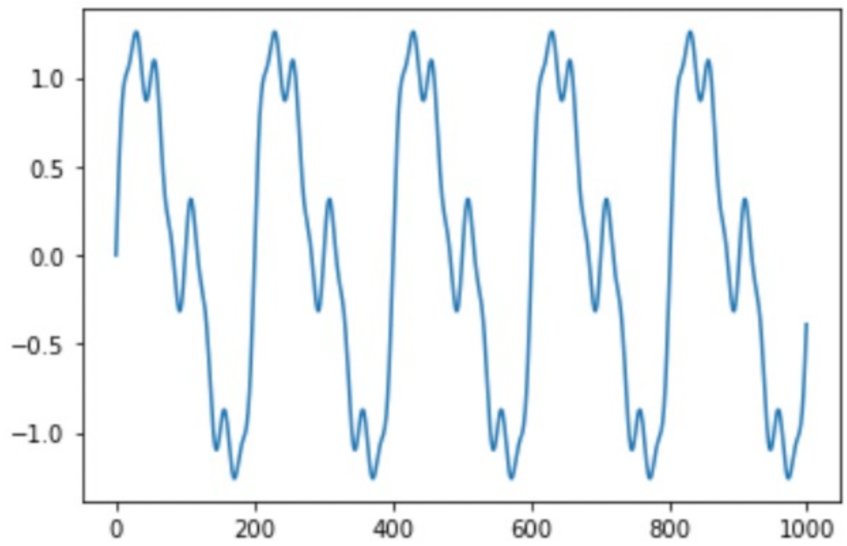

```
S = np.zeros(22050)
S[1] = 1.0
S[2] = 0.5
S[4] = 0.25
S[8] = 0.125
X = realIFFT(S)
plt.plot(X)
plt.show()
```



Digital Audio Fundamentals: The Inverse DFTR

Computer Science

```
S = np.zeros(22050)
S[220] = 1.0
S[440] = 0.5
S[880] = 0.25
S[1760] = 0.125
X = realIFFT(S)
plt.plot(X[:1000])
plt.show()

Audio(X,rate=SR)
```



Digital Audio Fundamentals: The Inverse DFTR

Except for the always-present "floating-point error," these are symmetric:

S = FFT(X)   X = IFFT(S)

X = IFFT(FFT(X))   S = FFT (IFFT(X))

Digital Audio Fundamentals: The Inverse DFTR

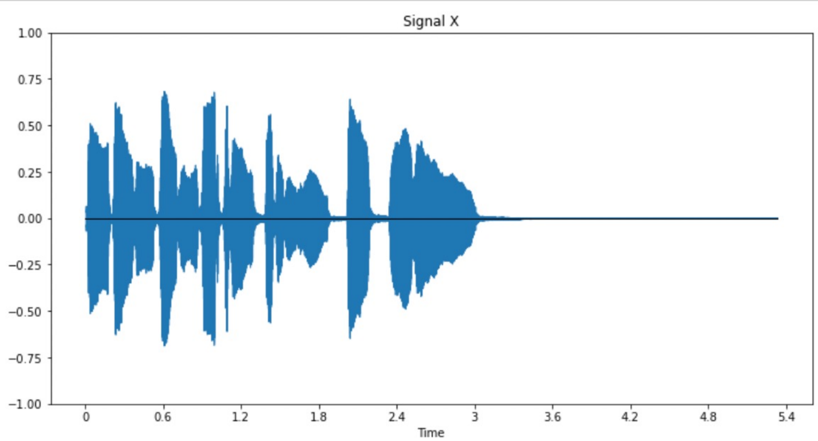
Computer Science

Except for the always-present "floating-point error," these are symmetric:

S = FFT(X) X = IFFT(S)

X = IFFT(FFT(X)) S = FFT (IFFT(X))

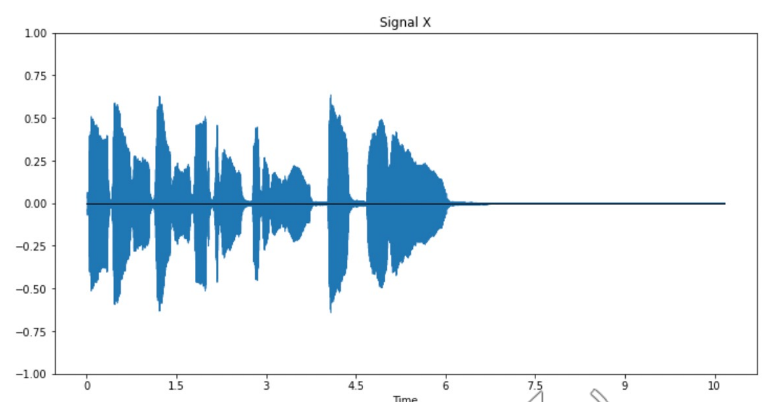
```
X, sr = librosa.load(librosa.ex('trumpet'))


displaySignal(X)
Audio(X,rate=sr)
```



0:00 / 0:05

```
X = np.fft.irfft( np.fft.fft(X) )
print(len(X))
displaySignal(X)
Audio(X,rate=44100)
```

235200



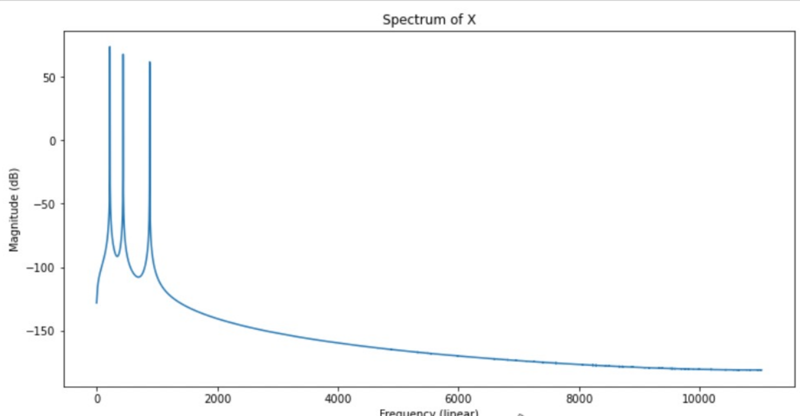
0:05 / 0:05

IMPORTANT NOTE:

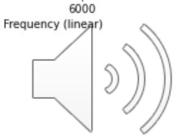
We are only talking about magnitude or power spectra, which don't show the phase!

Your ears can't distinguish phase information for a simple signal with only tonal data:

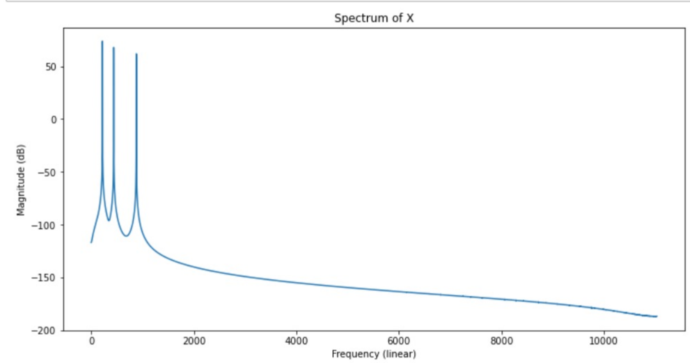
104]:
```
T = np.arange(SR*1)     # duration = 1 sec
X = 10000 * np.sin( 2 * np.pi * 220 * T / SR )
X += 5000 * np.sin( 2 * np.pi * 440 * T / SR )
X += 2500 * np.sin( 2 * np.pi * 880 * T / SR )

displaySpectrum(X)
Audio(X,rate=SR)
```



104]: 0:01 / 0:01



3]:
```
T = np.arange(SR*1)     # duration = 1 sec
X = 10000 * np.sin( 2 * np.pi * 220 * T / SR  + np.pi)
X += 5000 * np.sin( 2 * np.pi * 440 * T / SR + 1.23)
X += 2500 * np.sin( 2 * np.pi * 880 * T / SR - 3.42)

displaySpectrum(X)
Audio(X,rate=SR)
```



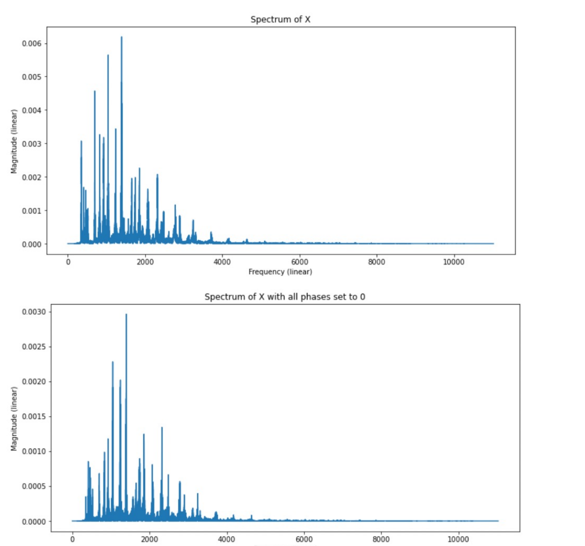
3]: 0:01 / 0:01

Digital Audio Fundamentals: The Inverse DFTR

The spectrum contains

- Tonal data (pitches)
- Timing data (onsets and rhythm)
- Noise

But that is only for simple signals, here is what happens if you remove all the phase information from the trumpet signal: The spectra look the same, but the signal is NOT the same in the time domain:



```
S = realFFT(X)

X1 = realIFFT(S)

displaySignal(X1)
Audio(X1,rate=SR)
```