CS 583– Computational Audio -- Fall, 2021

Wayne Snyder
Computer Science Department
Boston University

Lecture 09

Conclusions on Auto-Correlation for Pitch Detection

Discrete Sine Transform

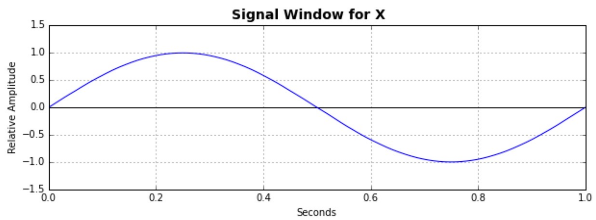
Complex Numbers

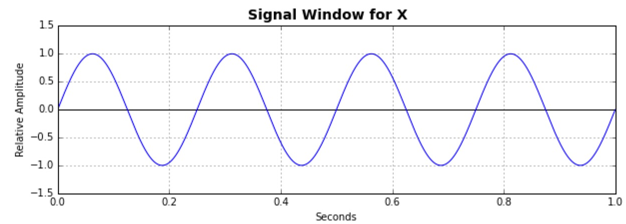
If times: The Discrete Fourier Transform

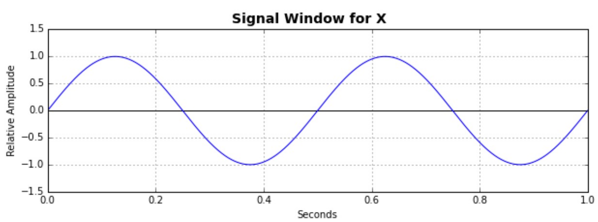
Computer Science

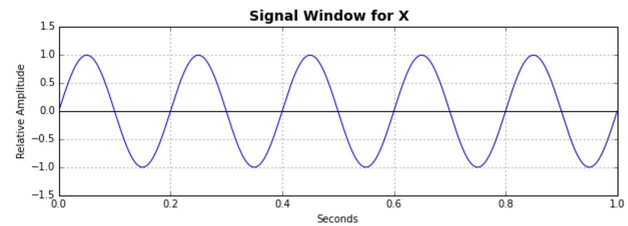
Digital Audio Fundamentals: Multiplying/Squaring Signals

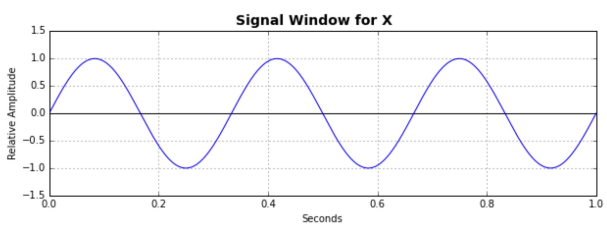
Define: For a signal X of length W samples (a “window”) a window frequency is one whose period P is such that W = P * k for some integer k, i.e., an integral number of periods exactly fit within the window; alternately, it begins and ends at same instantaneous phase.

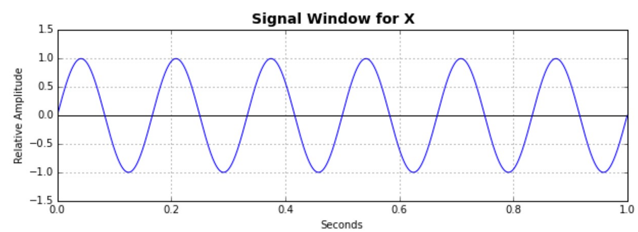












We will use these signals as **probe waves** to analyze a musical signal and assume that all such probe waves (for now) start at phase 0.0.

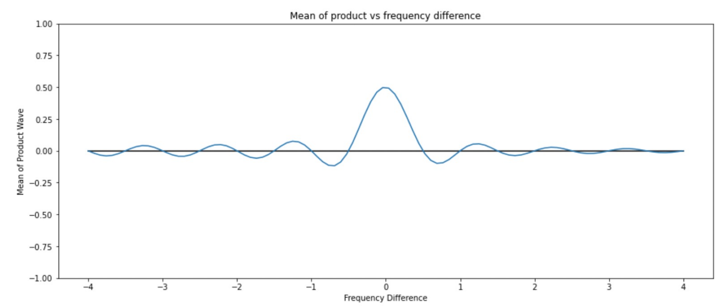
Digital Audio Fundamentals: Multiplying Sine Waves

Recall that when we take the correlation of two sine waves, we get 1.0 if the waves are the same frequency and phase, and close to 0.0 otherwise.

the correlation of two sine waves is

Dot Product

$$\frac{mean(X \cdot Y)}{\sigma_X \cdot \sigma_Y} = \frac{\sum_{k=0}^{N} X[k] \cdot Y[k]}{N \cdot \sigma_X \cdot \sigma_Y} = \frac{X@Y}{N \cdot \sigma_X \cdot \sigma_Y}$$



$$sinc(x) = \frac{sin(x)}{x}$$

However, if we suppose both waves have amplitude 1.0, we can simplify:

$$2 \cdot \frac{X@Y}{N} = 1.0$$

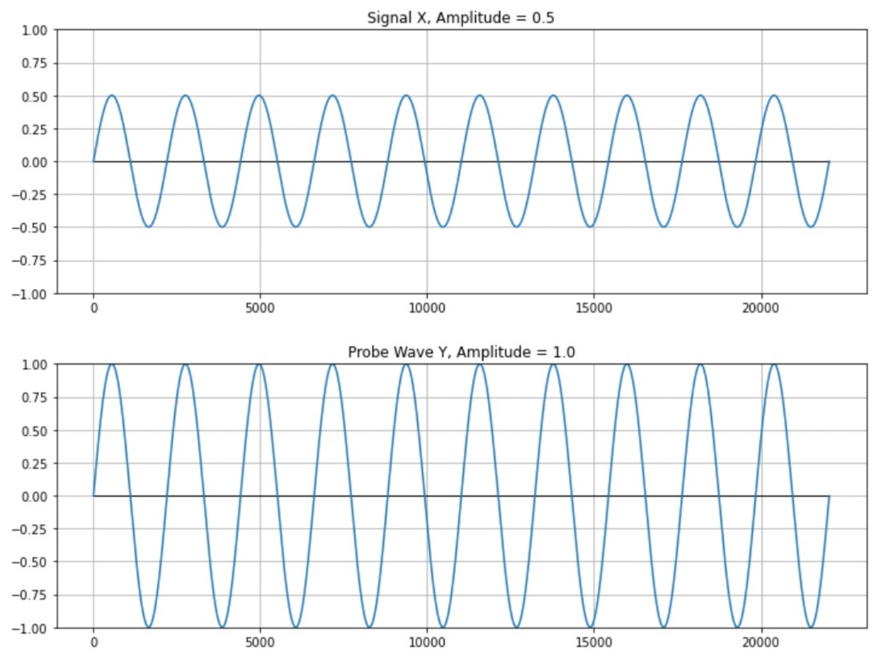
and if one (the "probe wave") has amplitude 1.0 and the other has amplitude A, where both have the same frequency and phase, we have:

$$2 \cdot \frac{A \cdot X@Y}{N} = A\left(2 \cdot \frac{X@Y}{N}\right) = A$$

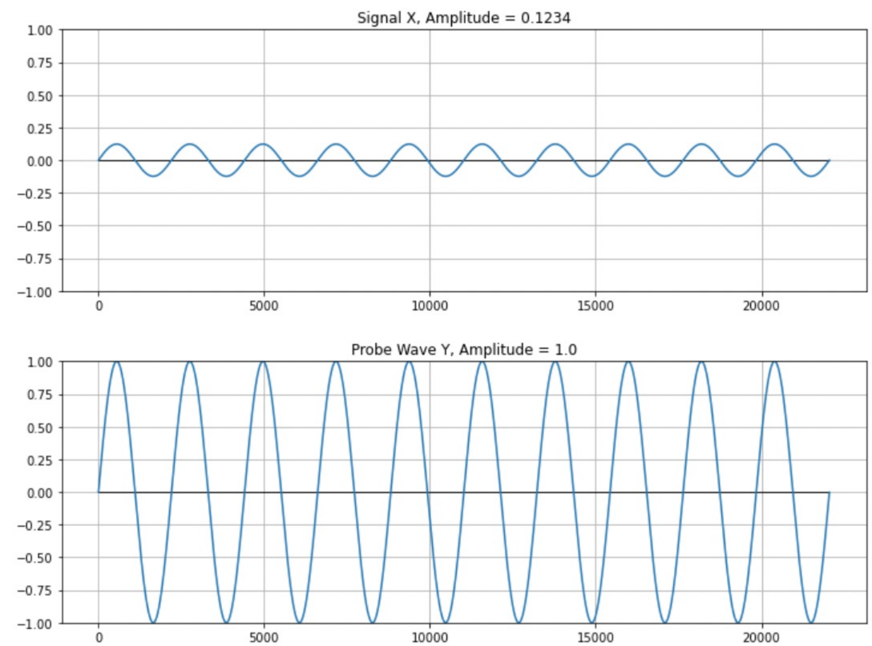
Digital Audio Fundamentals: Multiplying Sine Waves

Computer Science

Therefore we have a “detector” for finding the amplitude of a given signal X, as long as we know the frequency and phase:




```
2*(X @ Y)/N = 0.5
```

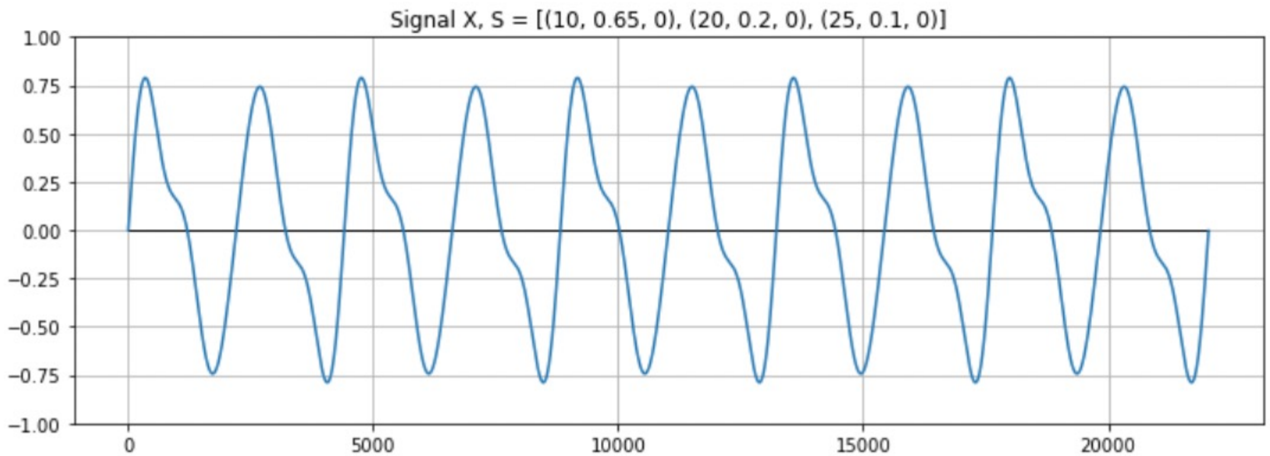


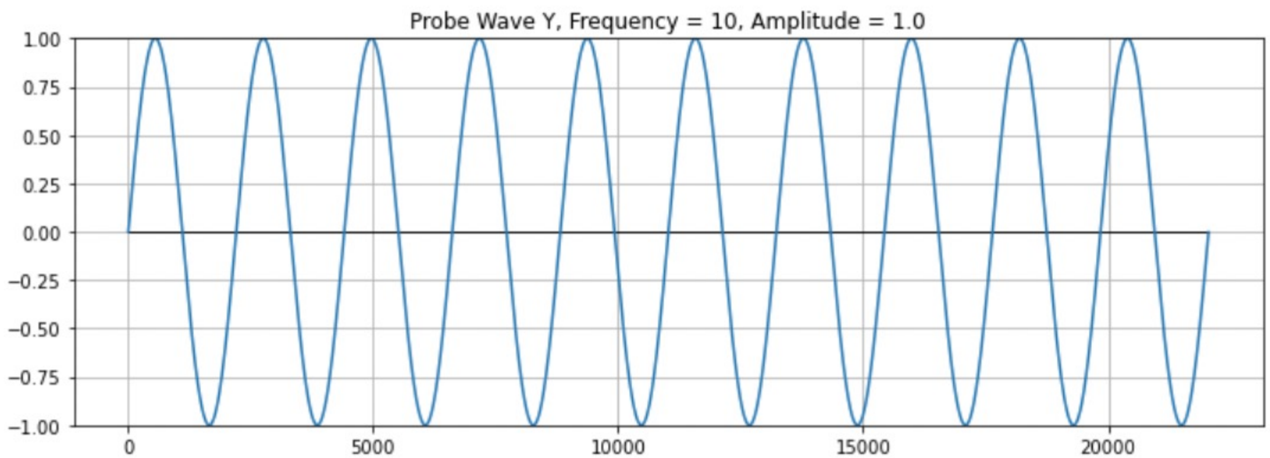

```
2*(X @ Y)/N = 0.1234
```

Digital Audio Fundamentals: Multiplying/Squaring Signals

Computer Science

Now, What happens when the signal is composite (not a simple sine wave)?

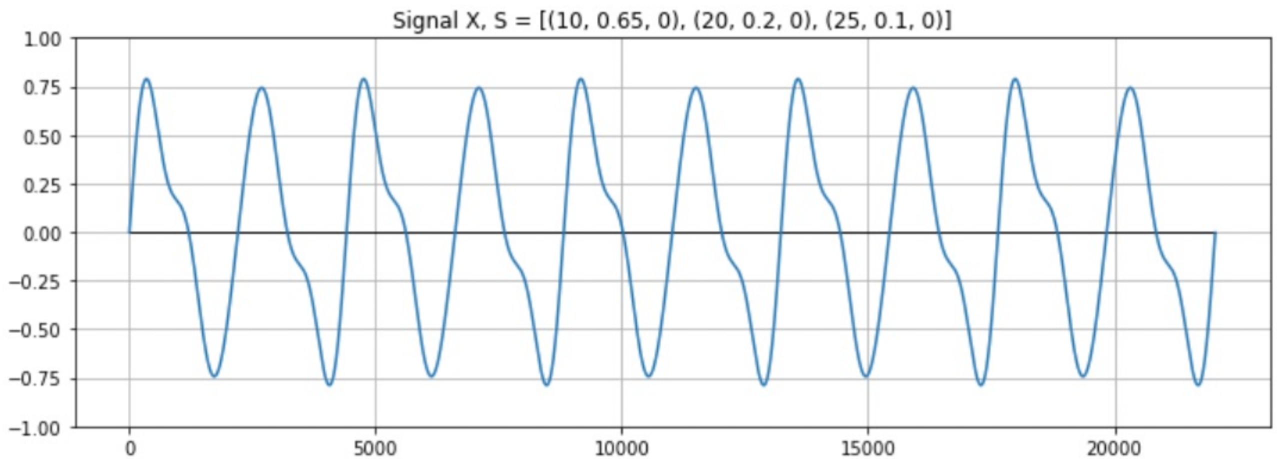


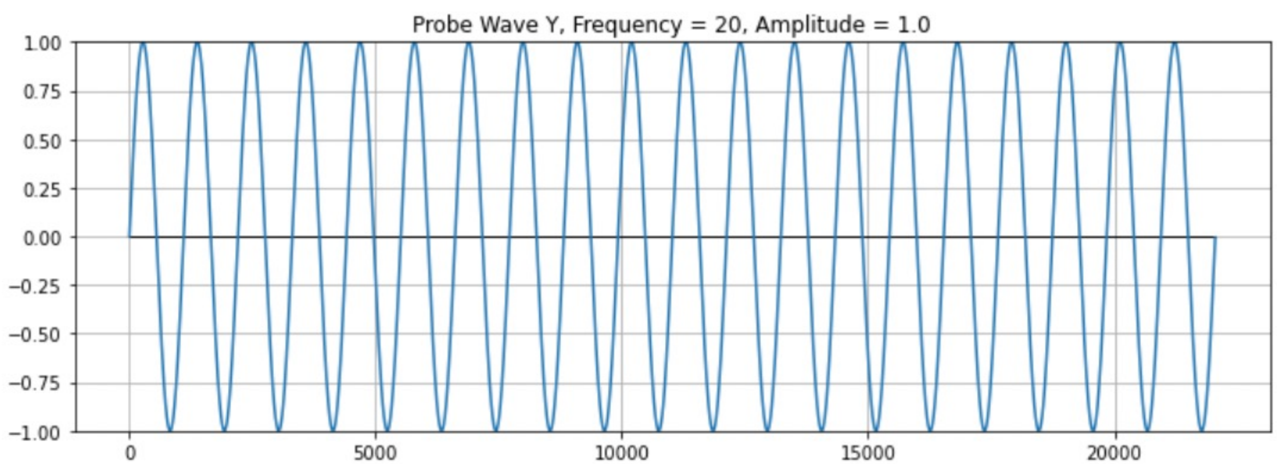



```
2*(X @ Y)/N = 0.65
```

Digital Audio Fundamentals: Multiplying/Squaring Signals

Now, What happens when the signal is composite (not a simple sine wave)?

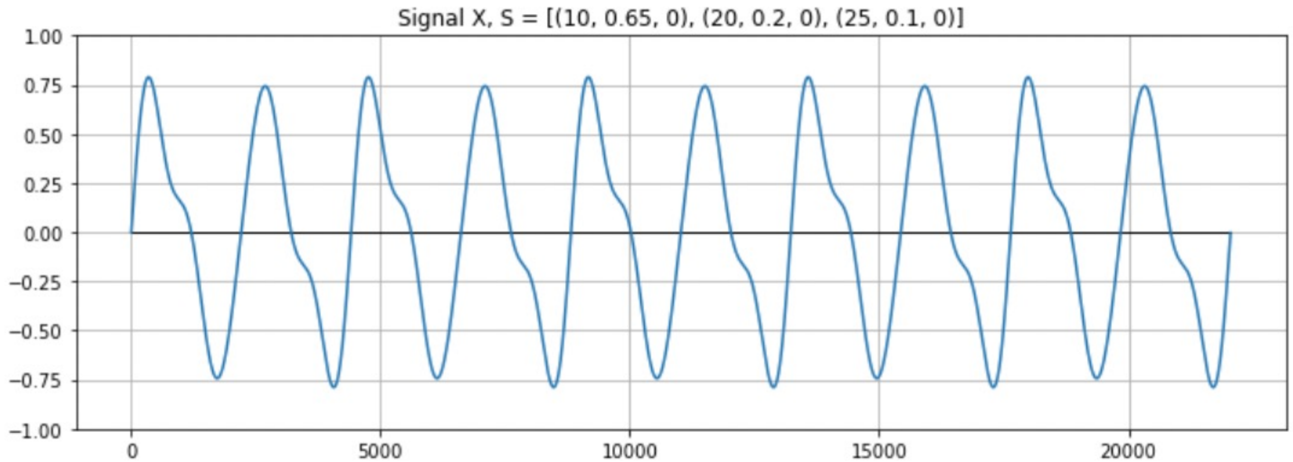


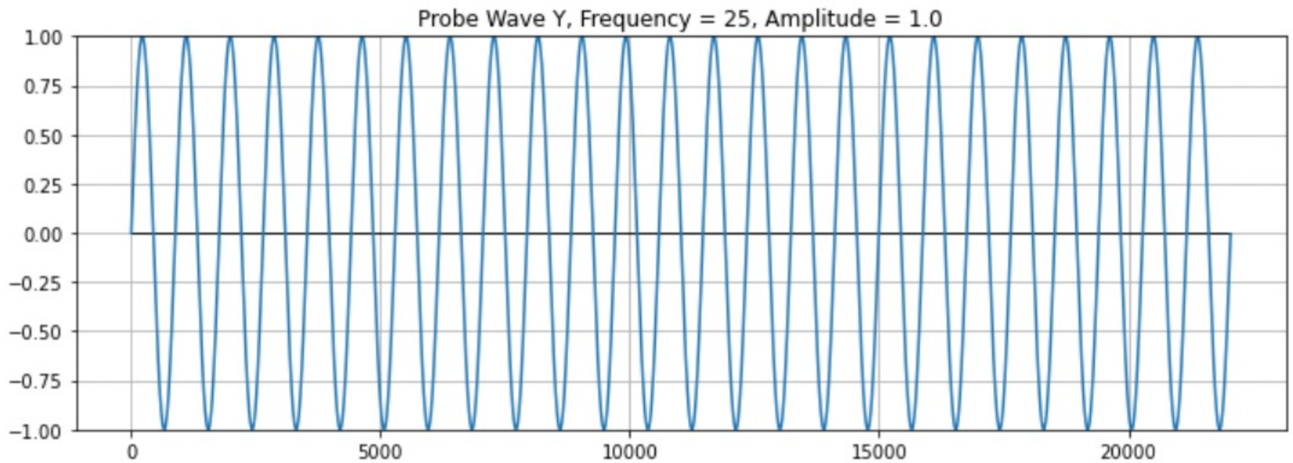



```
2*(X @ Y)/N = 0.2
```

Digital Audio Fundamentals: Multiplying/Squaring Signals

Now, What happens when the signal is composite (not a simple sine wave)?



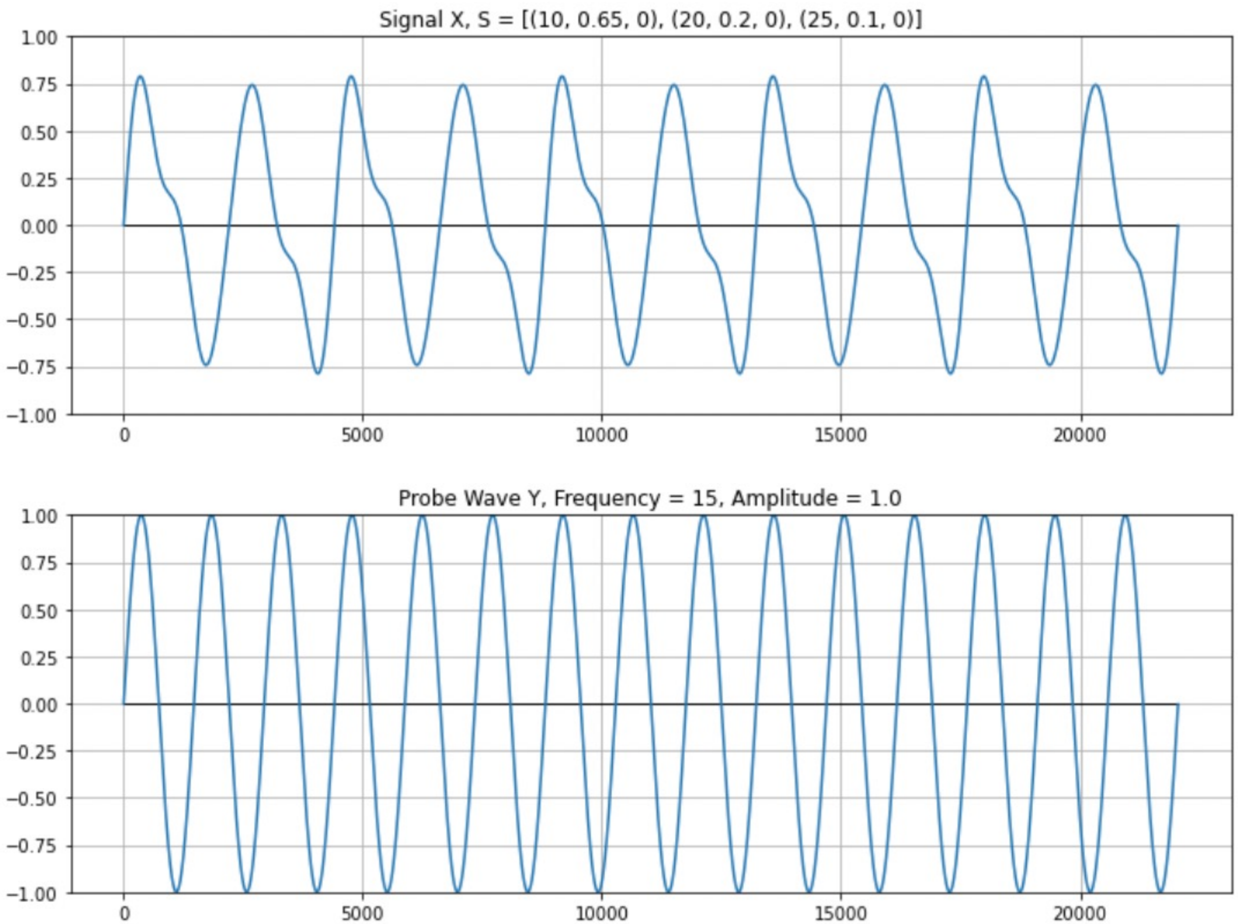



```
2*(X @ Y)/N = 0.1
```

Digital Audio Fundamentals: Multiplying/Squaring Signals

Computer Science

Now, What happens when the signal is composite (not a simple sine wave)?




```
2*(X @ Y)/N = -0.0
```

Punchline:

Window frequencies are orthogonal, and so the probe wave can detect components of waves just as easily as simple sine waves!

Digital Audio Fundamentals: Discrete Sine Transform

Doing this consistently for all window frequencies gives us the **Discrete Sine Transform**:

```
def DST( X ):
    W = len(X)                  # window length in samples
    S = [0] * (W // 2)
    for f in range( W//2 ):     # for each probe wave f in [0..W//2]
        for k in range(W):      # sum the products of signal and probe and save in S[f]
            S[f] += X[k] * sin(2 * pi * f * k / W)
        S[f] = 2 * S[f] / W     # normalize to mean of products to get actual amplitude
    return S
```

X = makeSignal([(3.0,0.5,0.0), (5.0,0.3,0.0), (10,0.2,0.0)], 1.0)
S = DST(X)

S[0]: 0.0
S[1]: 7.89649143503e-12
S[2]: -7.39108746491e-12
S[3]: 0.499999999998
S[4]: 4.00080251115e-12
S[5]: 0.29999999999
......
S[10]: 0.20000000001
S[11]: -1.83215982068e-12
......
S[22049]: 2.55635210657e-12

Digital Audio Fundamentals: Discrete Sine Transform

Computer Science

Doing this consistently for all window frequencies gives us the **Discrete Sine Transform**:

```
def DST( X ):
    W = len(X)                      # window length in samples
    S = [0] * (W // 2)
    for f in range( W//2 ):         # for each probe wave f in [0..W//2]
        for k in range(W):          # sum the products of signal and probe and save in S[f]
            S[f] += X[k] * sin(2 * pi * f * k / W)
        S[f] = 2 * S[f] / W         # normalize to mean of products to get actual amplitude
    return S
```

Note:

The transform can ONLY detect window frequencies = k * f for f = 1 / W (in secs)
= k * SR / W   (in samples)

So a window of 1.0 seconds can detect   0, 1, 2, …., 22049   ONLY
of 0.1 seconds can detect 0, 10, 20, 30, …., 22040
of 0.2 seconds can detect 0, 5, 10, …., 22040

Another problem is that this took 20minutes to run!

Double for loop with W = 44100…   44100 * 22050 = 972,405,000 executions of inner loop!

Digital Audio Fundamentals: Discrete Sine Transform

Doing this consistently for all window frequencies gives us the **Discrete Sine Transform**:

```
def DST( X ):
    W = len(X)                      # window length in samples
    S = [0] * (W // 2)
    for f in range( W//2 ):         # for each probe wave f in [0..W//2]
        for k in range(W):          # sum the products of signal and probe and save in S[f]
            S[f] += X[k] * sin(2 * pi * f * k / W)
        S[f] = 2 * S[f] / W         # normalize to mean of products to get actual amplitude
    return S
```

X = makeSignal([(30.0,0.5,0.0), (50.0,0.3,0.0), (100.0, 0.2,0.0)], 0.1)
S = DST(X)

Bin	Amp	Freq
S[0]:	0.0	0
S[1]:	-3.93616764277e-12	10 ……
S[3]:	0.4999999999987	30
S[4]:	6.72379914407e-12	40
S[5]:	0.30000000001	50
……		
S[10]:	0.29999999997	100
……		
S[2204]:	4.73093370979e-13	22040

This took about
15 seconds to run

Digital Audio Fundamentals: Discrete Sine Transform

Computer Science

Doing this consistently for all window frequencies gives us the **Discrete Sine Transform**:

```
def DST( X ):
    W = len(X)                      # window length in samples
    S = [0] * (W // 2)
    for f in range( W//2 ):         # for each probe wave f in [0..W//2]
        for k in range(W):          # sum the products of signal and probe and save in S[f]
            S[f] += X[k] * sin(2 * pi * f * k / W)
        S[f] = 2 * S[f] / W         # normalize to mean of products to get actual amplitude
    return S
```

X = makeSignal([(30.0,0.5,0.0), (50.0,0.3,0.0), (100.0, 0.2,0.0)], 0.2)
S = DST(X)

This took about 1 minute to run

Bin	Amp	Freq
S[0]:	0.0	0
S[1]:	9.58745925935e-13	5
	....	
S[6]:	0.5	30
	....	
S[10]:	0.30000000001	50
	....	
S[20]:	0.29999999999	100
	....	
S[4409]:	7.23056298634e-12	22045

Digital Audio Fundamentals: The Discrete Sine Transform

```
def DST( X ):
    W = len(X)                   # window length in samples
    S = [0] * (W // 2)
    for f in range( W//2 ):      # for each probe wave f in [0..W//2]
        for k in range(W):       # sum the products of signal and probe and save in S[f]
            S[f] += X[k] * sin(2 * pi * f * k / W)
        S[f] = 2 * S[f] / W      # normalize to mean of products to get actual amplitude
    return S
```

Returns a spectrum of amplitudes (in range -1 .. 1)

$S = [\ A_0,\ A_1,\ A_2,\ \ldots.,\ A_{N//2\ -\ 1}\ ]$ assuming w is even

for window frequencies

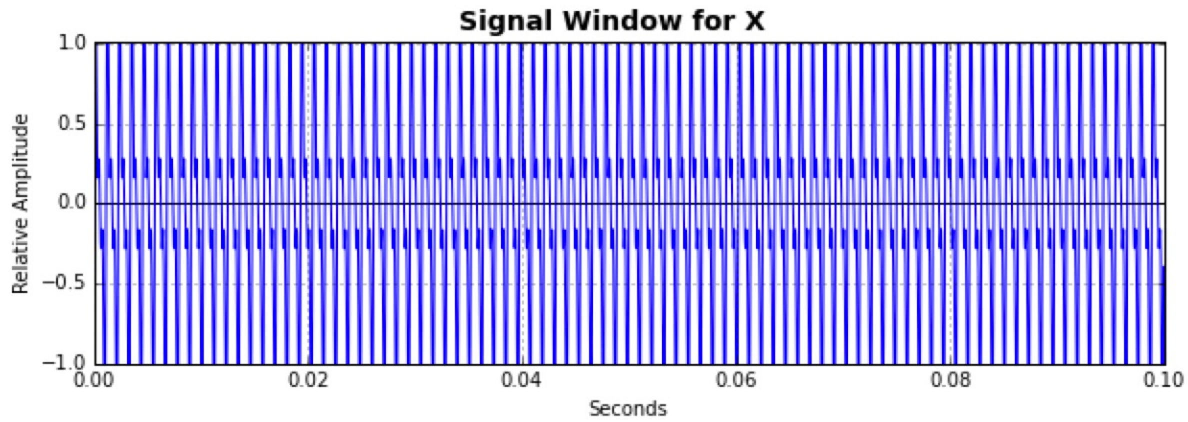
$W_f = [\ 0,\ 1,\ 2,\ \ldots.,\ N//1 - 1\ ]$

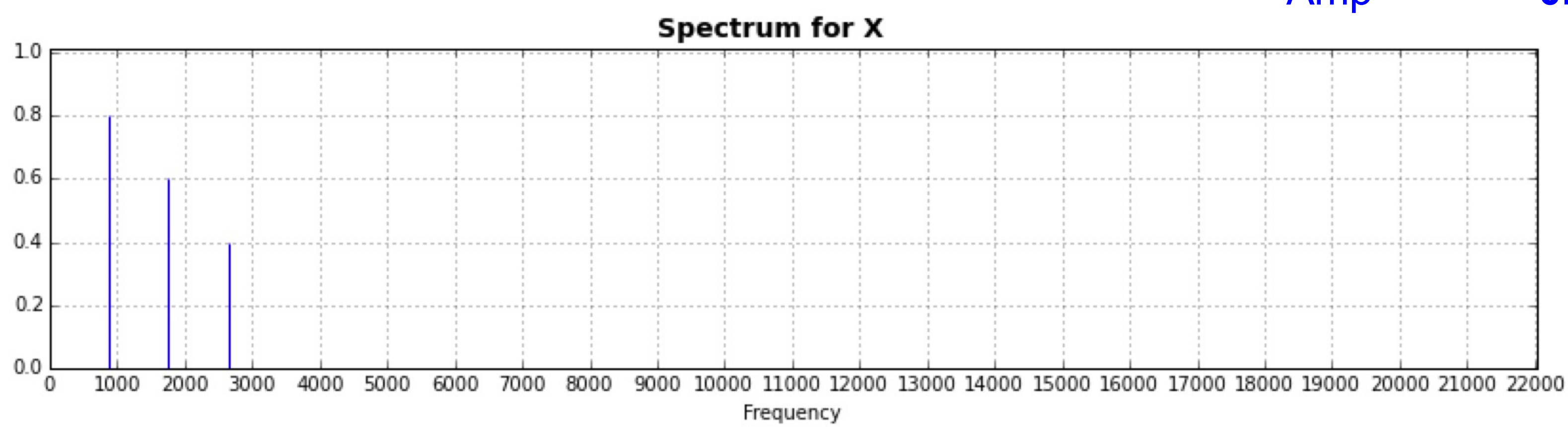
and actual frequencies

$F = [\ 0,\ 1R,\ 2R,\ \ldots.,\ R*(N//2 - 1)\ ]$ for R = SampleRate / W

Digital Audio Fundamentals: The Discrete Sine Transform

Spectrum: [(880, 0.8, 0), (1760, 0.6, 0), (2640, 0.4, 0)]






```
Freq     Amp

880.0    0.8
1760.0   0.6
2640.0   0.4
```

Interpreting Outputs from the Discrete Sine Transform:

88*SR/W = **880 Hz**
Amp = **0.8**

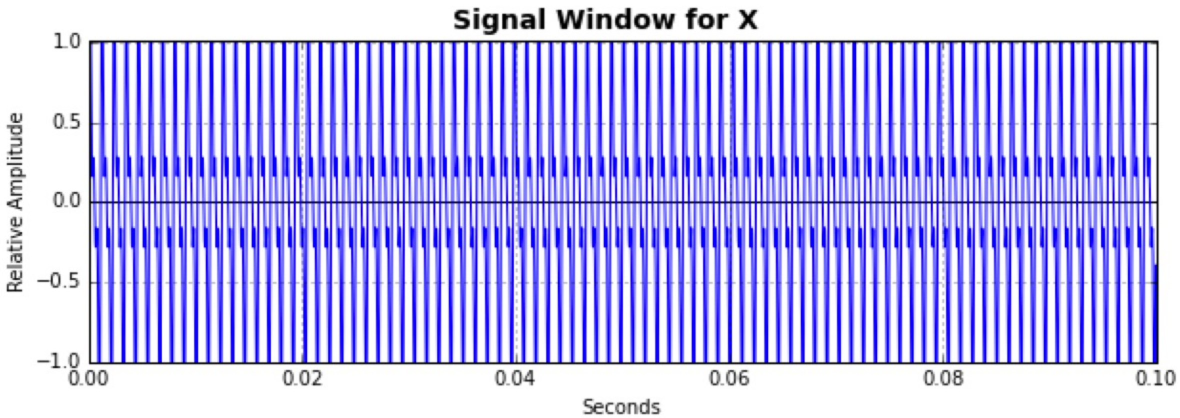
176*SR/W = **1760 Hz**
Amp = **0.6**

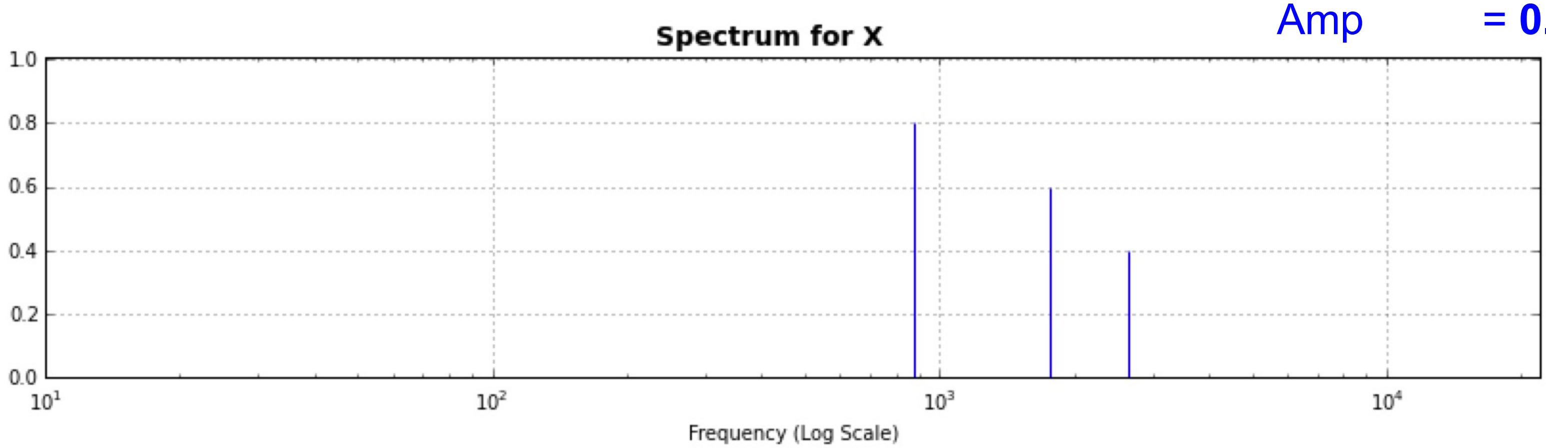
264*SR/W = **2640 Hz**
Amp = **0.4**

Digital Audio Fundamentals: The Discrete Sine Transform

Computer Science

Spectrum: [(880, 0.8, 0), (1760, 0.6, 0), (2640, 0.4, 0)]






```
Freq    Amp

880.0   0.8
1760.0  0.6
2640.0  0.4
```

Interpreting Outputs from the Discrete Sine Transform:

88*SR/W = **880 Hz**
Amp = **0.8**

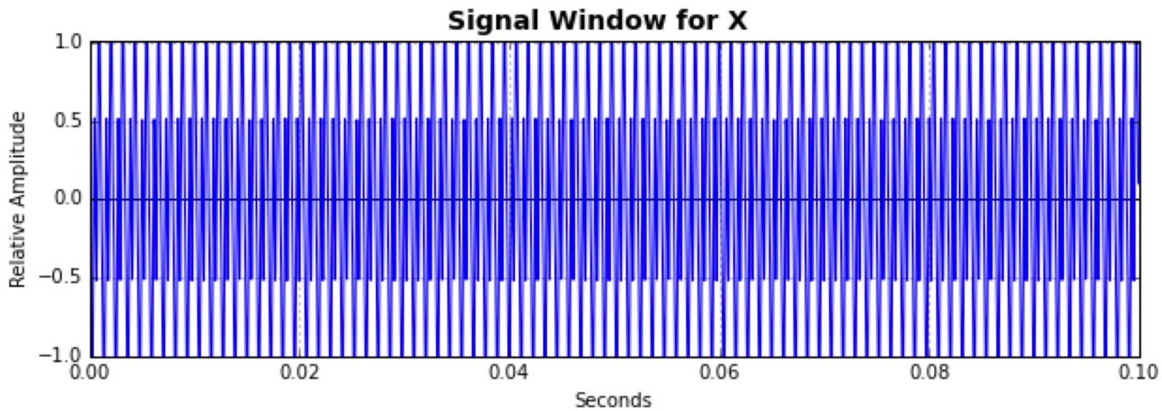
176*SR/W = **1760 Hz**
Amp = **0.6**

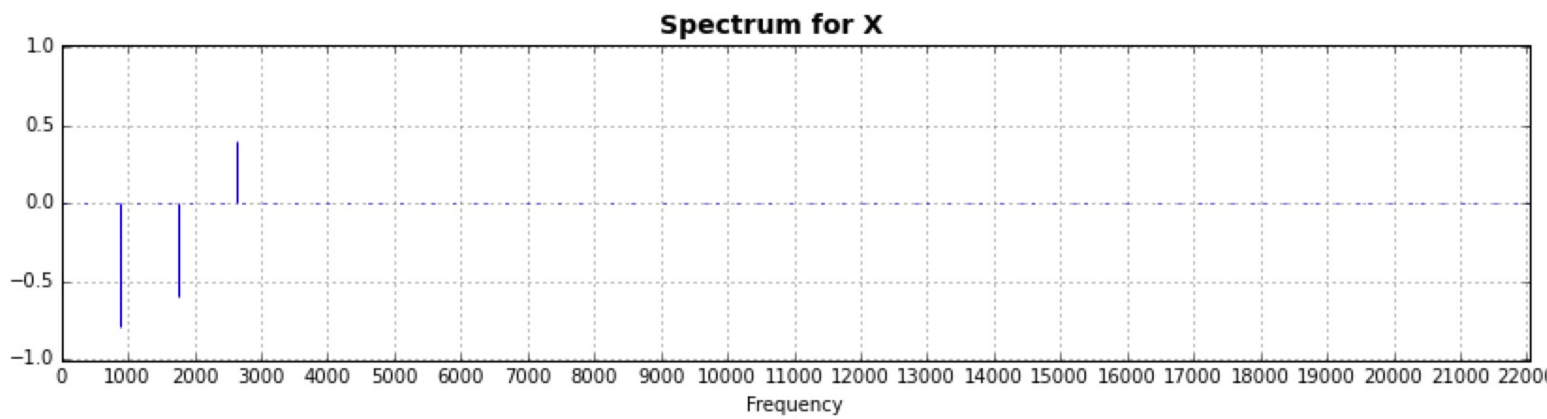
264*SR/W = **2640 Hz**
Amp = **0.4**

Digital Audio Fundamentals: The Discrete Sine Transform

Computer Science

Spectrum: [(880, -0.8, 0), (1760, -0.6, 0), (2640, 0.4, 0)]





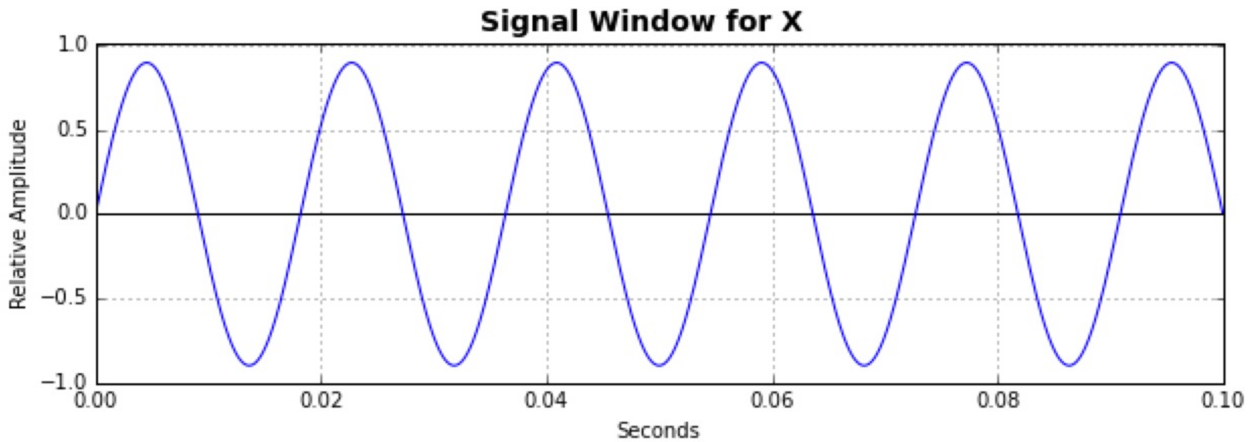

```
Freq    Amp

880.0   -0.8
1760.0  -0.6
2640.0  0.4
```

Interpreting Outputs from the Discrete Sine Transform:

Component sine waves may have a negative amplitude.

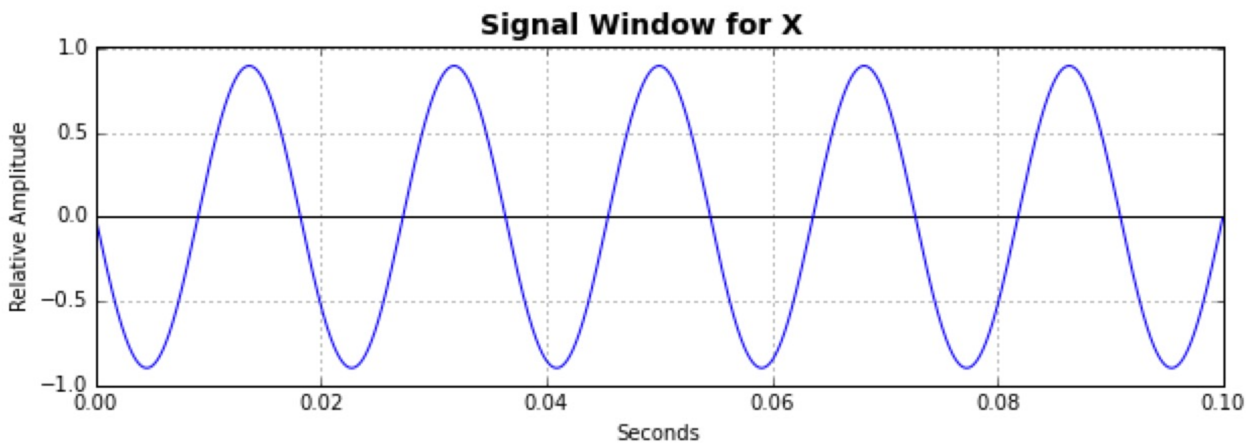
Digital Audio Fundamentals: The Discrete Sine Transform


```
In [260]: X = au.makeSignal([(55,0.9,0)],4410,"Samples")
     ...: au.displaySignal(X)
     ...:
```



Component sine waves may have a negative amplitude; they will produce the negative of a squared wave, and report negative amplitudes just as they report positive amplitudes.

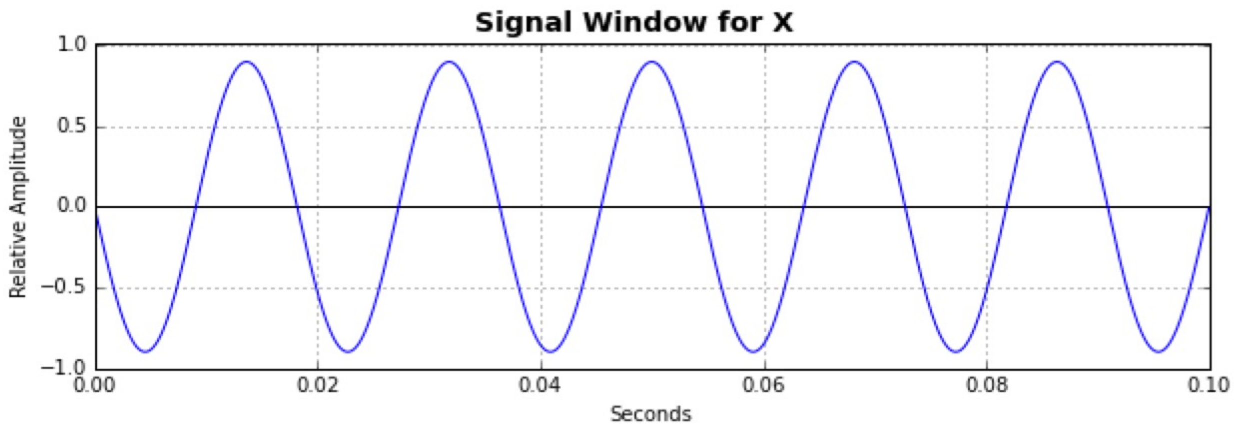
```
In [261]: X = au.makeSignal([(55,-0.9,0)],4410,"Samples")
     ...: au.displaySignal(X)
     ...:
```



Digital Audio Fundamentals: The Discrete Sine Transform

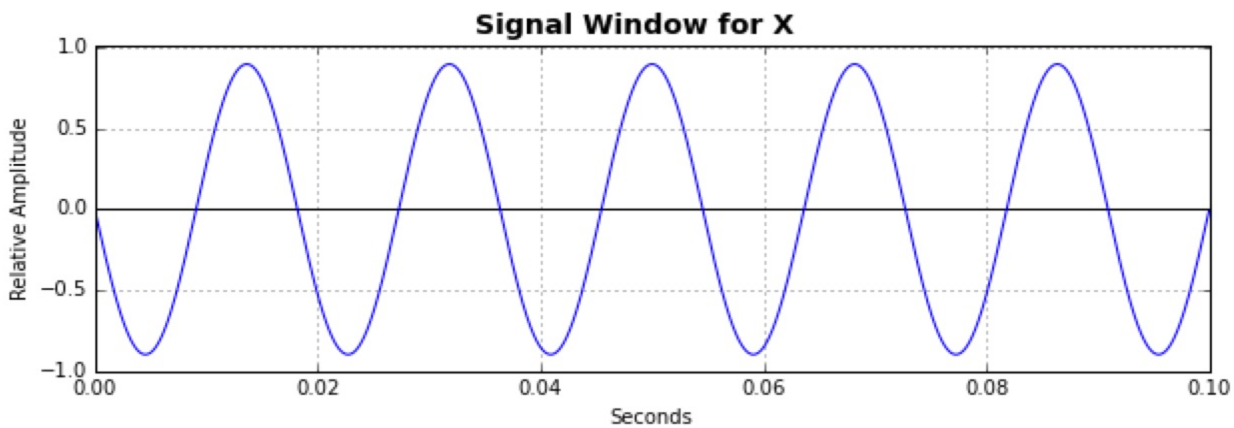
Computer Science

```
In [262]: X = au.makeSignal([(55,0.9,pi)],4410,"Samples")
     ...: au.displaySignal(X)
     ...:
```



The same effect can be gotten by delaying the phase by pi or by using a negative frequency: all will produce negative amplitudes.

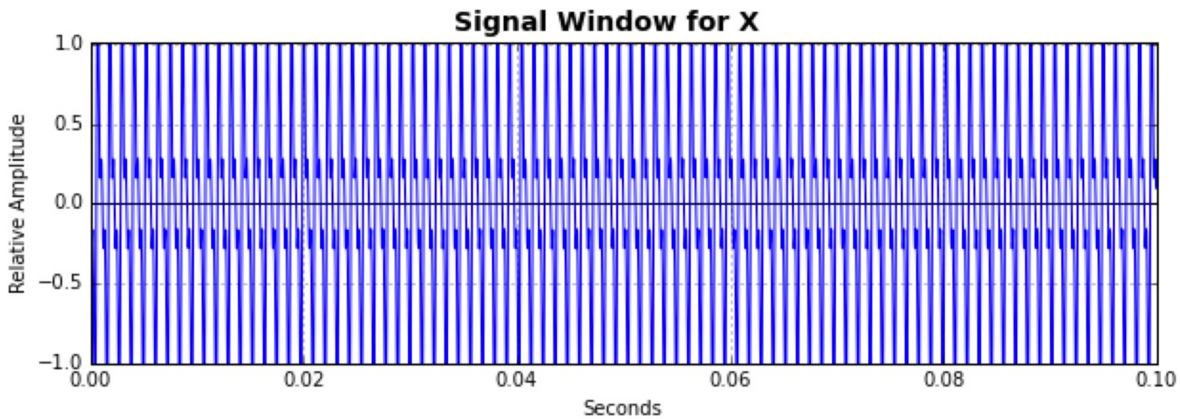
```
In [263]: X = au.makeSignal([(-55,0.9,0)],4410,"Samples")
     ...: au.displaySignal(X)
     ...:
```

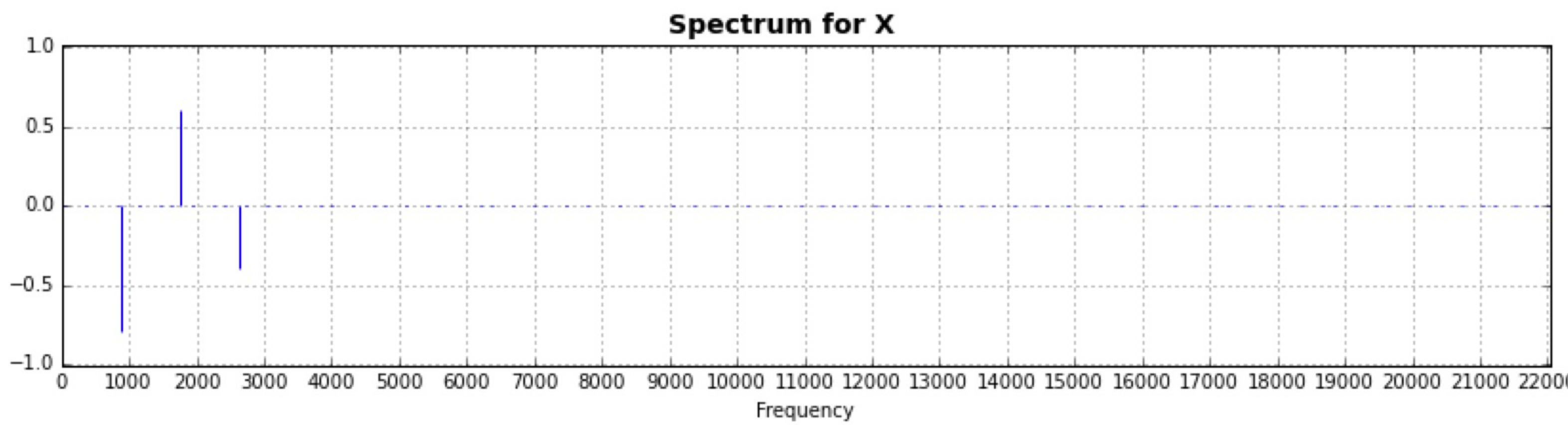


Digital Audio Fundamentals: The Discrete Sine Transform

Computer Science

Spectrum: [(880, 0.8, pi), (1760, 0.6, 0), (2640, 0.4, pi)]





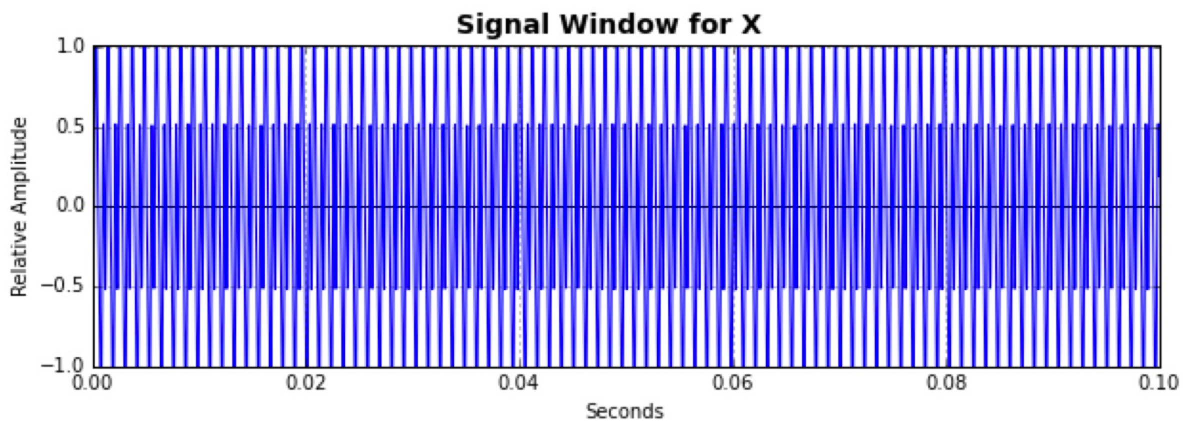
Freq	Amp
880.0	-0.8
1760.0	0.6
2640.0	-0.4

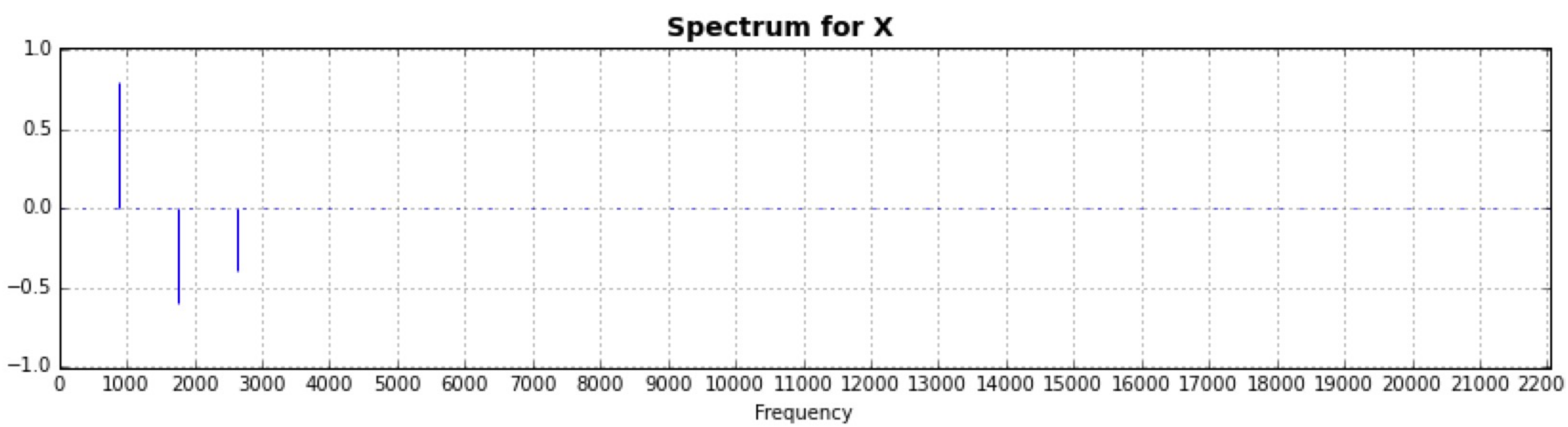
Interpreting Outputs from the Discrete Sine Transform:

Delaying a component by phase pi produces negative amplitudes.

Digital Audio Fundamentals: The Discrete Sine Transform

Spe Spectrum: [(880, 0.8, 0), (-1760, 0.6, 0), (-2640, 0.4, 0)]





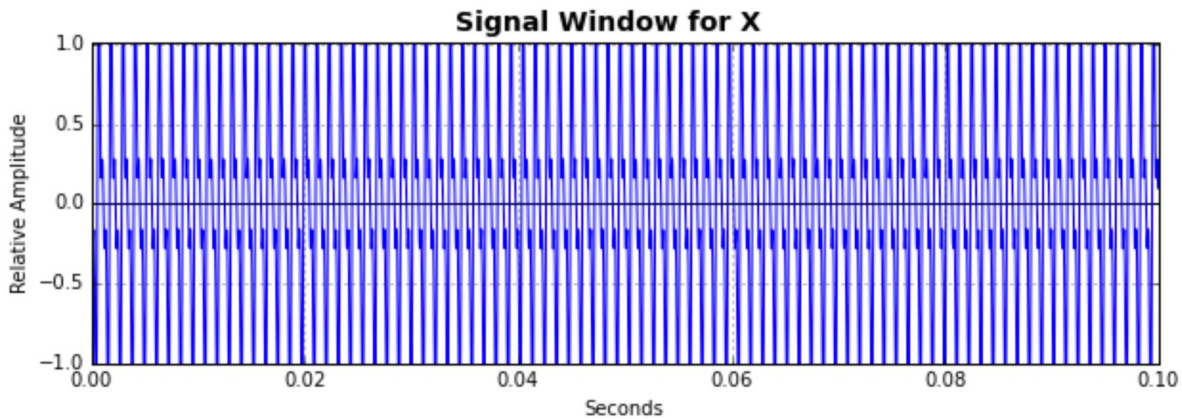
Freq	Amp
880.0	0.8
1760.0	−0.6
2640.0	−0.4

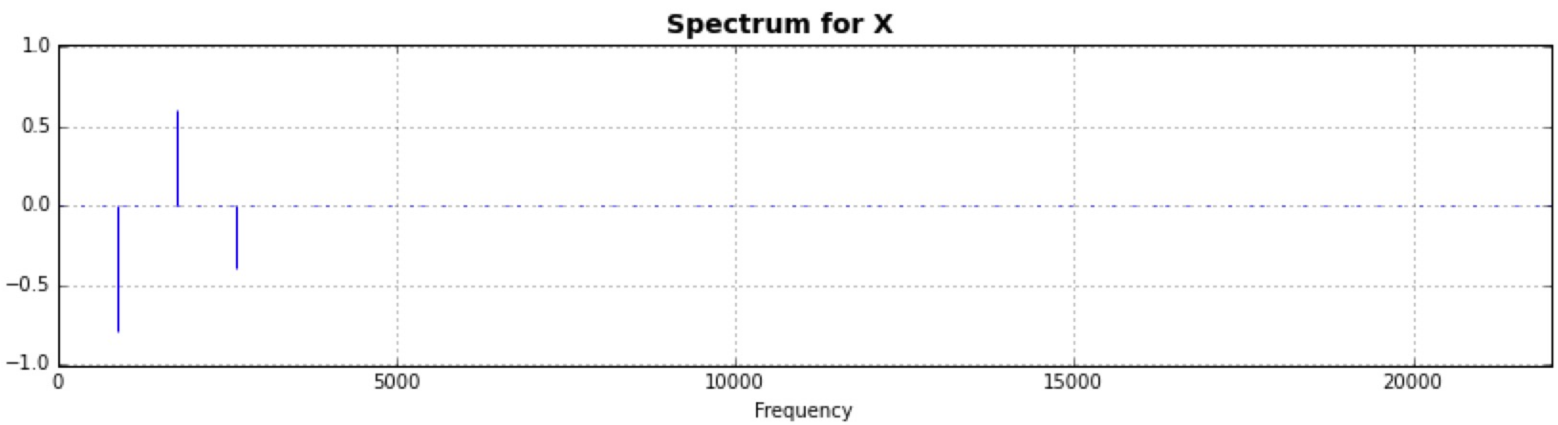
Interpreting Outputs from the Discrete Sine Transform:

Negative frequencies produce negative amplitudes.

Digital Audio Fundamentals: The Discrete Sine Transform

Spectrum: [(880, -0.8, 0), (-1760, -0.6, 0), (-2640, -0.4, pi)]





Freq	Amp
880.0	−0.8
1760.0	0.6
2640.0	−0.4

Interpreting Outputs from the Discrete Sine Transform:

Doing combinations of these will flip the amplitude back and forth:

Digital Audio Fundamentals: Discrete Sine Transform

Computer Science

There are three problems (so far):

(1) This is horribly inefficient: O(N^2) for N = len(X)

✓ Solution: There is a fast version of the transform, the Fast Fourier Transform (FFT), based on a recursive algorithm, which runs in O(N log(N)).

(2) The resolution is limited to multiples of f = SR / W (in samples)

✗ No solution, unfortunately, can try different window sizes, but stuck with this!

(3) All components and probe waves **have to be at the same phase (e.g., 0.0)**

✓ Solution: If we do all the work with **complex numbers**, we can avoid issues of phase

A brief summary of Complex Numbers on the board.....

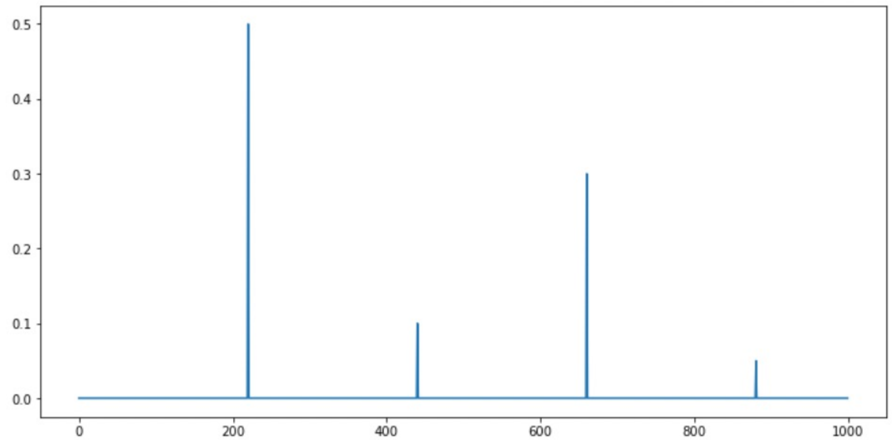
Digital Audio: The Discrete Fourier Transform

Computer Science

I have provided in the Intro Notebook an implementation of the FT which returns real results:

In [39]:
```
def realFFT(X):
    return 2*abs(np.fft.rfft(X))/len(X)

X = makeSignal(S=[(220,0.5,0),(440,0.1,0),(660,0.3,0),(880,0.05,0)])

S = realFFT(X)

print(S[220])
print(S[440])
print(S[660])
print(S[880])

print(len(X),len(S))

plt.figure(figsize=(12,6))
plt.plot(S[:1000])
plt.show()
```

```
0.49999999999999994
0.1
0.3000000000000003
0.05000000000000046
22050 11026
```



Digital Audio: The Discrete Fourier Transform

Computer Science

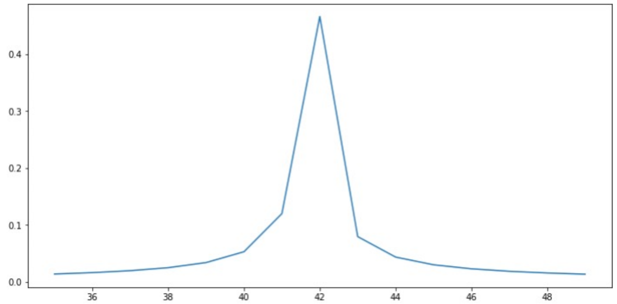
So, we have an efficient algorithm which does not care about phase, but problem 2 is still with us!

(2) The resolution is limited to multiples of f = SR / W (in samples)

✗ No solution, unfortunately, can try different window sizes, but stuck with this!

In [58]:
```
def realFFT(X):
    return 2*abs(np.fft.rfft(X))/len(X)

X = makeSignal(S=[(220,0.5,0)],duration = 0.19)

S = realFFT(X)

print("window frequency", round4(1/0.19))

print("220 =",220/(1/0.19), "*", round4(1/0.19))
print()

for i in range(35,50):
    print("S["+str(i)+"]:",S[i])

plt.figure(figsize=(12,6))
plt.plot(range(35,50),S[35:50])
plt.show()
```

```
window frequency 5.2632
220 = 41.8 * 5.2632

S[35]: 0.013844199359476984
S[36]: 0.01626521334271509
S[37]: 0.019698782735074l3
S[38]: 0.024945011066108816
S[39]: 0.03394908193120572
S[40]: 0.052990872882735976
S[41]: 0.11994678431300199
S[42]: 0.46639975281719664
S[43]: 0.07955360975791623
S[44]: 0.04359291600138813
S[45]: 0.03007436801890943
S[46]: 0.02298672244771206
S[47]: 0.018623282559498338
S[48]: 0.015666854011703157
S[49]: 0.013531516616080727
```



More on this next time!