# ECE 60146 Homework 1

## Dan Mani Binu

### January 2025

## 2.1 Creating the Base Class

In [125…

```
import math
class SignalProcessor ( object ):
    def __init__ ( self , data ):
        self . data = data
```

## 2.2 Creating the Sine Wave Function

*super()* helps initialize data through the parent class, while *amplitude* and *frequency* are local to the child class and updated within its ***init***

In [126…

```
class SineWaveFunction ( SignalProcessor ):
    def __init__ ( self , amplitude , frequency ):
        super () . __init__ ([])
        self . amplitude = amplitude
        self . frequency = frequency
```

## 2.3 Making Objects Callable

The _*call*_ function is invoked when the class name is used with the call operator *()*

In [127…

```
    def __call__(self,duration):
        self.data = []
        for i in range(duration):
            self.data.append(self.amplitude * math.sin(2*math.pi*self.freque
        print(f"Sine wave data:{self.data}")

    def __len__(self):
        return len(self.data)
```

## 2.4 Making Objects Iterable

The **iter** method resets the index to zero and returns the object itself, satisfying the iterator protocol. The **next** method retrieves the current value, advances the index, and raises StopIteration when the list is exhausted

In [128…

```
    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index >= len(self.data):
            raise StopIteration
        current_value = self.data[self.index]
        self.index += 1
        return current_value
```

## 2.6 Comparing Signals

The *eq* function here checks first whether the number of samples in *data* for both the signals are the same. If not it checks for the number of signals that are same with a tolerance of 0.01.

In [129…

```
    def __eq__(self,b):
        count = 0
        if len(self.data) != len(b.data):
            raise ValueError("Two signals are not equal in length")
        else:
            for i in range(len(self.data)):
                if abs(self.data[i]-b.data[i])<0.01:
                    count += 1
            return count
```

In [130…

```
class SineWaveFunction ( SignalProcessor ):
    def __init__ ( self , amplitude , frequency ):
        super () . __init__ ([])
        self . amplitude = amplitude
        self . frequency = frequency
    def __call__(self,duration):
        self.data = []
        for i in range(duration):
            self.data.append(self.amplitude * math.sin(2*math.pi*self.freque
        print(f"Sine wave data:{self.data}")

    def __len__(self):
        return len(self.data)

    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index >= len(self.data):
            raise StopIteration
        current_value = self.data[self.index]
        self.index += 1
        return current_value

    def __eq__(self,b):
        count = 0
        if len(self.data) != len(b.data):
            raise ValueError("Two signals are not equal in length")
        else:
            for i in range(len(self.data)):
                if abs(self.data[i]-b.data[i])<0.01:
                    count += 1
            return count
```

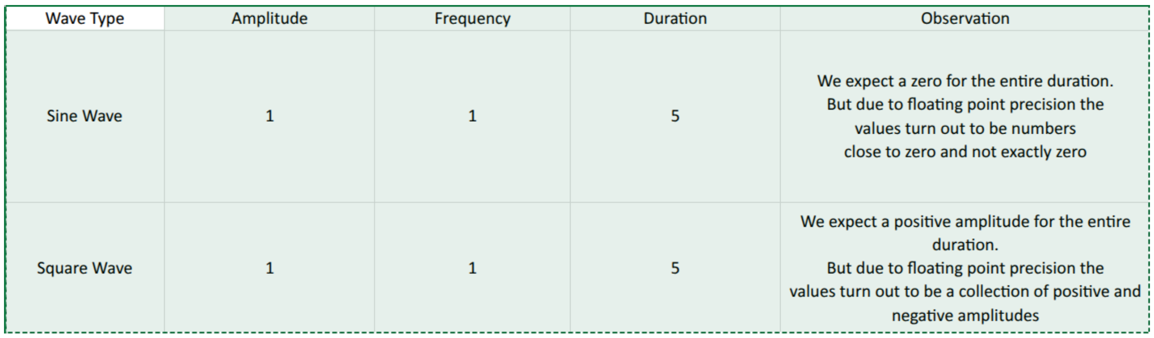
## 2.5 Square Wave Function:

*SquareWaveFunction* would inherit from *SineWaveFunction* as both of them have *amplitude*, *frequency* and *data* values. The data values generated from *SineWaveFunction* are then used to reinitialize data values for *SquareWaveFunction*

In [131…

```
class SquareWaveFunction ( SineWaveFunction ):
    def __init__ ( self , amplitude , frequency ):
        super (SquareWaveFunction,self) . __init__ ( 1 , frequency )
        self . amplitude = amplitude

    def __call__(self,duration):
        super(). __call__(duration)
        for i in range(len(self.data)):
            if self.data[i] >= 0:
                self.data[i] = self.amplitude
            else:
                self.data[i] = -self.amplitude
        print(f"Square wave data:{self.data}")
```

## 2.7 Demonstration:



| Wave Type | Amplitude | Frequency | Duration | Observation |
|---|---|---|---|---|
| Sine Wave | 1 | 1 | 5 | We expect a zero for the entire duration. But due to floating point precision the values turn out to be numbers close to zero and not exactly zero |
| Square Wave | 1 | 1 | 5 | We expect a positive amplitude for the entire duration. But due to floating point precision the values turn out to be a collection of positive and negative amplitudes |

In [132…

```
SG = SineWaveFunction(amplitude =1.0 ,frequency =1)
SG (duration = 5)
SG = SquareWaveFunction(amplitude =3.0 ,frequency =1)
SG (duration = 5)
```

```
Sine wave data:[0.0, -2.4492935982947064e-16, -4.898587196589413e-16, -7.347
880794884119e-16, -9.797174393178826e-16]
Sine wave data:[0.0, -7.347880794884119e-16, -1.4695761589768238e-15, -2.204
364238465236e-15, -2.9391523179536475e-15]
Square wave data:[3.0, -3.0, -3.0, -3.0, -3.0]
```

| | | | | |
|---|---|---|---|---|
| Sine Wave | 1 | 1.1 | 10 | Both return same values. This is called Aliasing. Because you are only looking once per integer,a wave spinning at 1.1 loops per second looks exactly like a wave spinning at 0.1 loops per second. The computer "misses" the extra full loop. |
| | 1 | 0.1 | 10 | |

In [133…

```
SG = SineWaveFunction(amplitude=1.0, frequency=1.1)
SG(duration=10)
SG = SineWaveFunction(amplitude=1.0, frequency=0.1)
SG(duration=10)
```

```
Sine wave data:[0.0, 0.5877852522924736, 0.951056516295154, 0.95105651629515
28, 0.5877852522924711, -2.2056021997384123e-15, -0.5877852522924776, -0.951
0565162951541, -0.9510565162951521, -0.5877852522924665]
Sine wave data:[0.0, 0.5877852522924731, 0.9510565162951535, 0.9510565162951
536, 0.5877852522924732, 1.2246467991473532e-16, -0.587785252292473, -0.9510
565162951535, -0.9510565162951536, -0.5877852522924734]
```

## 3.1 Composite signal function

Here, we define a class to handle signal addition.

The CompositeSignalFunction accepts a list of other signal objects (like sine or square waves) and sums them up.

The **call** method handles the logic in two steps: first, it ensures all input waves are generated for the given duration, and then it accumulates their values into a single data list to create the composite output.

In [134…

```
class CompositeSignalFunction ( SignalProcessor ):
    def __init__ ( self , inputs ):
        super () . __init__ ([])
        self.inputs = inputs

    def __call__(self,duration):
        self.data = [0]*duration
        for waves in self.inputs:
            waves(duration)
        for i in range(duration):
            for waves in self.inputs:
                self.data[i]+=waves.data[i]
        print(f"Composite signal data:{self.data}")
```

In [135…
```
SG = SineWaveFunction ( amplitude = 1.0 , frequency = 0.1)
SW = SquareWaveFunction ( amplitude = 0.5 , frequency = 0.05)
CSG = CompositeSignalFunction ( inputs = [ SG , SW ] )
CSG ( duration = 10 )
```

```
Sine wave data:[0.0, 0.5877852522924731, 0.9510565162951535, 0.9510565162951
536, 0.5877852522924732, 1.2246467991473532e-16, -0.587785252292473, -0.9510
565162951535, -0.9510565162951536, -0.5877852522924734]
Sine wave data:[0.0, 0.15450849718747370, 0.29389262614623657, 0.404508497187
4737, 0.47552825814757677, 0.5, 0.4755282581475768, 0.4045084971874737, 0.29
38926261462366, 0.15450849718747375]
Square wave data:[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
Composite signal data:[0.5, 1.0877852522924731, 1.4510565162951536, 1.451056
5162951536, 1.0877852522924734, 0.5000000000000001, -0.08778525229247303, -
0.45105651629515353, -0.45105651629515364, -0.08778525229247336]
```

## 3.2 Signal Visualization

The following code creates a visualization for all the signals generated, step by step, in code cells:

First we generate all three signals for a duration of 80 timesteps

In [136…
```
SG = SineWaveFunction ( amplitude = 1.0 , frequency = 0.1)
SW = SquareWaveFunction ( amplitude = 0.5 , frequency = 0.05)
CSG = CompositeSignalFunction ( inputs = [ SG , SW ] )
CSG ( duration = 80 )
```

```
Sine wave data:[0.0, 0.5877852522924731, 0.9510565162951535, 0.9510565162951
536, 0.5877852522924732, 1.2246467991473532e-16, -0.587785252292473, -0.9510
565162951535, -0.9510565162951536, -0.5877852522924734, -2.4492935982947064e
-16, 0.5877852522924729, 0.9510565162951535, 0.9510565162951536, 0.587785252
2924734, 3.6739403974420594e-16, -0.5877852522924728, -0.9510565162951534, -
0.9510565162951538, -0.5877852522924735, -4.898587196589413e-16, 0.587785252
2924727, 0.9510565162951534, 0.9510565162951538, 0.5877852522924736, 6.12323
3995736766e-16, -0.5877852522924726, -0.9510565162951534, -0.951056516295153
8, -0.5877852522924737, -7.347880794884119e-16, 0.5877852522924725, 0.951056
5162951533, 0.9510565162951539, 0.5877852522924738, 8.572527594031472e-16, -
0.5877852522924725, -0.9510565162951533, -0.9510565162951539, -0.58778525229
24739, -9.797174393178826e-16, 0.5877852522924724, 0.9510565162951533, 0.951
0565162951539, 0.587785252292474, 1.102182119232618e-15, -0.587785252292472
2, -0.9510565162951532, -0.951056516295154, -0.5877852522924741, -1.22464679
91473533e-15, 0.5877852522924693, 0.9510565162951532, 0.9510565162951529, 0.
5877852522924742, 4.899825157862589e-15, -0.587785252292472, -0.951056516295
1542, -0.951056516295154, -0.5877852522924771, -1.4695761589768238e-15, 0.58
77852522924748, 0.9510565162951531, 0.9510565162951552, 0.5877852522924744,
-1.9606728399089416e-15, -0.5877852522924718, -0.951056516295152, -0.9510565
162951541, -0.5877852522924716, -1.7145055188062944e-15, 0.5877852522924688,
0.951056516295153, 0.951056516295153, 0.5877852522924746, 5.3896838775215305
e-15, -0.5877852522924716, -0.9510565162951541, -0.9510565162951542, -0.5877
852522924776]
Sine wave data:[0.0, 0.1545084971874737, 0.29389262614623657, 0.404508497187
4737, 0.47552825814757677, 0.5, 0.4755282581475768, 0.4045084971874737, 0.29
38926261462366, 0.15450849718747375, 6.123233995736766e-17, -0.1545084971874
7364, -0.2938926261462365, -0.40450849718747367, -0.47552825814757677, -0.5,
-0.4755282581475768, -0.4045084971874738, -0.2938926261462367, -0.1545084971
874738, -1.2246467991473532e-16, 0.1545084971874736, 0.29389262614623646, 0.
4045084971874736, 0.47552825814757677, 0.5, 0.4755282581475768, 0.4045084971
8747384, 0.2938926261462367, 0.1545084971874739, 1.8369701987210297e-16, -0.
15450849718747353, -0.2938926261462364, -0.4045084971874736, -0.475528258147
5767, -0.5, -0.4755282581475769, -0.40450849718747384, -0.29389262614623674,
-0.15450849718747395, -2.4492935982947064e-16, 0.15450849718747348, 0.293892
62614623635, 0.40450849718747356, 0.4755282581475767, 0.5, 0.475528258147576
9, 0.4045084971874739, 0.2938926261462368, 0.154508497187474, 3.061616997868
383e-16, -0.15450849718747256, -0.2938926261462363, -0.40450849718747406, -
0.4755282581475767, -0.5, -0.4755282581475769, -0.4045084971874734, -0.29389
262614623685, -0.1545084971874749, -3.6739403974420594e-16, 0.15450849718747
42, 0.29389262614623624, 0.40450849718747295, 0.47552825814757665, 0.5, 0.47
552825814757693, 0.4045084971874745, 0.2938926261462369, 0.1545084971874732
8, 4.286263797015736e-16, -0.15450849718747245, -0.29389262614623624, -0.404
50849718747395, -0.47552825814757665, -0.5, -0.47552825814757693, -0.4045084
9718747345, -0.29389262614623696, -0.15450849718747503]
Square wave data:[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.
5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.
5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5,
-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.
5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5]
Composite signal data:[0.5, 1.0877852522924731, 1.4510565162951536, 1.451056
5162951536, 1.0877852522924734, 0.5000000000000001, -0.08778525229247303, -
0.45105651629515353, -0.45105651629515364, -0.08778525229247336, 0.499999999
9999998, 0.08778525229247292, 0.45105651629515353, 0.45105651629515364, 0.08
778525229247336, -0.4999999999999996, -1.087785252292473, -1.451056516295153
4, -1.4510565162951536, -1.0877852522924734, -0.5000000000000004, 1.08778525
22924727, 1.4510565162951534, 1.4510565162951536, 1.0877852522924736, 0.5000
000000000007, -0.08778525229247258, -0.4510565162951534, -0.4510565162951537
5, -0.08778525229247369, 0.4999999999999993, 0.08778525229247247, 0.45105651
62951533, 0.45105651629515386, 0.0877852522924738, -0.49999999999999917, -1.
0877852522924725, -1.4510565162951532, -1.4510565162951539, -1.0877852522924
738, -0.500000000000001, 1.0877852522924725, 1.4510565162951532, 1.451056516
2951539, 1.087785252292474, 0.5000000000000011, -0.08778525229247225, -0.451
0565162951532, -0.451056516295154, -0.08778525229247414, 0.4999999999999988,
0.08778525229246925, 0.4510565162951532, 0.45105651629515287, 0.087785252292
47425, -0.4999999999999951, -1.087785252292472, -1.451056516295154, -1.45105
6516295154, -1.0877852522924771, -0.5000000000000014, 1.0877852522924747, 1.
4510565162951532, 1.4510565162951552, 1.0877852522924742, 0.4999999999999980
6, -0.0877852522924718, -0.451056516295152, -0.4510565162951541, -0.08778525
229247158, 0.4999999999999983, 0.08778525229246881, 0.451056516295153, 0.451
056516295153, 0.08778525229247458, -0.4999999999999946, -1.0877852522924716,
-1.451056516295154, -1.451056516295154, -1.0877852522924776]
```

This helper function simplifies the process of converting our time-domain signals into the frequency domain. It uses

numpy.fft.fft to compute the raw Fourier Transform. Since the FFT output for real-valued signals is symmetric (the second half

is a mirror of the first), we slice the array to keep only the first half (positive frequencies) and calculate the absolute

magnitude to make it ready for plotting.

In [137…
```python
def get_fft_spectrum(signal_data):
    fft_vals = np.fft.fft(signal_data)
    # We only need the first half (positive frequencies)
    half_n = len(signal_data) // 2
    magnitude = np.abs(fft_vals)[:half_n]
    return magnitude
```

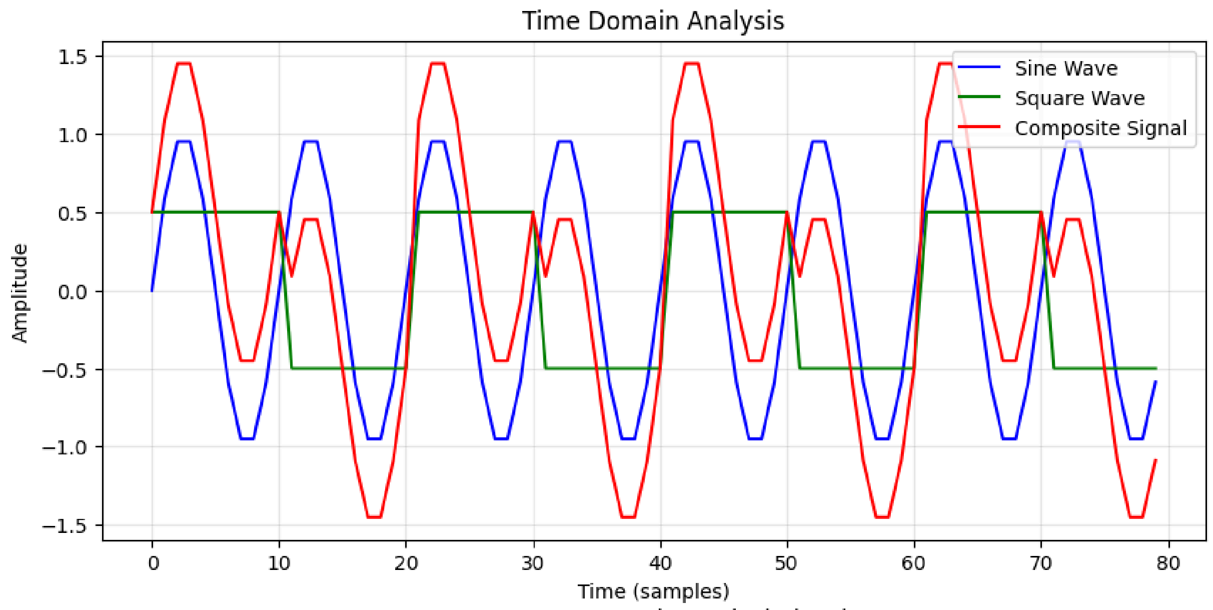
We now create a canvas for two plots in matplotlib, one for the time domain analysis and the other for frequency domain analysis.

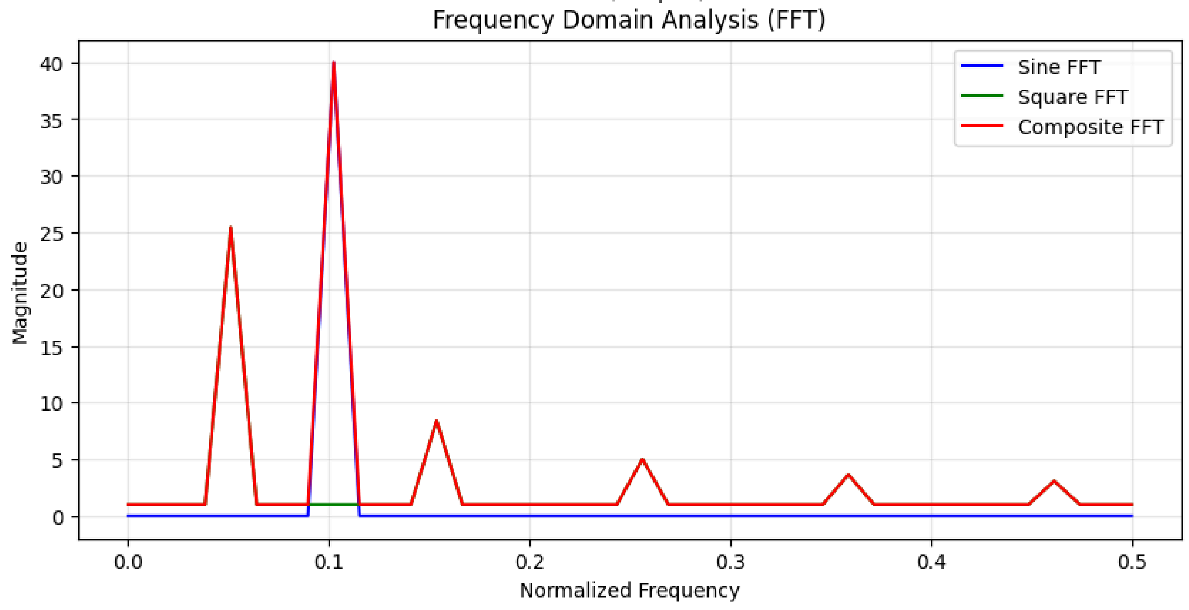
In this step, we compute the FFT magnitude for the sine, square, and composite signals using our helper function. We then plot these spectrums against a normalized frequency axis (ranging from 0 to 0.5). This visualization allows us to clearly observe the fundamental frequency in the sine wave, the odd harmonics inherent in the square wave, and how these components stack together in the composite signal.

The waves are plotted with distinct colors

In [138…
```python
import matplotlib.pyplot as plt
import numpy as np
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))

time_axis = range(80)
```

```
ax1.plot(time_axis, SG.data, label='Sine Wave', color='blue')
ax1.plot(time_axis, SW.data, label='Square Wave', color='green')
ax1.plot(time_axis, CSG.data, label='Composite Signal', color='red')

ax1.set_title('Time Domain Analysis')
ax1.set_xlabel('Time (samples)')
ax1.set_ylabel('Amplitude')
ax1.legend(loc='upper right')
ax1.grid(True, alpha=0.3)


fft_sine = get_fft_spectrum(SG.data)
fft_square = get_fft_spectrum(SW.data)
fft_composite = get_fft_spectrum(CSG.data)

# Create frequency axis (Normalized Frequency 0 to 0.5)
freqs = np.linspace(0, 0.5, len(fft_sine))

ax2.plot(freqs, fft_sine, label='Sine FFT', color='blue')
ax2.plot(freqs, fft_square, label='Square FFT', color='green')
ax2.plot(freqs, fft_composite, label='Composite FFT', color='red')

ax2.set_title('Frequency Domain Analysis (FFT)')
ax2.set_xlabel('Normalized Frequency')
ax2.set_ylabel('Magnitude')
ax2.legend(loc='upper right')
ax2.grid(True, alpha=0.3)

plt.show()
```





### Time Domain (Top Plot):

This graph shows the application of the principle of superposition. The Composite Signal (Red) is simply the sum of the Sine

(Blue) wave and the Square (Green) wave. It can clearly be noticed that the red graph maintains the sinus nature of the sine

wave. However, it is displaced upwards or downwards depending on the state of the square wave.

### Frequency Domain (Bottom Plot):

The following graph shows what the signals look like in terms of their frequency components.

The large red spike (around 0.1) corresponds to the fundamental, common to all three signals.

The Sine Wave (Blue) is a pure wave and therefore has one spike and no harmonics.

## Codebase

In [139…

```
import math

class SignalProcessor ( object ):
    def __init__ ( self , data ):
        self . data = data

class SineWaveFunction ( SignalProcessor ):
    def __init__ ( self , amplitude , frequency ):
        super () . __init__ ([])
        self . amplitude = amplitude
        self . frequency = frequency
    def __call__(self,duration):
        self.data = []
        for i in range(duration):
            self.data.append(self.amplitude * math.sin(2*math.pi*self.freque
        print(f"Sine wave data:{self.data}")

    def __len__(self):
        return len(self.data)

    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index >= len(self.data):
            raise StopIteration
        current_value = self.data[self.index]
        self.index += 1
        return current_value

    def __eq__(self,b):
        count = 0
        if len(self.data) != len(b.data):
            raise ValueError("Two signals are not equal in length")
        else:
            for i in range(len(self.data)):
                if abs(self.data[i]-b.data[i])<0.01:
                    count += 1
            return count

class SquareWaveFunction ( SineWaveFunction ):
    def __init__ ( self , amplitude , frequency ):
        super (SquareWaveFunction,self) . __init__ ( 1 , frequency )
        self . amplitude = amplitude
    def __call__(self,duration):
        super(). __call__(duration)
        for i in range(len(self.data)):
            if self.data[i] >= 0:
                self.data[i] = self.amplitude
            else:
                self.data[i] = -self.amplitude
        print(f"Square wave data:{self.data}")
```

In [140…

```
SG1 = SineWaveFunction(amplitude =2.0 ,frequency =0.1)
SG1 (duration = 5)
```

```
Sine wave data:[0.0, 1.1755705045849463, 1.902113032590307, 1.90211303259030
73, 1.1755705045849465]
```

In [141…

```
SG = SineWaveFunction(amplitude =2.0 ,frequency =0.1)
SG ( duration =5 ) # Create and print the sine wave
print (len( SG ) )
```

```
Sine wave data:[0.0, 1.1755705045849463, 1.902113032590307, 1.90211303259030
73, 1.1755705045849465]
5
```

In [142…

```
SG = SineWaveFunction(amplitude =2.0 ,frequency =0.1)
SG(duration =5 )
print([val for val in SG])
```

```
Sine wave data:[0.0, 1.1755705045849463, 1.902113032590307, 1.90211303259030
73, 1.1755705045849465]
[0.0, 1.1755705045849463, 1.902113032590307, 1.9021130325903073, 1.175570504
5849465]
```

In [143…

```
SW = SquareWaveFunction(amplitude =3.0 , frequency =0.1)
SW(duration =5 )
print(len( SW ) ) # Output : 5
print([val for val in SW])
```

```
Sine wave data:[0.0, 1.7633557568774194, 2.8531695488854605, 2.8531695488854
61, 1.7633557568774196]
Square wave data:[3.0, 3.0, 3.0, 3.0, 3.0]
5
[3.0, 3.0, 3.0, 3.0, 3.0]
```

In [144…

```
SG1 = SineWaveFunction (amplitude =2.0 , frequency =0.1)
SG1 (duration =5)
SG2 = SineWaveFunction ( amplitude =2.0 , frequency =0.15 )
SG2 ( duration =5 )
print ( SG1 == SG2 ) # Output : number of matching elements
SG3 = SineWaveFunction ( amplitude =2.0 , frequency =0.1 )
SG3(duration =3)
print(SG1 == SG3)
```

```
Sine wave data:[0.0, 1.1755705045849463, 1.902113032590307, 1.90211303259030
73, 1.1755705045849465]
Sine wave data:[0.0, 1.618033988749895, 1.9021130325903073, 0.61803398874989
5, -1.175570504584946]
2
Sine wave data:[0.0, 1.1755705045849463, 1.902113032590307]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[144], line 8
      6 SG3 = SineWaveFunction ( amplitude =2.0 , frequency =0.1 )
      7 SG3(duration =3)
----> 8 print(SG1 == SG3)

Cell In[139], line 35, in SineWaveFunction.__eq__(self, b)
     33 count = 0
     34 if len(self.data) != len(b.data):
---> 35     raise ValueError("Two signals are not equal in length")
     36 else:
     37     for i in range(len(self.data)):

ValueError: Two signals are not equal in length
```

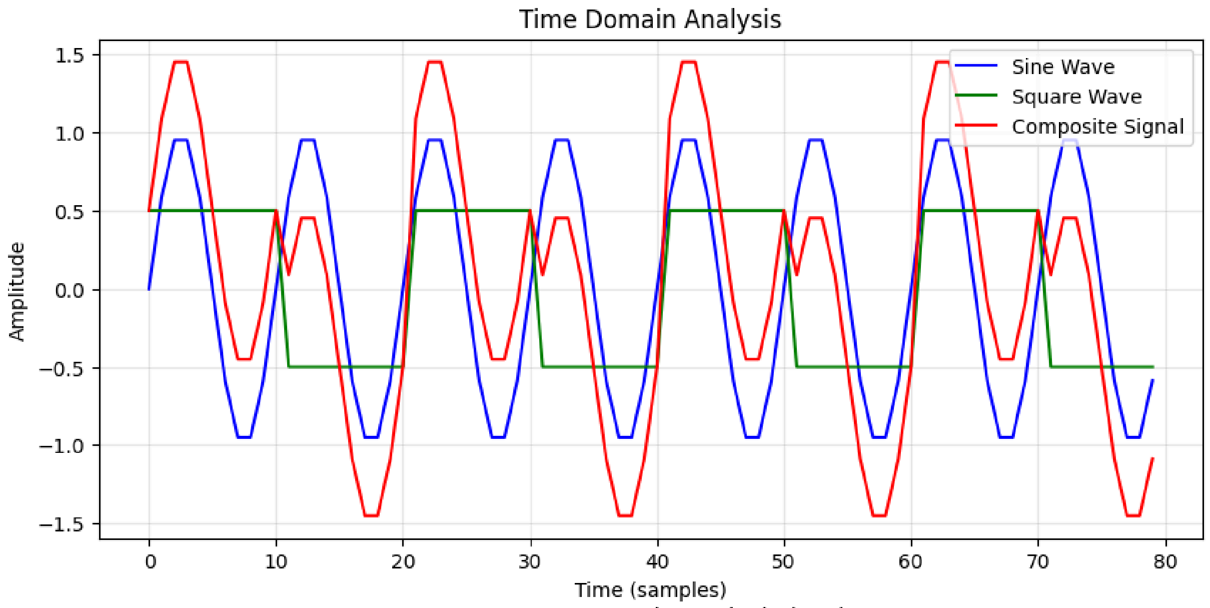
## Bonus Section

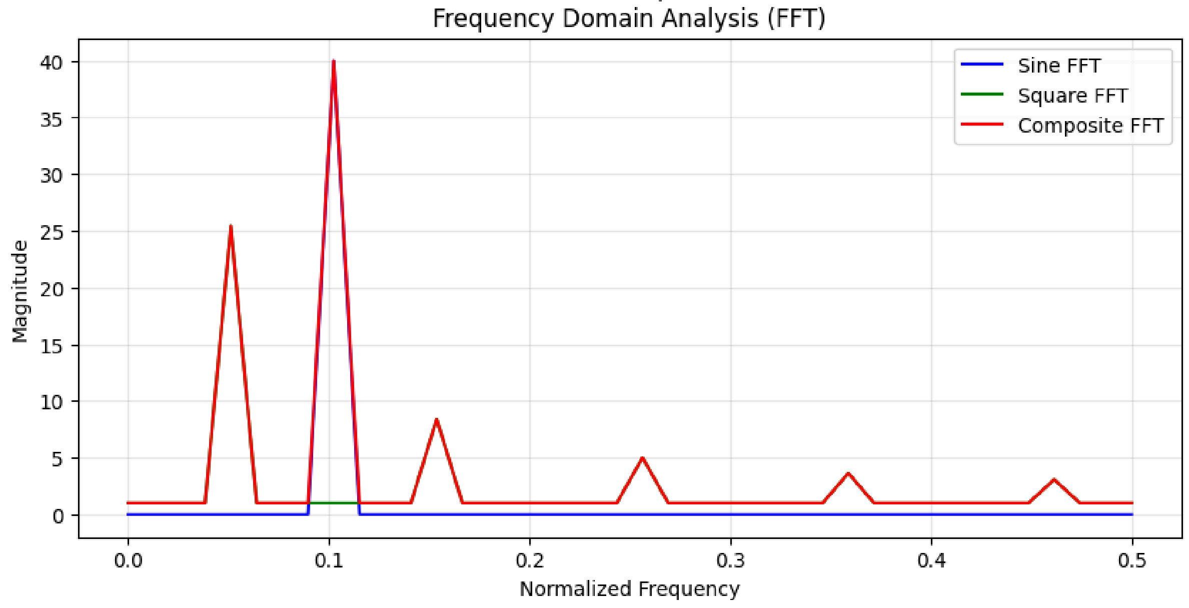
In [ ]:
```python
class CompositeSignalFunction ( SignalProcessor ):
    def __init__ ( self , inputs ):
        super () . __init__ ([])
        self.inputs = inputs

    def __call__(self,duration):
        self.data = [0]*duration
        for waves in self.inputs:
            waves(duration)
        for i in range(duration):
            for waves in self.inputs:
                self.data[i]+=waves.data[i]
        print(f"Composite signal data:{self.data}")
```

In [145…
```python
SG = SineWaveFunction ( amplitude = 1.0 , frequency = 0.1)
SW = SquareWaveFunction ( amplitude = 0.5 , frequency = 0.05)
CSG = CompositeSignalFunction ( inputs = [ SG , SW ] )
CSG ( duration = 80 )

def get_fft_spectrum(signal_data):
    fft_vals = np.fft.fft(signal_data)
    # We only need the first half (positive frequencies)
    half_n = len(signal_data) // 2
    magnitude = np.abs(fft_vals)[:half_n]
    return magnitude

import matplotlib.pyplot as plt
import numpy as np
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 10))
```

```python
time_axis = range(80)

ax1.plot(time_axis, SG.data, label='Sine Wave', color='blue')
ax1.plot(time_axis, SW.data, label='Square Wave', color='green')
ax1.plot(time_axis, CSG.data, label='Composite Signal', color='red')

ax1.set_title('Time Domain Analysis')
ax1.set_xlabel('Time (samples)')
ax1.set_ylabel('Amplitude')
ax1.legend(loc='upper right')
ax1.grid(True, alpha=0.3)


fft_sine = get_fft_spectrum(SG.data)
fft_square = get_fft_spectrum(SW.data)
fft_composite = get_fft_spectrum(CSG.data)

# Create frequency axis (Normalized Frequency 0 to 0.5)
freqs = np.linspace(0, 0.5, len(fft_sine))

ax2.plot(freqs, fft_sine, label='Sine FFT', color='blue')
ax2.plot(freqs, fft_square, label='Square FFT', color='green')
ax2.plot(freqs, fft_composite, label='Composite FFT', color='red')

ax2.set_title('Frequency Domain Analysis (FFT)')
ax2.set_xlabel('Normalized Frequency')
ax2.set_ylabel('Magnitude')
ax2.legend(loc='upper right')
ax2.grid(True, alpha=0.3)

plt.show()
```

```
Sine wave data:[0.0, 0.5877852522924731, 0.9510565162951535, 0.9510565162951
536, 0.5877852522924732, 1.2246467991473532e-16, -0.587785252292473, -0.9510
565162951535, -0.9510565162951536, -0.5877852522924734, -2.4492935982947064e
-16, 0.5877852522924729, 0.9510565162951535, 0.9510565162951536, 0.587785252
2924734, 3.6739403974420594e-16, -0.5877852522924728, -0.9510565162951534, -
0.9510565162951538, -0.5877852522924735, -4.898587196589413e-16, 0.587785252
2924727, 0.9510565162951534, 0.9510565162951538, 0.5877852522924736, 6.12323
3995736766e-16, -0.5877852522924726, -0.9510565162951534, -0.951056516295153
8, -0.5877852522924737, -7.347880794884119e-16, 0.5877852522924725, 0.951056
5162951533, 0.9510565162951539, 0.5877852522924738, 8.572527594031472e-16, -
0.5877852522924725, -0.9510565162951533, -0.9510565162951539, -0.58778525229
24739, -9.797174393178826e-16, 0.5877852522924724, 0.9510565162951533, 0.951
0565162951539, 0.587785252292474, 1.102182119232618e-15, -0.587785252292472
2, -0.9510565162951532, -0.951056516295154, -0.5877852522924741, -1.22464679
91473533e-15, 0.5877852522924693, 0.9510565162951532, 0.9510565162951529, 0.
5877852522924742, 4.899825157862589e-15, -0.587785252292472, -0.951056516295
1542, -0.951056516295154, -0.5877852522924771, -1.4695761589768238e-15, 0.58
77852522924748, 0.9510565162951531, 0.9510565162951552, 0.5877852522924744,
-1.9606728399089416e-15, -0.5877852522924718, -0.951056516295152, -0.9510565
162951541, -0.5877852522924716, -1.7145055188062944e-15, 0.5877852522924688,
0.951056516295153, 0.951056516295153, 0.5877852522924746, 5.3896838775215305
e-15, -0.5877852522924716, -0.9510565162951541, -0.9510565162951542, -0.5877
852522924776]
Sine wave data:[0.0, 0.1545084971874737, 0.29389262614623657, 0.404508497187
4737, 0.47552825814757677, 0.5, 0.4755282581475768, 0.4045084971874737, 0.29
38926261462366, 0.15450849718747375, 6.123233995736766e-17, -0.1545084971874
7364, -0.2938926261462365, -0.40450849718747367, -0.47552825814757677, -0.5,
-0.4755282581475768, -0.4045084971874738, -0.2938926261462367, -0.1545084971
874738, -1.2246467991473532e-16, 0.1545084971874736, 0.29389262614623646, 0.
4045084971874736, 0.47552825814757677, 0.5, 0.4755282581475768, 0.4045084971
8747384, 0.2938926261462367, 0.1545084971874739, 1.8369701987210297e-16, -0.
15450849718747353, -0.2938926261462364, -0.4045084971874736, -0.475528258147
5767, -0.5, -0.4755282581475769, -0.40450849718747384, -0.29389262614623674,
-0.15450849718747395, -2.4492935982947064e-16, 0.15450849718747348, 0.293892
62614623635, 0.40450849718747356, 0.4755282581475767, 0.5, 0.475528258147576
9, 0.4045084971874739, 0.2938926261462368, 0.154508497187474, 3.061616997868
383e-16, -0.15450849718747256, -0.2938926261462363, -0.40450849718747406, -
0.4755282581475767, -0.5, -0.4755282581475769, -0.4045084971874734, -0.29389
262614623685, -0.1545084971874749, -3.6739403974420594e-16, 0.15450849718747
42, 0.29389262614623624, 0.40450849718747295, 0.47552825814757665, 0.5, 0.47
552825814757693, 0.4045084971874745, 0.2938926261462369, 0.154508497187473
28, 4.286263797015736e-16, -0.15450849718747245, -0.29389262614623624, -0.404
50849718747395, -0.47552825814757665, -0.5, -0.47552825814757693, -0.4045084
9718747345, -0.29389262614623696, -0.15450849718747503]
Square wave data:[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.
5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.
5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5,
-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.
5, 0.5, 0.5, 0.5, 0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5]
Composite signal data:[0.5, 1.0877852522924731, 1.4510565162951536, 1.451056
5162951536, 1.0877852522924734, 0.5000000000000001, -0.08778525229247303, -
0.45105651629515353, -0.45105651629515364, -0.08778525229247336, 0.499999999
9999998, 0.08778525229247292, 0.45105651629515353, 0.45105651629515364, 0.08
778525229247336, -0.4999999999999996, -1.087785252292473, -1.451056516295153
4, -1.4510565162951536, -1.0877852522924734, -0.5000000000000004, 1.08778525
22924727, 1.4510565162951534, 1.4510565162951536, 1.0877852522924736, 0.5000
000000000007, -0.08778525229247258, -0.4510565162951534, -0.4510565162951537
5, -0.08778525229247369, 0.4999999999999993, 0.08778525229247247, 0.45105651
62951533, 0.45105651629515386, 0.0877852522924738, -0.49999999999999917, -1.
0877852522924725, -1.4510565162951532, -1.4510565162951539, -1.0877852522924
738, -0.500000000000001, 1.0877852522924725, 1.4510565162951532, 1.451056516
2951539, 1.087785252292474, 0.5000000000000011, -0.08778525229247225, -0.451
0565162951532, -0.451056516295154, -0.08778525229247414, 0.4999999999999988,
0.08778525229246925, 0.4510565162951532, 0.45105651629515287, 0.087785252292
47425, -0.4999999999999951, -1.087785252292472, -1.451056516295154, -1.45105
6516295154, -1.0877852522924771, -0.5000000000000014, 1.0877852522924747, 1.
4510565162951532, 1.4510565162951552, 1.0877852522924742, 0.4999999999999980
6, -0.0877852522924718, -0.451056516295152, -0.4510565162951541, -0.08778525
229247158, 0.4999999999999983, 0.08778525229246881, 0.451056516295153, 0.451
056516295153, 0.08778525229247458, -0.4999999999999946, -1.0877852522924716,
-1.451056516295154, -1.451056516295154, -1.0877852522924776]
```


Time Domain Analysis
Sine Wave
Square Wave
Composite Signal
Amplitude
Time (samples)


Frequency Domain Analysis (FFT)
Sine FFT
Square FFT
Composite FFT
Magnitude
Normalized Frequency