Project 2: PWM Spectra

By Dillon Labonte

EE201 Signals and Systems

Professor Tim Monk

Spring 2024

To begin a Fourier Series representation of a PWM signal, we must first understand what a PWM signal is. Pulse Width Modulation (PWM) is a pulse train signal used to express analog signals digitally. The percentage of time that the signal is high is referred to as the duty cycle.

$$D = \frac{T_1}{T_0} \tag{1}$$

Where T_1 is the time in seconds that the signal is high, and T_0 is the fundamental period in seconds of the signal. For this project, T_0 is defined to be:

$$T_0 = \frac{1}{f_0} \tag{2}$$

Where f_0 is the fundamental frequency of the signal in Hertz based on the student's ID number rounded to three significant figures. An ID number of 2258458 rounded to 3 significant figures is 2260000, therefore $f_0 = 2260000 \text{ Hz}$ and $T_0 \approx 0.44 \ \mu s$.

Using these definitions, we can now define a PWM signal mathematically:

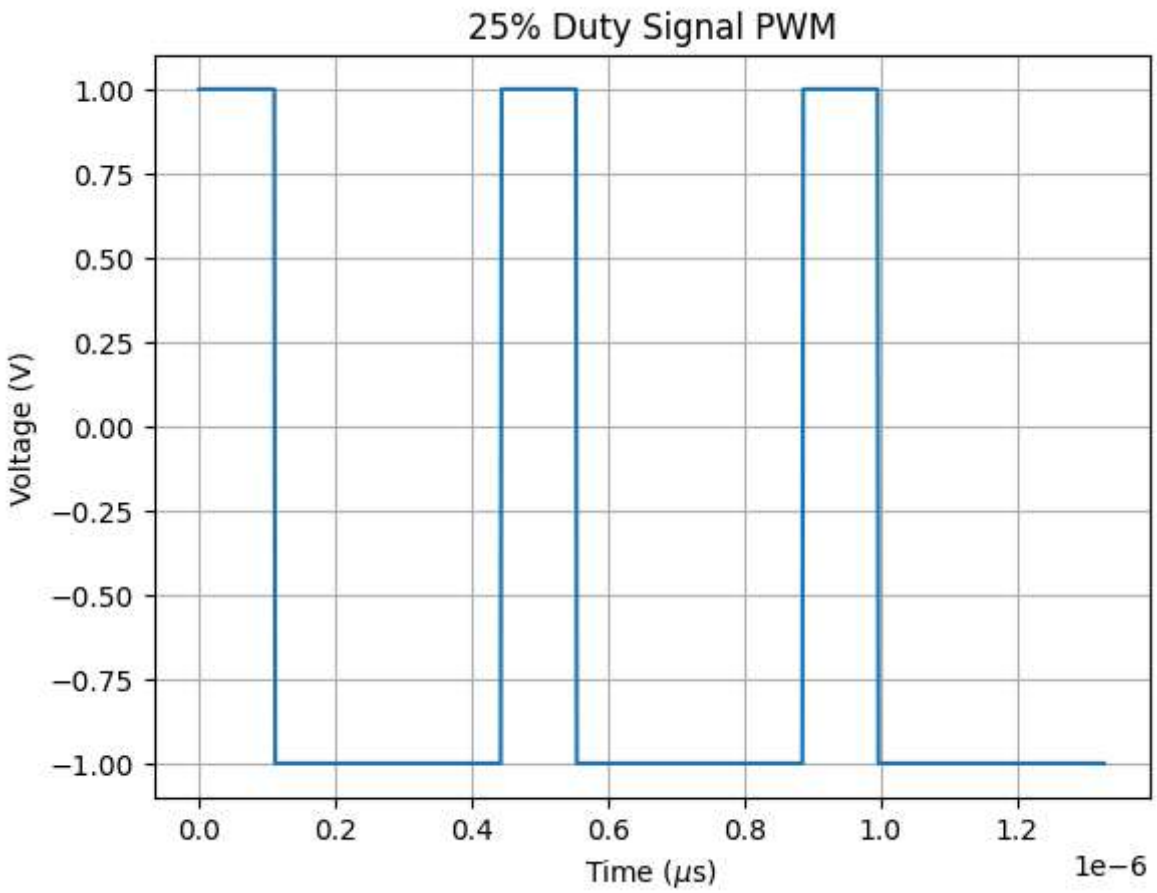
$$x\left(t\right) = \begin{cases} +V & \text{if } 0 \leq t \leq D \cdot T_0 \\ -V & \text{if } D \cdot T_0 < t < T_0 \end{cases} \tag{3}$$

For this project, $+V$ will be 1 volt and $-V$ will be -1 volt. The following plot shows a PWM signal with 25% duty cycle over 3 periods.

In [ ]:
```
# @title 25% Duty Cycle PWM
'''
Dillon Labonte
EE201 Signals and Systems
Professor Monk
Project 2
2/3/2024
https://colab.research.google.com/drive/1bsS0mF4i7BUzzEXjq7_A7QWgTEEXZQBe?usp=
sharing
This link should be accessible to anyone
'''

#import necessary modules
import numpy as np
import matplotlib.pyplot as plt
```

```
K = 1000 #number of points
tVals = np.linspace(0, 0.000001327433628, K) #creating t axis values
d = 0.25 #duty cycle (percentage)
f = 2260000 #fundamental frequency (student ID 2258458, rounded to 3
            # significant figures) in Hz
T = 1.0/f #fundamental period (s)

#creating PWM signal, switching between 1V and -1V
def PWMsignal(t, d):
  output = np.zeros_like(t)
  for i in range(len(t)):
    if (t[i] % T < (d * T)):
      output[i] = 1
    else:
      output[i] = -1
  return output

yVals = PWMsignal(tVals, d) #y values for PWM signal

plt.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
plt.grid(True)
plt.title("25% Duty Signal PWM")
plt.xlabel("Time ($\mu$s)")
plt.ylabel("Voltage (V)")
```

Out[ ]:
```
Text(0, 0.5, 'Voltage (V)')
```



To represent this signal using the complex Fourier Series, we must define two equations. The synthesis equation:

$$x(t) = \sum_{m=-\infty}^{\infty} c_m e^{jm\omega_0 t} \tag{4}$$

Which is used to sum up an infinite number of complex exponentials with coefficients c_m to represent the signal $x(t)$. The second equation is the analysis equation:

$$c_m = \frac{1}{T_0} \int_{T_0} x(t) e^{-jm\omega_0 t}\, dt \tag{5}$$

Which is used to solve for the coefficients c_m when integrating over the fundamental period T_0. This equation is primarily used for the derivation of a PWM signal Fourier Series representation.

To begin this representation, the analysis equation is used to find the coefficients c_m using the values that are known.

$$c_m = \frac{1}{T_0} \left(\int_0^{T_1} 1 \cdot e^{-jm\omega_0 t}\, dt + \int_{T_1}^{T_0} -1 \cdot e^{-jm\omega_0 t}\, dt \right) \tag{6}$$

Where T_1 is the period in seconds that the signal is high. Simplifying gives us:

$$c_m = \frac{1}{T_0} \left(\int_0^{T_1} 1 \cdot e^{-jm\omega_0 t}\, dt - \int_{T_1}^{T_0} 1 \cdot e^{-jm\omega_0 t}\, dt \right) \tag{7}$$

Integrating this results in:

$$c_m = \frac{1}{T_0} \left(\frac{1}{-jm\omega_0} e^{-jm\omega_0 t} \Big|_0^{T_1} - \frac{1}{-jm\omega_0} e^{-jm\omega_0 t} \Big|_{T_1}^{T_0} \right) \tag{8}$$

$$\implies c_m = \frac{1}{-jm\omega_0 T_0} \left(e^{-jm\omega_0 t} \Big|_0^{T_1} - e^{-jm\omega_0 t} \Big|_{T_1}^{T_0} \right) \tag{9}$$

It is important to note that the fundamental radian frequency ω_0 is defined as:

$$\omega_0 = \frac{2\pi}{T_0} \tag{10}$$

Plugging this into equation 9 results in:

$$c_m = \frac{1}{-jm\frac{2\pi}{T_0} T_0} \left(e^{-jm\frac{2\pi}{T_0} t} \Big|_0^{T_1} - e^{-jm\frac{2\pi}{T_0} t} \Big|_{T_1}^{T_0} \right) \tag{11}$$

$$\implies c_m = \frac{1}{-jm2\pi} \left(e^{-jm\frac{2\pi}{T_0} t} \Big|_0^{T_1} - e^{-jm\frac{2\pi}{T_0} t} \Big|_{T_1}^{T_0} \right) \tag{12}$$

Next, we substitute our bounds of integration into the equation for t.

$$c_m = \frac{1}{-jm2\pi} \left[\left(e^{-jm2\pi\frac{T_1}{T_0}} - e^{-jm\frac{2\pi}{T_0}(0)} \right) - \left(e^{-jm\frac{2\pi}{T_0} T_0} - e^{-jm2\pi\frac{T_1}{T_0}} \right) \right] \tag{13}$$

Plugging equation 1 into equation 13 and simplifying results in:

$$c_m = \frac{1}{-jm2\pi}\left[\left(e^{-jm2\pi D} - 1\right) - \left(e^{-jm2\pi} - e^{-jm2\pi D}\right)\right] \tag{14}$$

It is important to note that $e^{-jm2\pi}$ will always equal $\cos(0) = 1$ for integer values of m. Simplifying equation 14 gives us:

$$c_m = \frac{1}{-jm2\pi}\left[\left(e^{-jm2\pi D} - 1\right) - \left(1 - e^{-jm2\pi D}\right)\right] \tag{15}$$

$$\implies c_m = \frac{1}{-jm2\pi}\left[e^{-jm2\pi D} - 1 - 1 + e^{-jm2\pi D}\right] \tag{16}$$

$$\implies c_m = \frac{1}{-jm2\pi}\left[2e^{-jm2\pi D} - 2\right] \tag{17}$$

$$\implies c_m = \frac{2}{-jm2\pi}\left[e^{-jm2\pi D} - 1\right] \tag{18}$$

$$\implies c_m = \frac{1}{-jm\pi}\left[e^{-jm2\pi D} - 1\right] \tag{19}$$

By factoring out the average frequency of the terms inside the parentheses, this equation can be simplified further.

$$c_m = \frac{e^{\frac{-jm2\pi D}{2}}}{-jm\pi}\left[e^{\frac{-jm2\pi D}{2}} - e^{\frac{jm2\pi D}{2}}\right] \tag{20}$$

$$\implies c_m = \frac{e^{-jm\pi D}}{-jm\pi}\left[e^{-jm\pi D} - e^{jm\pi D}\right] \tag{21}$$

$$\implies c_m = \frac{e^{-jm\pi D}}{m\pi}\left[\frac{e^{jm\pi D} - e^{-jm\pi D}}{j}\right] \tag{22}$$

$$\implies c_m = \frac{2\sin(m\pi D)}{m\pi D} \cdot De^{-jm\pi D} \tag{23}$$

$$\implies \boxed{c_m = 2\text{sinc}(m\pi D) \cdot De^{-jm\pi D}} \tag{24}$$

This is now our derived and simplified equation for the complex coefficients c_m for any integer value of m. It is important to note that c_0 would result in the indeterminate form $\frac{0}{0}$ and must be dealt with by plugging 0 into equation 5 for m.

$$c_0 = \frac{1}{T_0}\left(\int_0^{T_1} 1 \cdot e^{-j(0)\omega_0 t}\,dt - \int_{T_1}^{T_0} 1 \cdot e^{-j(0)\omega_0 t}\,dt\right) \tag{25}$$

$$\implies c_0 = \frac{1}{T_0}\left(\int_0^{T_1} 1\,dt - \int_{T_1}^{T_0} 1\,dt\right) \tag{26}$$

$$\implies c_0 = \frac{1}{T_0}\left(t\big|_0^{T_1} - t\big|_{T_1}^{T_0}\right) \tag{27}$$

$$\implies c_0 = \frac{1}{T_0}((T_1 - 0) - (T_0 - T_1)) \tag{28}$$

$$\implies c_0 = \frac{1}{T_0}(T_1 - (T_0 - T_1)) \tag{29}$$

$$\implies c_0 = \frac{1}{T_0}(T_1 - T_0 + T_1) \tag{30}$$

$$\implies c_0 = \frac{1}{T_0}(2T_1 - T_0) \tag{31}$$

$$\implies c_0 = \frac{2T_1}{T_0} - \frac{T_0}{T_0} \tag{32}$$

Plugging equation 1 into equation 12 and simplifying results in:

$$\boxed{c_0 = 2D - 1} \tag{33}$$

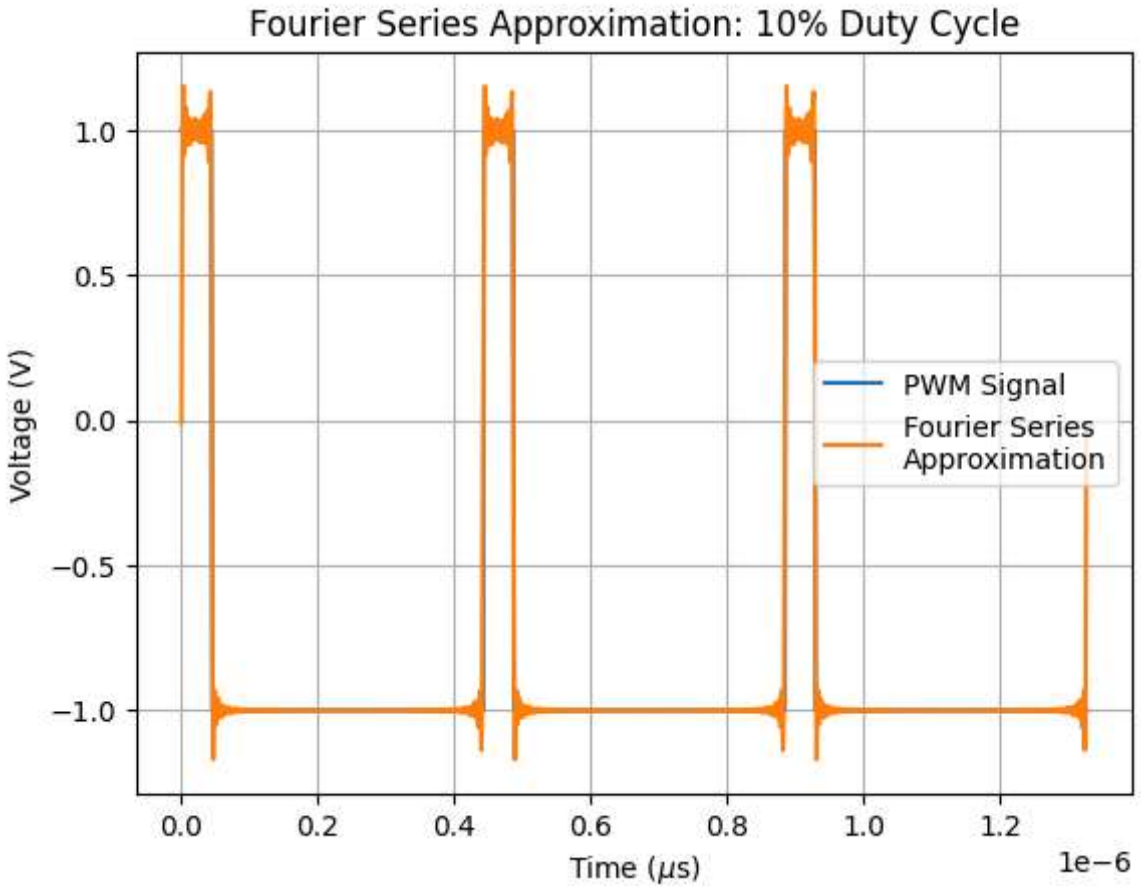
The following plots show the Fourier Series approximation (limited number of terms used), as well as the harmonic magnitudes and phases for duty cycles of: 10%, 25%, 50%, 75%, 90%, and 100%.

In [ ]:
```
# @title FS 10% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 10%

N = 100 #number of terms in FS
d = 0.1 #duty cycle

yVals = PWMsignal(tVals, d) #y values for PWM signal

FScoeff10 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal10 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics10 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1, N):
  FScoeff10[n] = (1 / (-1j * n * np.pi)) * \
  (np.exp(-1j * n * 2 * np.pi * d) - 1)
  FSsignal10 += FScoeff10[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff10[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))

#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff10[0] = ((2 * d) - 1)
FSsignal10 += FScoeff10[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude10 = np.arange(0, N)
for n in range (0, N):
  coefficientMagnitude10[n] = 20*np.log10(np.abs(FScoeff10[n]))

#determining coefficient phases for plotting
coefficientPhase10 = np.arange(0, N)
for n in range (0, N):
  coefficientPhase10[n] = (180/np.pi)*np.angle(FScoeff10[n])
```

```
#FS versus PWM plot
fig, (plot1) = plt.subplots(1)
plot1.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot1.plot(tVals, FSsignal10, label = "Fourier Series\nApproximation")
plot1.set_title('Fourier Series Approximation: 10% Duty Cycle')
plot1.grid()
plot1.set_ylabel('Voltage (V)')
plot1.set_xlabel('Time ($\mu$s)')
plot1.legend(loc = "center right")
```

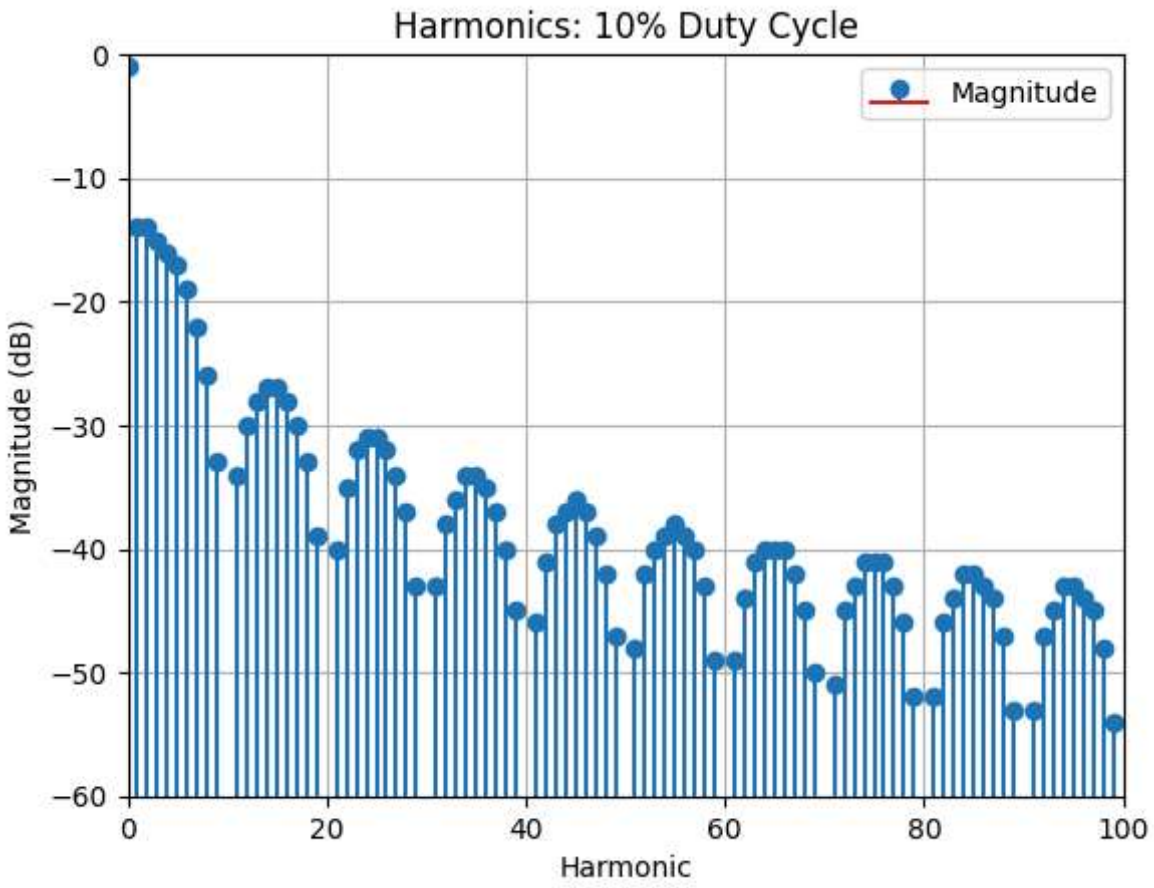
```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335: ComplexWar
ning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7eb16013f9a0>
```



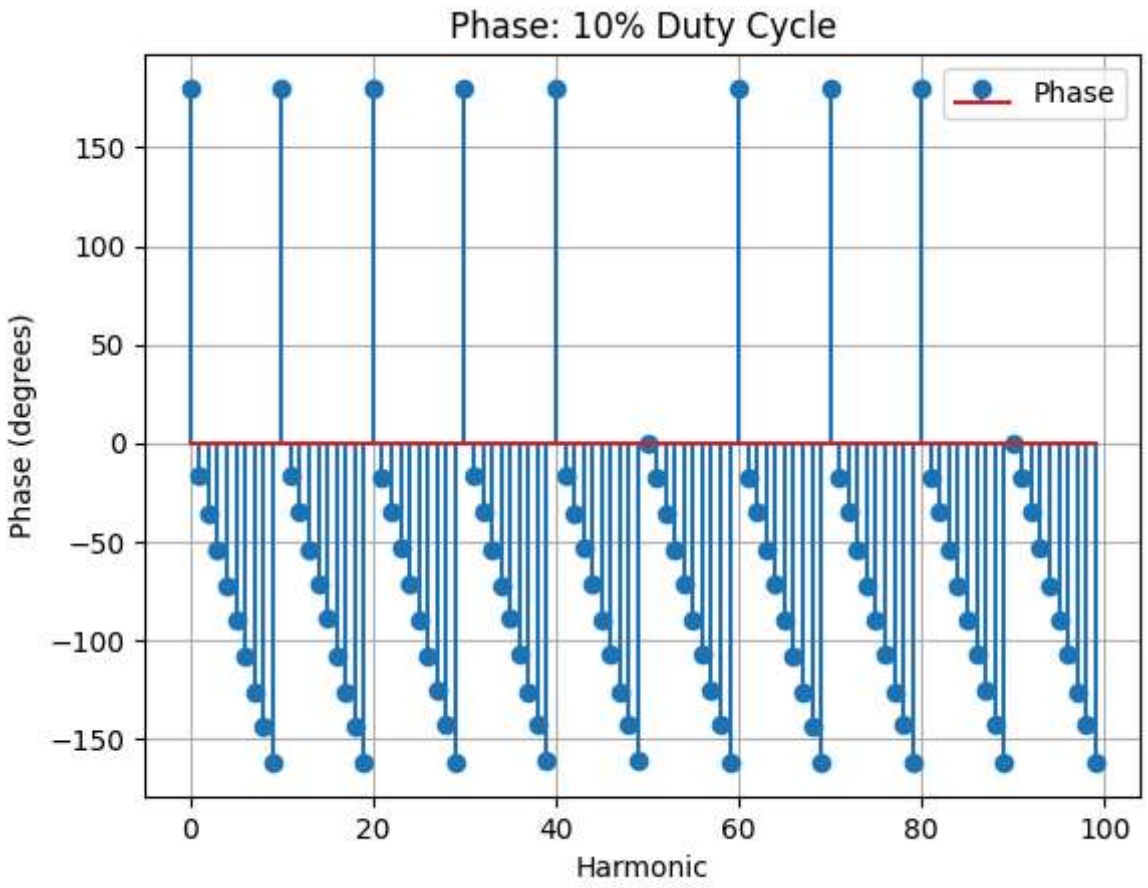
In [ ]:
```
# @title Harmonics 10% Duty Cycle
#harmonics plot
fig, (plot2) = plt.subplots(1)
plot2.stem(harmonics10, coefficientMagnitude10, label='Magnitude', \
           bottom = -100)
plot2.set_title('Harmonics: 10% Duty Cycle')
plot2.grid(True)
plot2.set_xlabel('Harmonic')
plot2.set_ylabel('Magnitude (dB)')
plot2.axis([0, 100, -60, 0])
plot2.legend(loc = 'best')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a51583b5420>
```



In [ ]:
```
# @title Phase 10% Duty Cycle
#phase plot
fig, (plot3) = plt.subplots(1)
plot3.stem(harmonics10, coefficientPhase10, label='Phase')
plot3.set_title('Phase: 10% Duty Cycle')
plot3.grid(True)
plot3.set_xlabel('Harmonic')
plot3.set_ylabel('Phase (degrees)')
plot3.legend(loc = 'best')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a5145b89630>
```



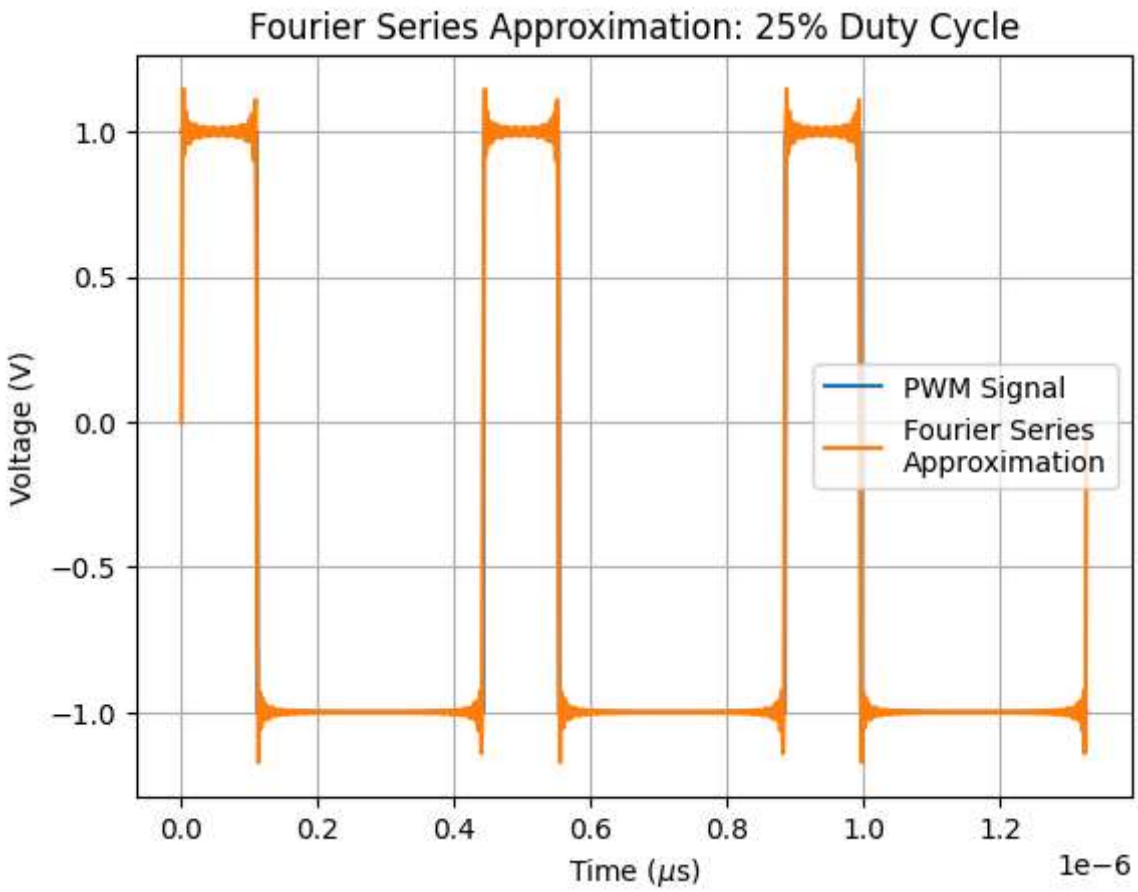
In [ ]:

```
# @title FS 25% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 25%

N = 100 #number of terms in FS
d = 0.25 #duty cycle

yVals = PWMsignal(tVals, d) #y values for PWM signal

FScoeff25 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal25 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics25 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1,N):
  FScoeff25[n] = (1 / (-1j * n * 2 * np.pi)) * \
   (2 * np.exp(-1j * n * 2 * np.pi * d) - 2)
  FSsignal25 += FScoeff25[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff25[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))

#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff25[0] = (2 * d) - 1
FSsignal25 += FScoeff25[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude25 = np.arange(0, N)
for n in range (0, N):
  coefficientMagnitude25[n] = 20*np.log10(np.abs(FScoeff25[n]))

#determining coefficient phases for plotting
coefficientPhase25 = np.arange(0, N)
```

```
for n in range (0, N):
  coefficientPhase25[n] = 180/np.pi*np.angle(FScoeff25[n])

#FS versus PWM plot
fig, (plot4) = plt.subplots(1)
plot4.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot4.plot(tVals, FSsignal25, label = "Fourier Series\nApproximation")
plot4.set_title('Fourier Series Approximation: 25% Duty Cycle')
plot4.grid()
plot4.set_ylabel('Voltage (V)')
plot4.set_xlabel('Time ($\mu$s)')
plot4.legend(loc = "center right")
```

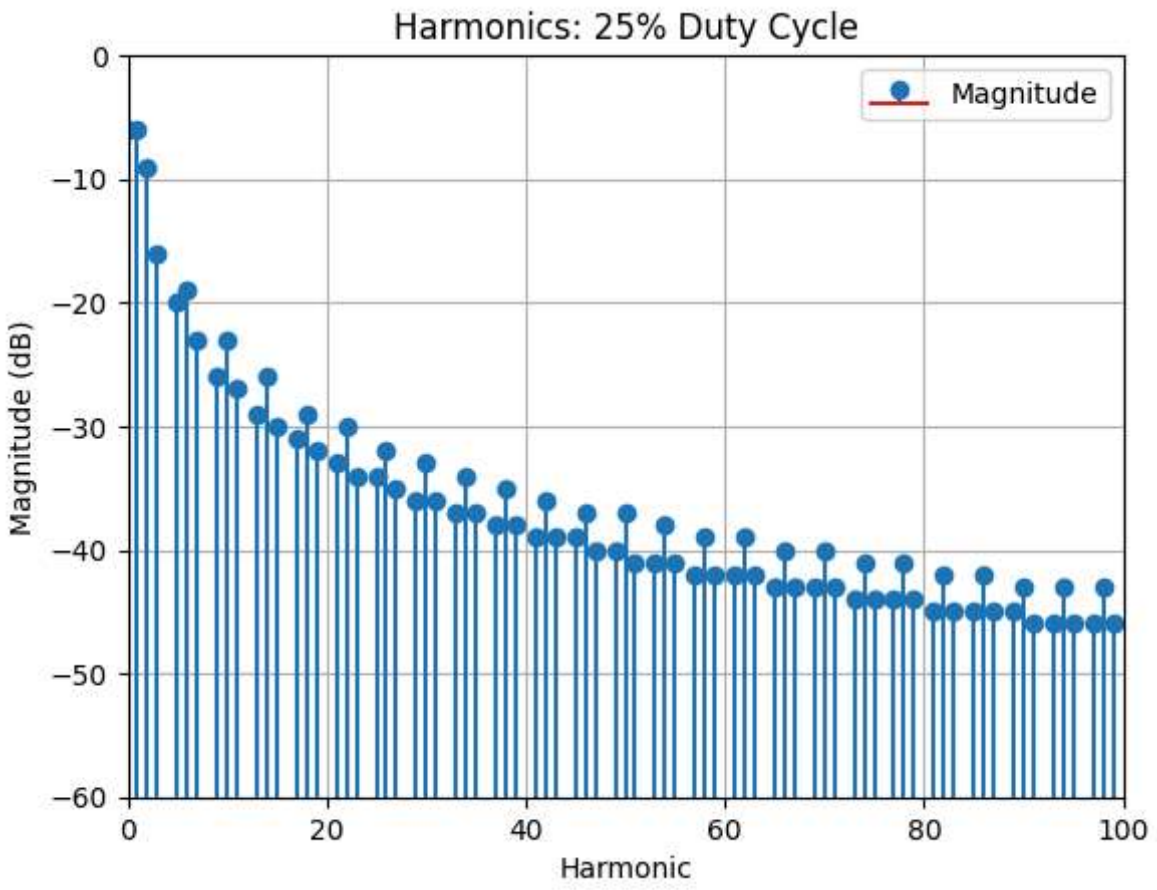
```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335: ComplexWar
ning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a51476b41c0>
```



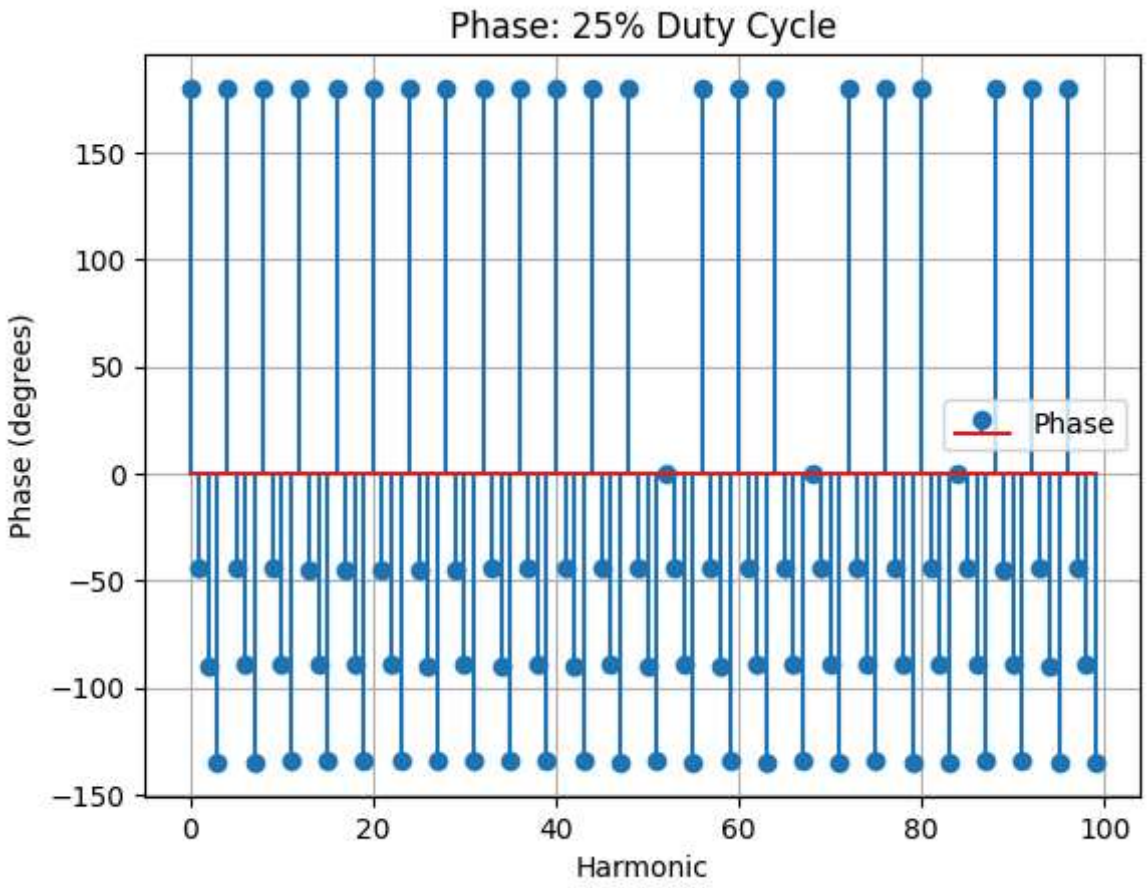
In [ ]:
```
# @title Harmonics 25% Duty Cycle
#harmonics plot
fig, (plot5) = plt.subplots(1)
plot5.stem(harmonics25, coefficientMagnitude25, label='Magnitude', \
           bottom = -100)
plot5.set_title('Harmonics: 25% Duty Cycle')
plot5.grid(True)
plot5.set_xlabel('Harmonic')
plot5.set_ylabel('Magnitude (dB)')
plot5.axis([0, 100, -60, 0])
plot5.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a5147513850>



In [ ]:
```
# @title Phase 25% Duty Cycle
#phase plot
fig, (plot6) = plt.subplots(1)
plot6.stem(harmonics25, coefficientPhase25, label='Phase')
plot6.set_title('Phase: 25% Duty Cycle')
plot6.grid(True)
plot6.set_xlabel('Harmonic')
plot6.set_ylabel('Phase (degrees)')
plot6.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a514759e440>

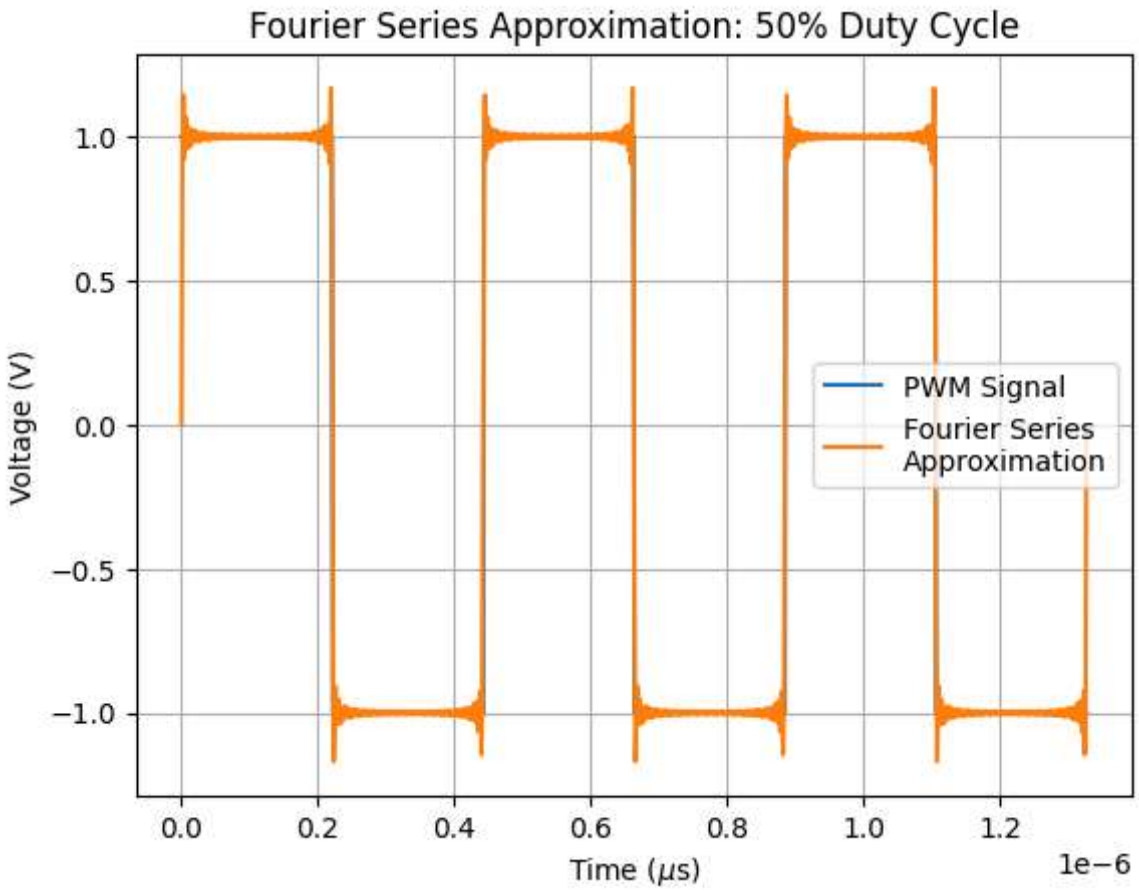


In [ ]:

```
# @title FS 50% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 50%

N = 100 #number of terms in FS
d = 0.5 #duty cycle

yVals = PWMsignal(tVals, d) #y values for PWM signal

FScoeff50 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal50 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics50 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1,N):
  FScoeff50[n] = (1 / (-1j * n * 2 * np.pi)) * \
   (2 * np.exp(-1j * n * 2 * np.pi * d) - 2)
  FSsignal50 += FScoeff50[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff50[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))

#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff50[0] = (2 * d) - 1
FSsignal50 += FScoeff50[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude50 = np.arange(0, N)
for n in range (1, N):
  coefficientMagnitude50[n] = 20*np.log10(np.abs(FScoeff50[n]))

#determining coefficient phases for plotting
coefficientPhase50 = np.arange(0, N)
```

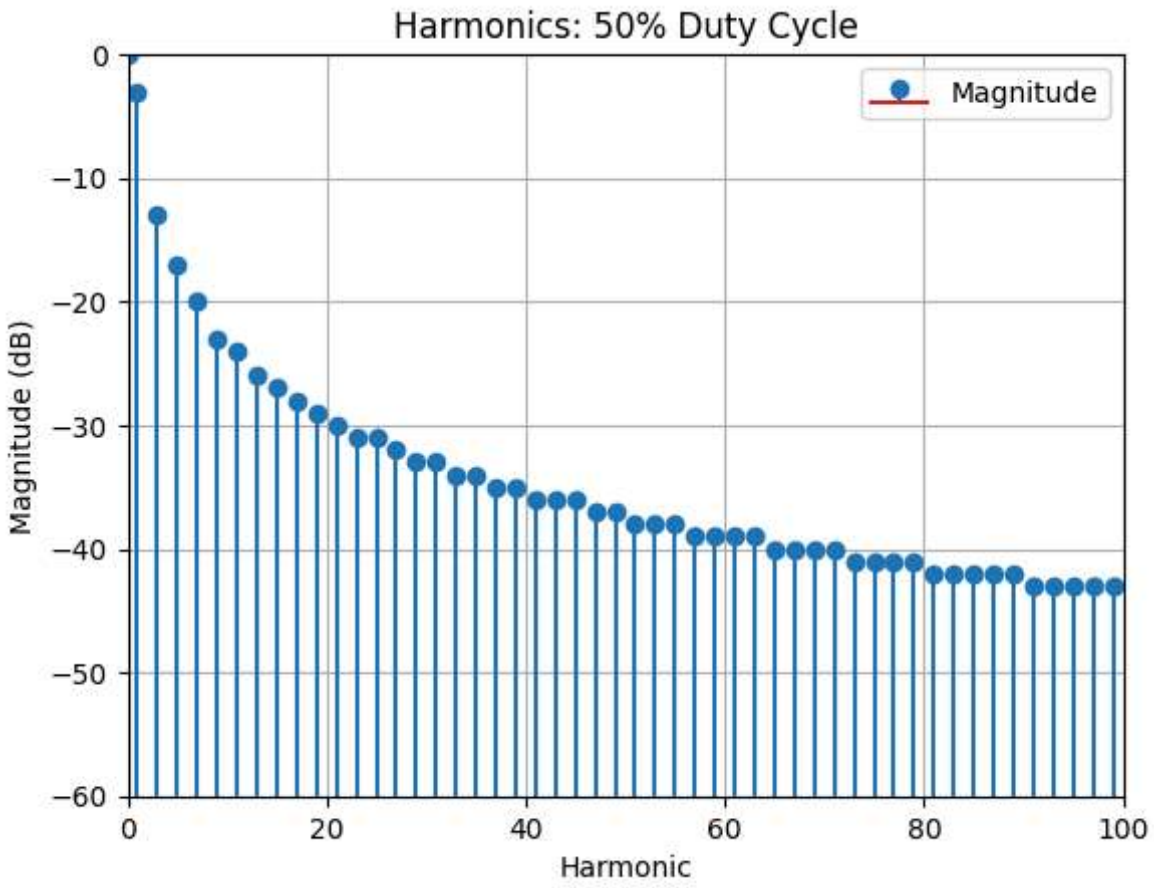
```
for n in range (0, N):
  coefficientPhase50[n] = 180/np.pi*np.angle(FScoeff50[n])

#FS versus PWM plot
fig, (plot7) = plt.subplots(1)
plot7.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot7.plot(tVals, FSsignal50, label = "Fourier Series\nApproximation")
plot7.set_title('Fourier Series Approximation: 50% Duty Cycle')
plot7.grid()
plot7.set_ylabel('Voltage (V)')
plot7.set_xlabel('Time ($\mu$s)')
plot7.legend(loc = "center right")
```

Out[ ]: <matplotlib.legend.Legend at 0x7a5147385810>



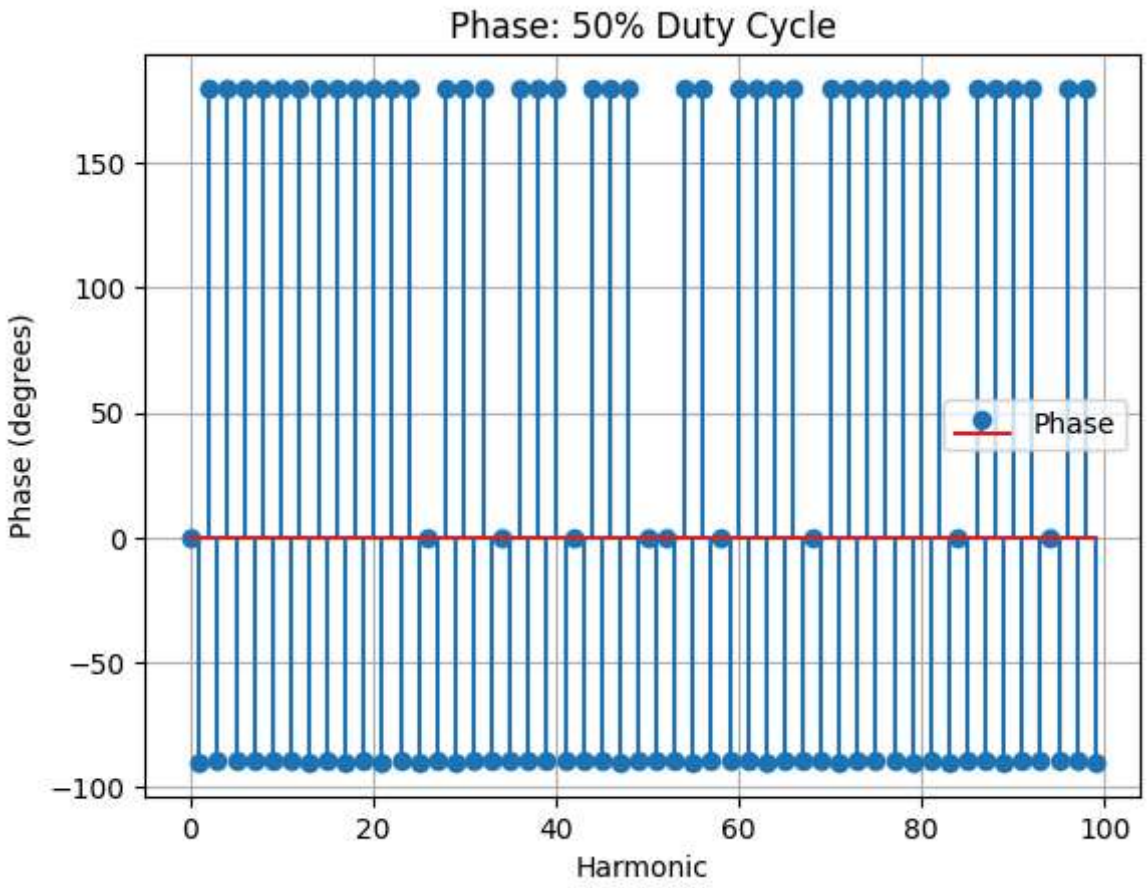
In [ ]:
```
# @title Harmonics 50% Duty Cycle
#harmonics plot
fig, (plot8) = plt.subplots(1)
plot8.stem(harmonics50, coefficientMagnitude50, label='Magnitude', \
           bottom = -100)
plot8.set_title('Harmonics: 50% Duty Cycle')
plot8.grid(True)
plot8.set_xlabel('Harmonic')
plot8.set_ylabel('Magnitude (dB)')
plot8.axis([0, 100, -60, 0])
plot8.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a51473dbc10>



In [ ]:
```
# @title Phase 50% Duty Cycle
#phase plot
fig, (plot9) = plt.subplots(1)
plot9.stem(harmonics50, coefficientPhase50, label='Phase')
plot9.set_title('Phase: 50% Duty Cycle')
plot9.grid(True)
plot9.set_xlabel('Harmonic')
plot9.set_ylabel('Phase (degrees)')
plot9.legend(loc = 'best')
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a5147272e90>
```



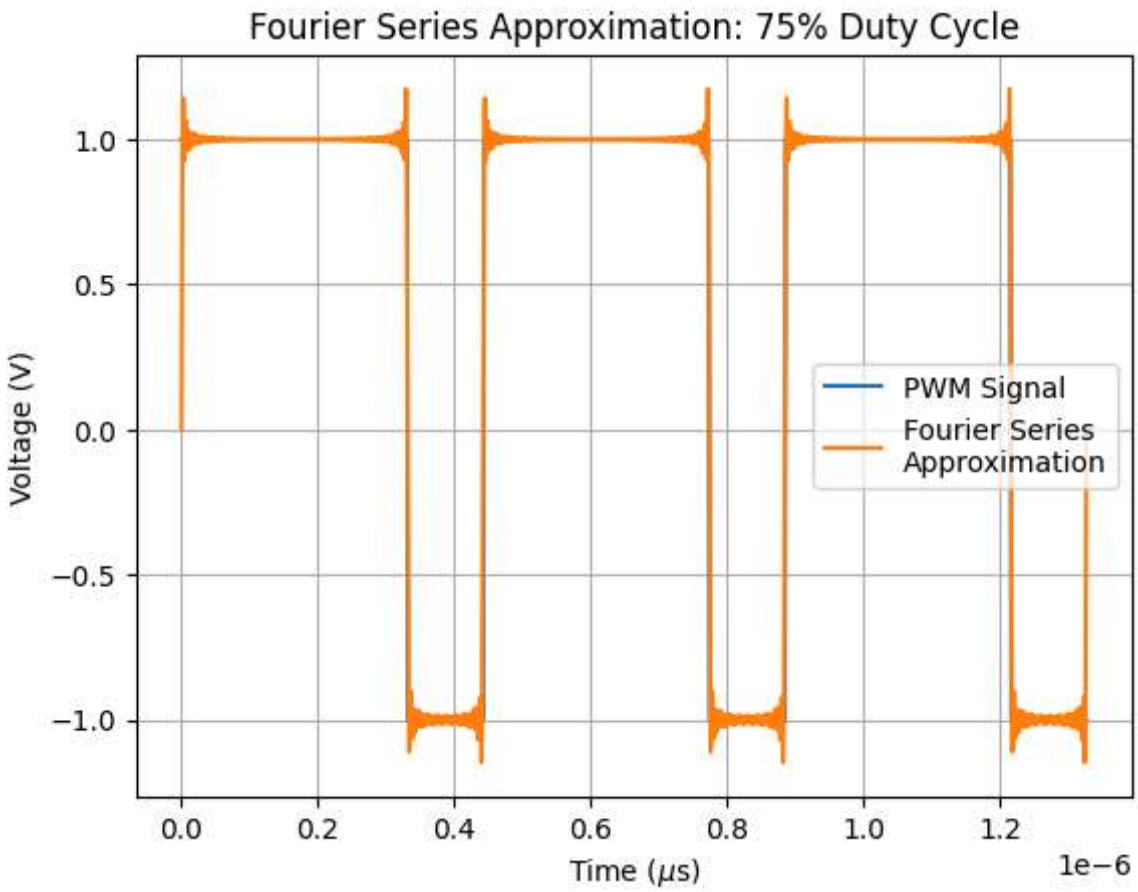
In [ ]:

```
# @title FS 75% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 75%

N = 100 #number of terms in FS
d = 0.75 #duty cycle

yVals = PWMsignal(tVals, d) #y values for PWM signal

FScoeff75 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal75 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics75 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1,N):
  FScoeff75[n] = (1 / (-1j * n * 2 * np.pi)) * \
   (2 * np.exp(-1j * n * 2 * np.pi * d) - 2)
  FSsignal75 += FScoeff75[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff75[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))

#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff75[0] = (2 * d) - 1
FSsignal75 += FScoeff75[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude75 = np.arange(0, N)
for n in range (0, N):
  coefficientMagnitude75[n] = 20*np.log10(np.abs(FScoeff75[n]))

#determining coefficient phases for plotting
coefficientPhase75 = np.arange(0, N)
```

```
for n in range (0, N):
  coefficientPhase75[n] = 180/np.pi*np.angle(FScoeff75[n])

#FS versus PWM plot
fig, (plot10) = plt.subplots(1)
plot10.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot10.plot(tVals, FSsignal75, label = "Fourier Series\nApproximation")
plot10.set_title('Fourier Series Approximation: 75% Duty Cycle')
plot10.grid()
plot10.set_ylabel('Voltage (V)')
plot10.set_xlabel('Time ($\mu$s)')
plot10.legend(loc = "center right")
```

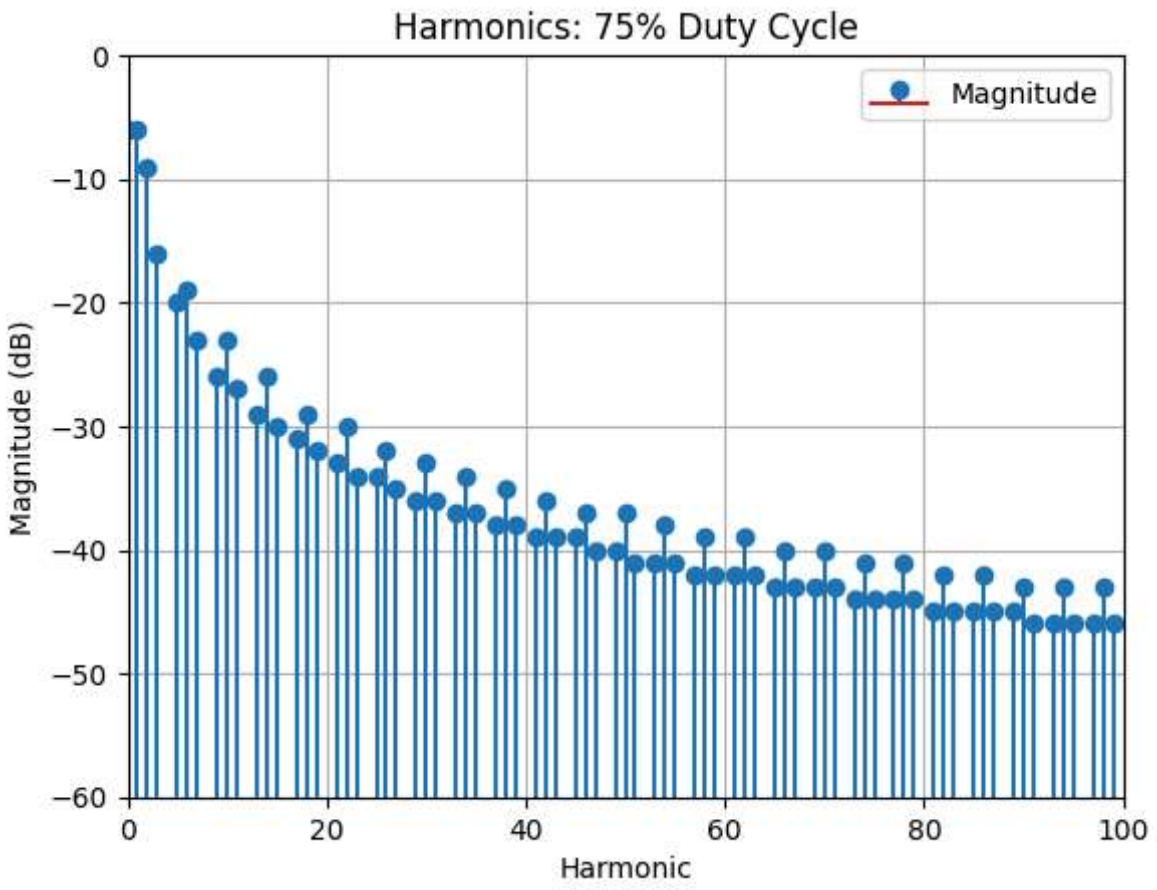
```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335: ComplexWar
ning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a514714ffd0>
```



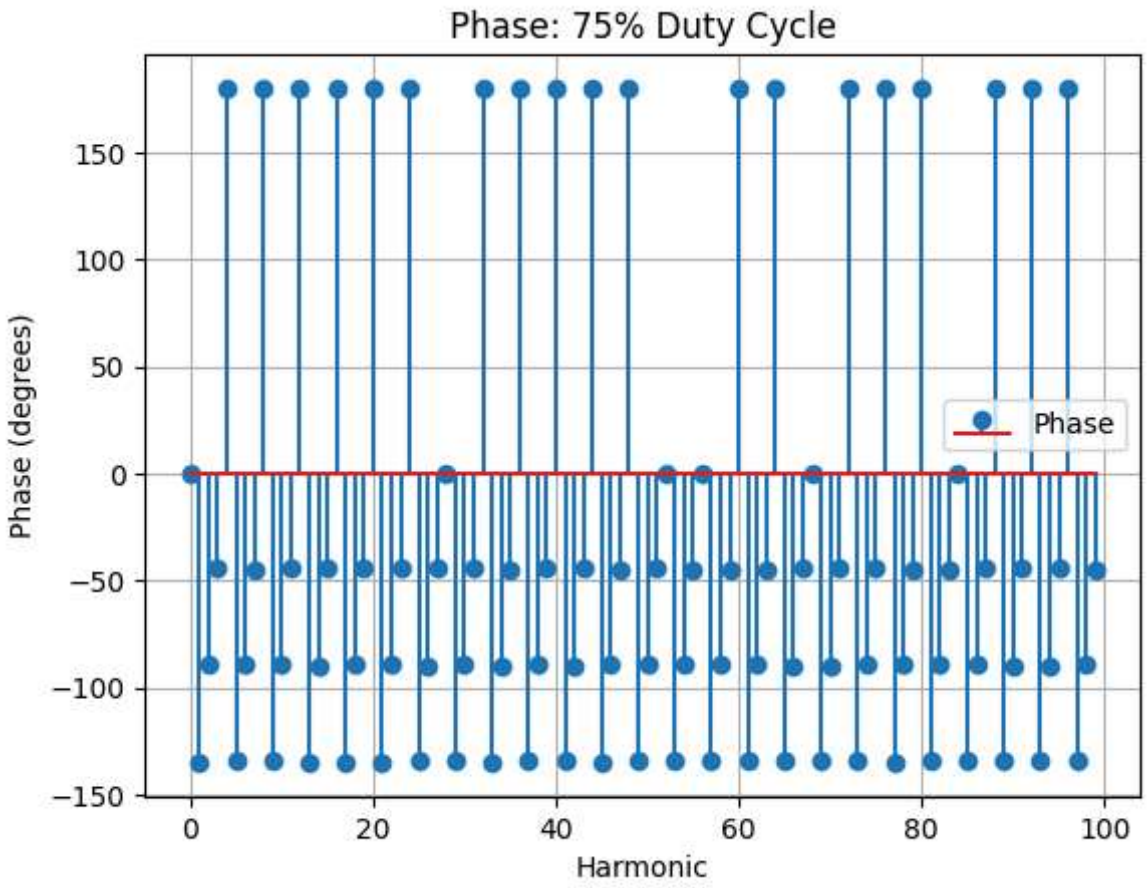
In [ ]:
```
# @title Harmonics 75% Duty Cycle
#harmonics plot
fig, (plot11) = plt.subplots(1)
plot11.stem(harmonics75, coefficientMagnitude75, label='Magnitude', \
            bottom = -100)
plot11.set_title('Harmonics: 75% Duty Cycle')
plot11.grid(True)
plot11.set_xlabel('Harmonic')
plot11.set_ylabel('Magnitude (dB)')
plot11.axis([0, 100, -60, 0])
plot11.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a51471bb3d0>



In [ ]:
```
# @title Phase 75% Duty Cycle
#phase plot
fig, (plot12) = plt.subplots(1)
plot12.stem(harmonics75, coefficientPhase75, label='Phase')
plot12.set_title('Phase: 75% Duty Cycle')
plot12.grid(True)
plot12.set_xlabel('Harmonic')
plot12.set_ylabel('Phase (degrees)')
plot12.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a514714ffa0>



In [ ]:

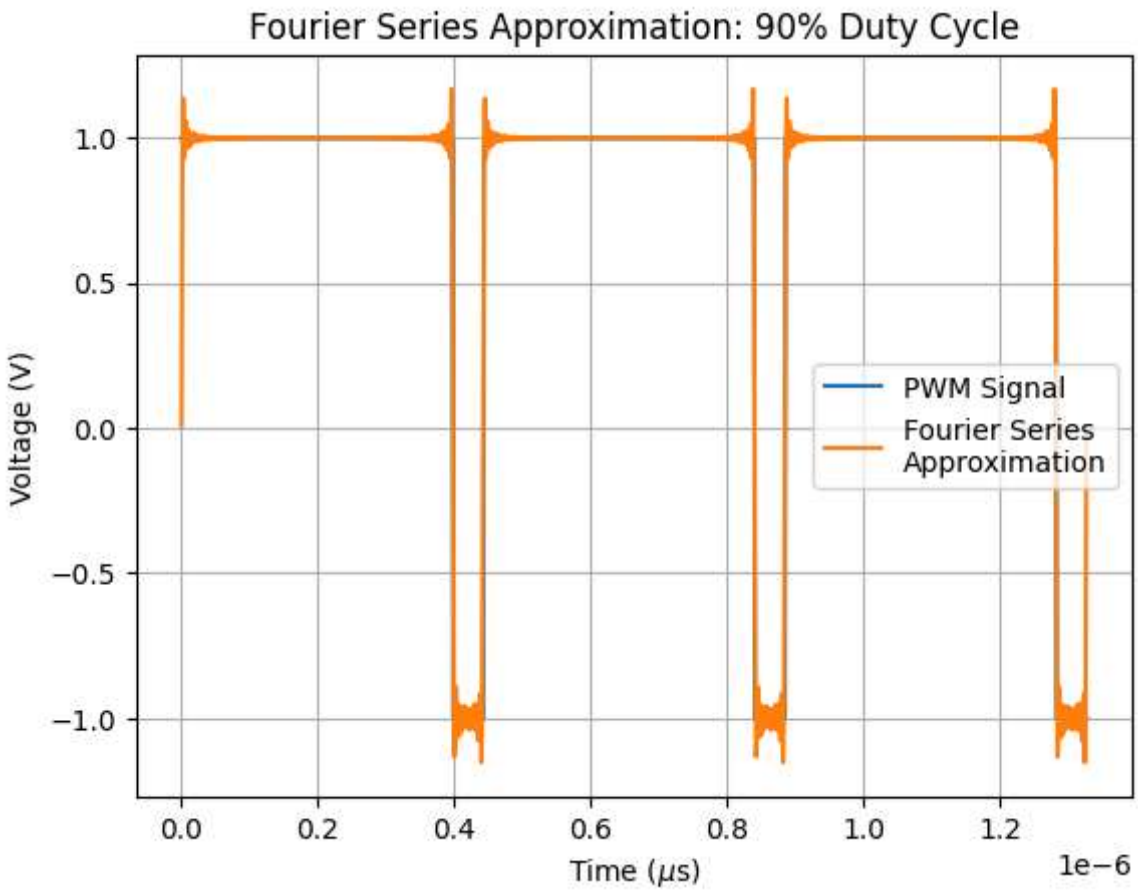
```
# @title FS 90% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 90%

N = 100 #number of terms in FS
d = 0.90 #duty cycle

yVals = PWMsignal(tVals, d) #y values for PWM signal

FScoeff90 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal90 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics90 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1,N):
  FScoeff90[n] = (1 / (-1j * n * 2 * np.pi)) * \
   (2 * np.exp(-1j * n * 2 * np.pi * d) - 2)
  FSsignal90 += FScoeff90[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff90[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))

#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff90[0] = (2 * d) - 1
FSsignal90 += FScoeff90[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude90 = np.arange(0, N)
for n in range (0, N):
  coefficientMagnitude90[n] = 20*np.log10(np.abs(FScoeff90[n]))

#determining coefficient phases for plotting
coefficientPhase90 = np.arange(0, N)
```

```
for n in range (0, N):
  coefficientPhase90[n] = 180/np.pi*np.angle(FScoeff90[n])

#FS versus PWM plot
fig, (plot13) = plt.subplots(1)
plot13.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot13.plot(tVals, FSsignal90, label = "Fourier Series\nApproximation")
plot13.set_title('Fourier Series Approximation: 90% Duty Cycle')
plot13.grid()
plot13.set_ylabel('Voltage (V)')
plot13.set_xlabel('Time ($\mu$s)')
plot13.legend(loc = "center right")
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335: ComplexWar
ning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Out[ ]:

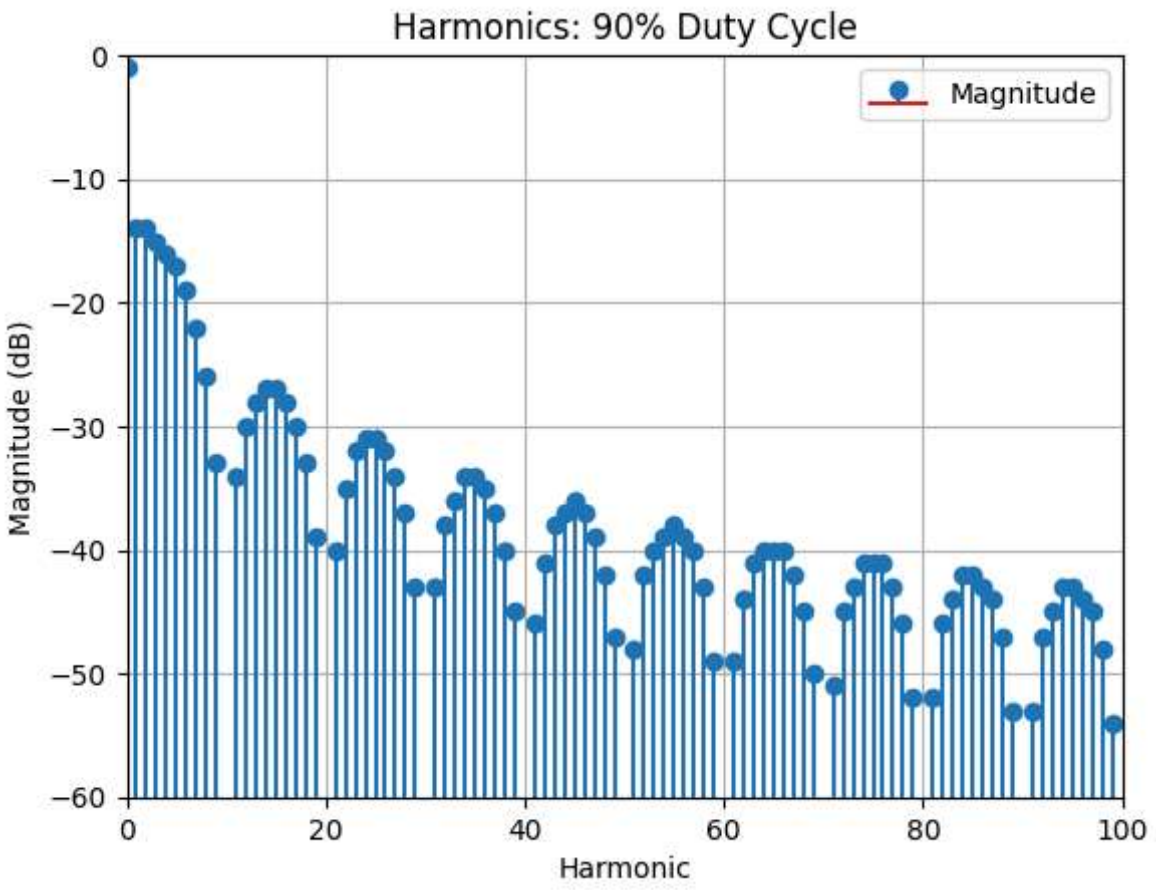
```
<matplotlib.legend.Legend at 0x7a5146f4bfd0>
```



In [ ]:

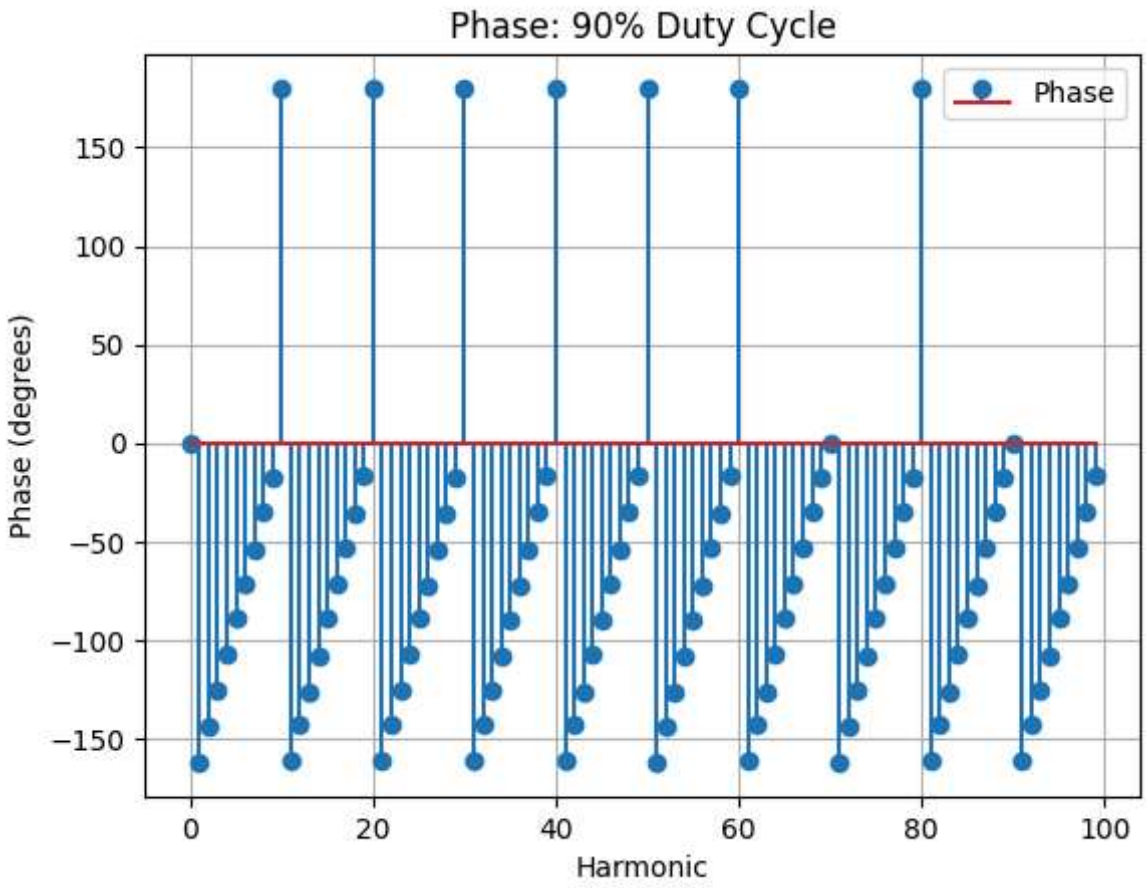
```
# @title Harmonics 90% Duty Cycle
#harmonics plot
fig, (plot14) = plt.subplots(1)
plot14.stem(harmonics90, coefficientMagnitude90, label='Magnitude', \
            bottom = -100)
plot14.set_title('Harmonics: 90% Duty Cycle')
plot14.grid(True)
plot14.set_xlabel('Harmonic')
plot14.set_ylabel('Magnitude (dB)')
plot14.axis([0, 100, -60, 0])
plot14.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a5146fdcc40>



In [ ]:
```
# @title Phase 90% Duty Cycle
#phase plot
fig, (plot15) = plt.subplots(1)
plot15.stem(harmonics90, coefficientPhase90, label='Phase')
plot15.set_title('Phase: 90% Duty Cycle')
plot15.grid(True)
plot15.set_xlabel('Harmonic')
plot15.set_ylabel('Phase (degrees)')
plot15.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a5146e461d0>



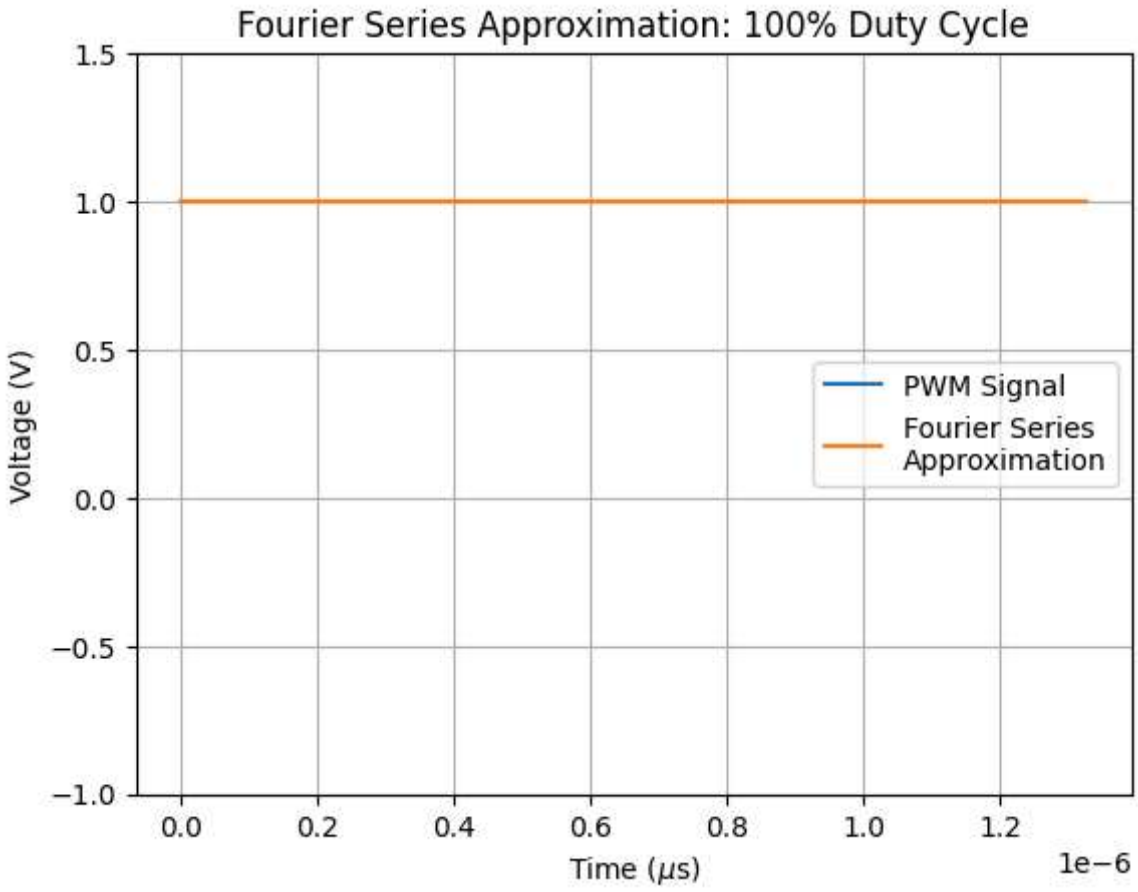
In [ ]:

```
# @title FS 100% Duty Cycle
#PLOTS FOR DUTY CYCLE OF 100%

N = 100 #number of terms in FS
d = 1 #duty cycle

#yVals = PWMsignal(tVals, d) #y values for PWM signal
yVals = np.ones(1000)

FScoeff100 = np.zeros(N, dtype = 'complex_') #FS coefficients array
FSsignal100 = np.zeros(K, dtype = 'complex_') #FS output array
harmonics100 = np.arange(0,N) #harmonic indices

#synthesis of FS using derived synthesis and analysis equations
for n in np.arange(1,N):
  FScoeff100[n] = (1 / (-1j * n * 2 * np.pi)) * \
   (2 * np.exp(-1j * n * 2 * np.pi * d) - 2)
  FSsignal100 += FScoeff100[n] * (np.exp(1j * n * 2 * np.pi * f * tVals)) + \
  np.conj(FScoeff100[n]) * (np.exp(-1j * n * 2 * np.pi * f * tVals))
#adding the 0th coefficient
#(cannot do this in the for loop above, would result in divide by 0 error)
FScoeff100[0] = (2 * d) - 1
FSsignal100 += FScoeff100[0]

#determining coefficient magnitudes for plotting
coefficientMagnitude100 = np.arange(0, N)
for n in range (0, N):
  coefficientMagnitude100[n] = 20*np.log10(np.abs(FScoeff100[n]))

#determining coefficient phases for plotting
coefficientPhase100 = np.arange(0, N)
```

```
for n in range (0, N):
  coefficientPhase100[n] = 180/np.pi*np.angle(FScoeff100[n])

#FS versus PWM plot
fig, (plot16) = plt.subplots(1)
plot16.plot(tVals, yVals, label = "PWM Signal") #plotting PWM signal
#plotting FS approximation
plot16.plot(tVals, FSsignal100, label = "Fourier Series\nApproximation")
plot16.set_title('Fourier Series Approximation: 100% Duty Cycle')
plot16.grid()
plot16.set_ylabel('Voltage (V)')
plot16.set_xlabel('Time ($\mu$s)')
plot16.set_ylim(-1, 1.5)
plot16.legend(loc = "center right")
```

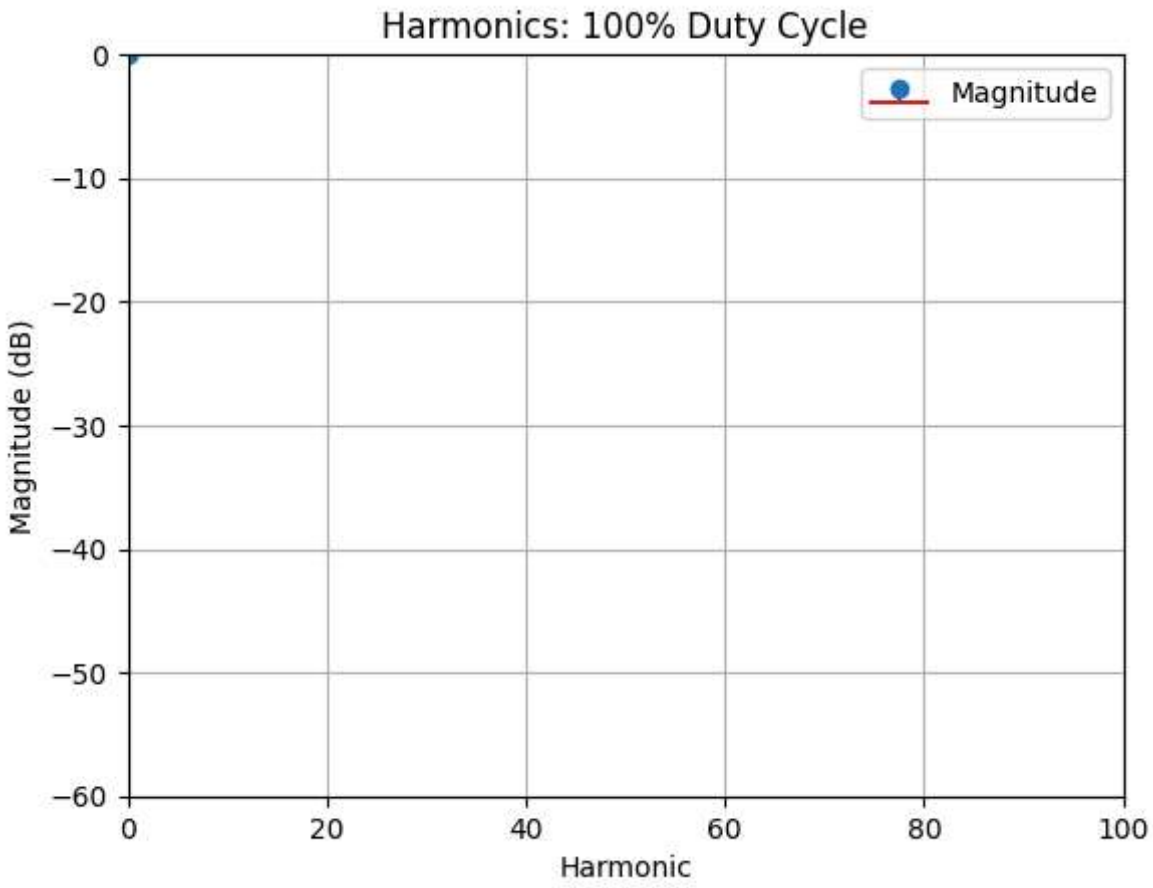
```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335: ComplexWar
ning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7a517824f8e0>
```



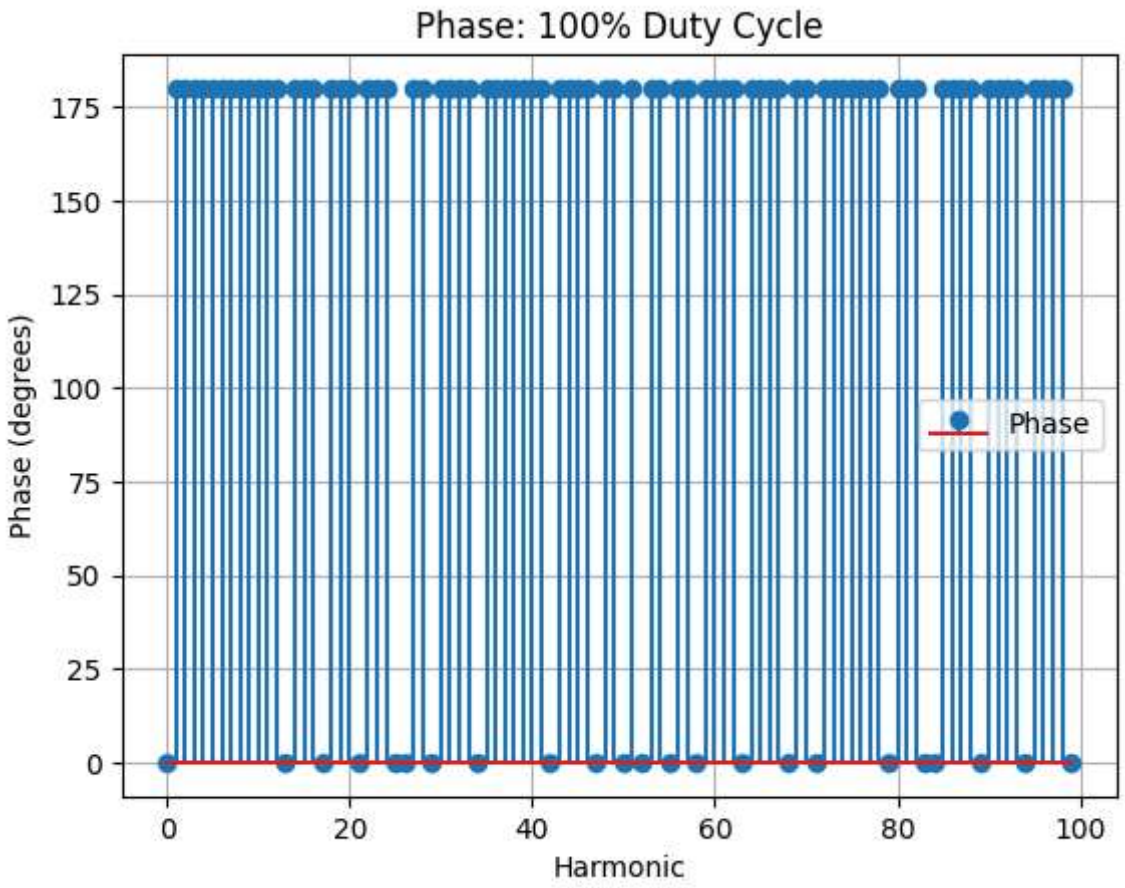
In [ ]:
```
# @title Harmonics 100% Duty Cycle
#harmonics plot
fig, (plot17) = plt.subplots(1)
plot17.stem(harmonics100, coefficientMagnitude100, label='Magnitude', \
            bottom = -100)
plot17.set_title('Harmonics: 100% Duty Cycle')
plot17.grid(True)
plot17.set_xlabel('Harmonic')
plot17.set_ylabel('Magnitude (dB)')
plot17.axis([0, 100, -60, 0])
plot17.legend(loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7a5147549840>



In [ ]:
```
# @title Phase 100% Duty Cycle
#phase plot
fig, (plot18) = plt.subplots(1)
plot18.stem(harmonics100, coefficientPhase100, label='Phase')
plot18.set_title('Phase: 100% Duty Cycle')
plot18.grid(True)
plot18.set_xlabel('Harmonic')
plot18.set_ylabel('Phase (degrees)')
plot18.legend(loc = 'best')
```

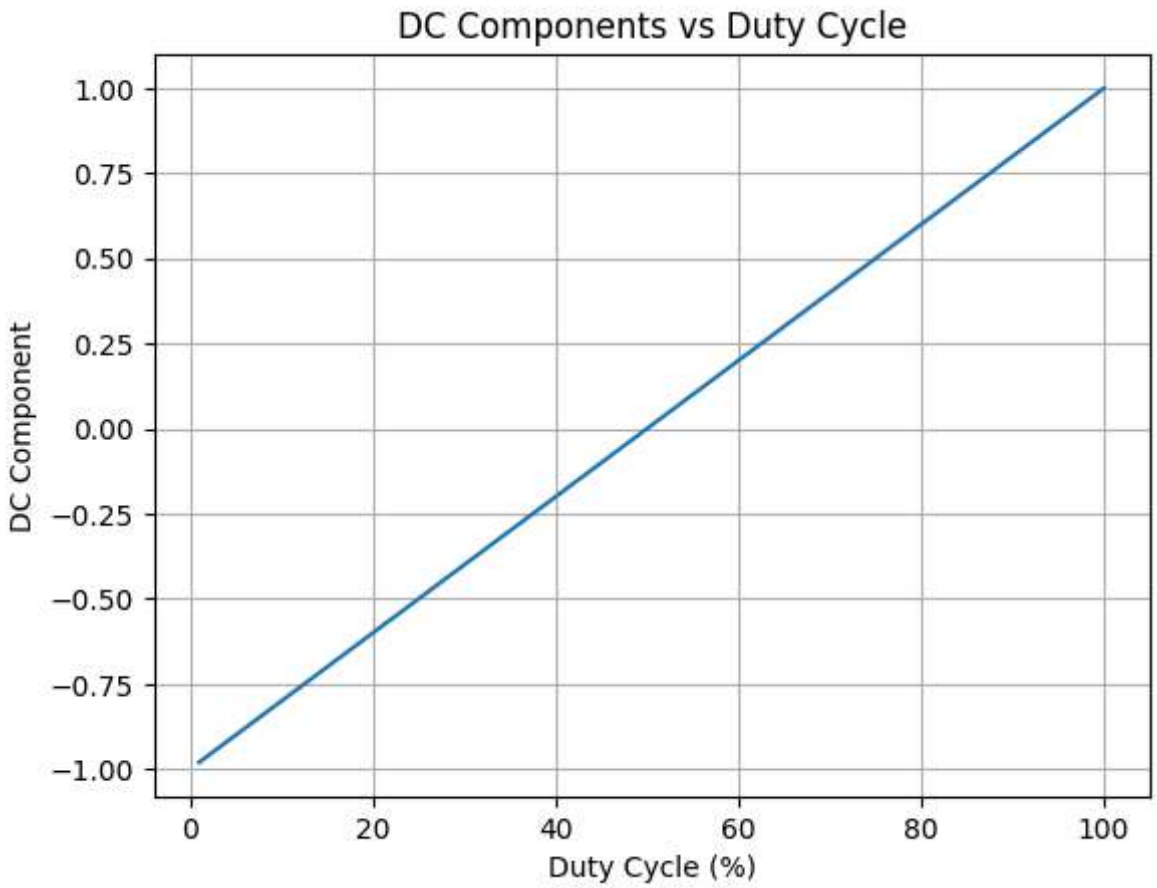
Out[ ]: <matplotlib.legend.Legend at 0x7a51475131c0>



Notice that the harmonic magnitude plot for the 100% duty cycle is nearly blank, however one harmonic survives which is the fundamental. This is a constant and represents the average value of the signal ($1V$).

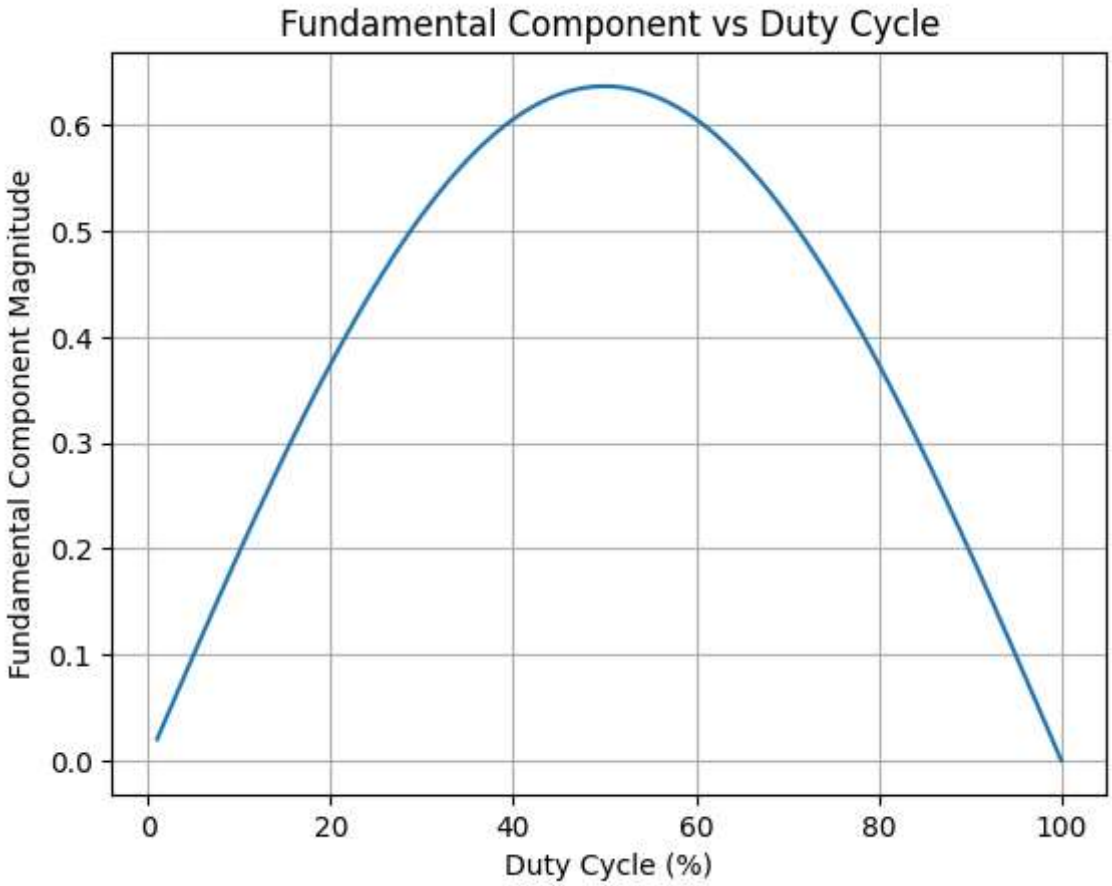
The following plot shows the DC component of the Fourier Series as duty cycle increases. The DC component is constant and represents the average value of the function.

In [ ]:
```
# @title DC Component vs Duty Cycle
#PLOT FOR DC COMPONENT VS DUTY CYCLE
DCindices = np.arange(1, 101)
DCcomponent = np.zeros(100)

for d in range (1, 101):
  DCcomponent[d-1] = (2.0 * (d / 100)) - 1

fig, (plot19) = plt.subplots(1)
plot19.plot(DCindices, DCcomponent)
plot19.set_title("DC Components vs Duty Cycle")
plot19.set_xlabel("Duty Cycle (%)")
plot19.set_ylabel("DC Component")
plot19.grid(True)
```



The following plot shows the fundamental component magnitudes of the Fourier Series for increasing duty cycles. The fundamental component is the sinusoid with the lowest frequency.

In [ ]:
```
# @title Fundamental Component vs Duty Cycle
#PLOT FOR FUNDAMENTAL VS DUTY CYCLE
FundamentalIndices = np.arange(1, 101, dtype = 'complex_')
FundamentalComponent = np.zeros(100, dtype = 'complex_')

for d in range (1, 101):
  FundamentalComponent[d-1] = np.abs((1 / (-1j * 2 * np.pi)) * \
   (2 * np.exp(-1j * 2 * np.pi * (d/100)) - 2))

fig, (plot20) = plt.subplots(1)
plot20.plot(FundamentalIndices, FundamentalComponent)
plot20.set_title("Fundamental Component vs Duty Cycle")
plot20.set_xlabel("Duty Cycle (%)")
plot20.set_ylabel("Fundamental Component Magnitude")
plot20.grid(True)
```



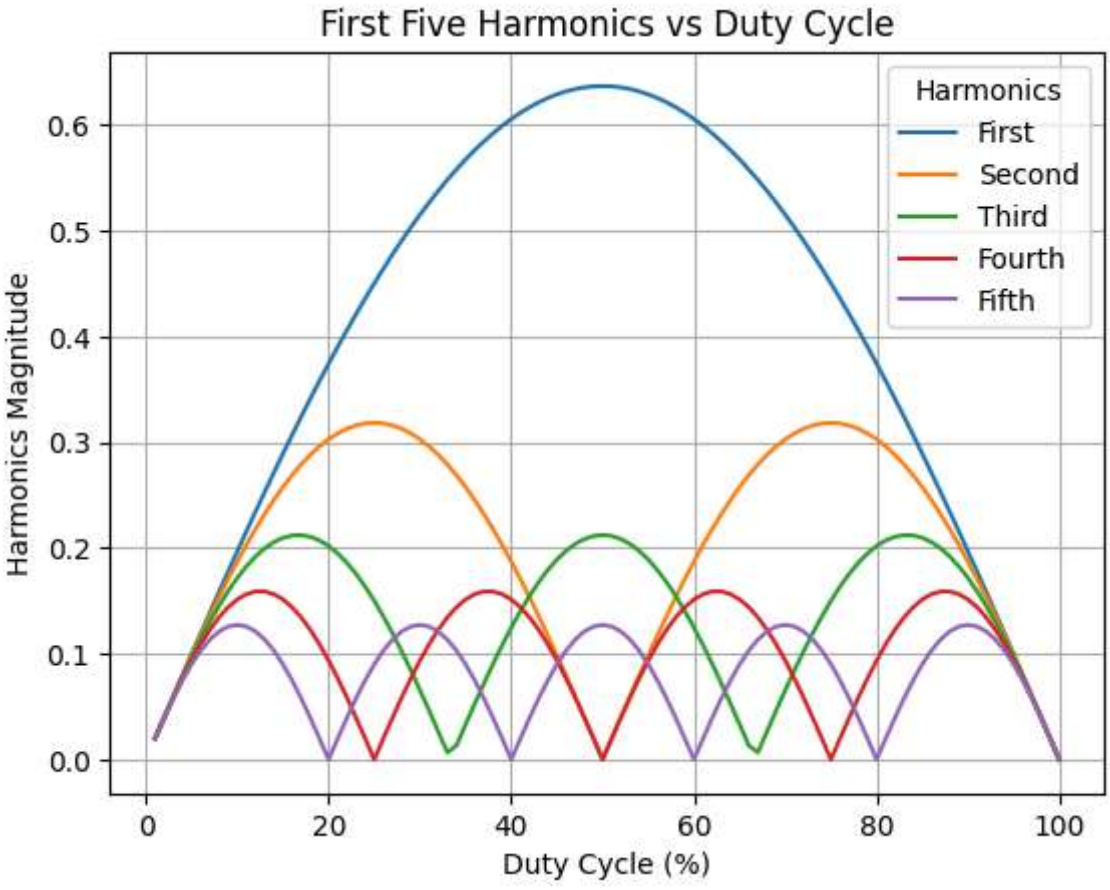
The following plot shows the first five harmonic magnitudes of the Fourier Series for increasing duty cycles. The plot validates the fact that harmonics are simply just multiples of the fundamental component.

In [ ]:

```
# @title First Five Harmonics vs Duty Cycle
#PLOT FOR FIRST FIVE HARMONICS VS DUTY CYCLE
FirstHarmonicIndices = np.arange(1, 101, dtype = 'complex_')
FirstHarmonics = np.zeros(100, dtype = 'complex_')
for d in range (1, 101):
  FirstHarmonics[d-1] = np.abs((1 / (-1j * 2 * np.pi) * \
   (2 * np.exp(-1j * 2 * np.pi * (d/100)) - 2)))

SecondHarmonicIndices = np.arange(1, 101, dtype = 'complex_')
SecondHarmonics = np.zeros(100, dtype = 'complex_')
for d in range (1, 101):
  SecondHarmonics[d-1] = np.abs((1 / (-1j * 2 * 2 * np.pi) * \
   (2 * np.exp(-1j * 2 * 2 * np.pi * (d/100)) - 2)))

ThirdHarmonicIndices = np.arange(1, 101, dtype = 'complex_')
ThirdHarmonics = np.zeros(100, dtype = 'complex_')
for d in range (1, 101):
  ThirdHarmonics[d-1] = np.abs((1 / (-1j * 3 * 2 * np.pi) * \
   (2 * np.exp(-1j * 3 * 2 * np.pi * (d/100)) - 2)))

FourthHarmonicIndices = np.arange(1, 101, dtype = 'complex_')
FourthHarmonics = np.zeros(100, dtype = 'complex_')
for d in range (1, 101):
  FourthHarmonics[d-1] = np.abs((1 / (-1j * 4 * 2 * np.pi) * \
   (2 * np.exp(-1j * 4 * 2 * np.pi * (d/100)) - 2)))
```

```
FifthHarmonicIndices = np.arange(1, 101, dtype = 'complex_')
FifthHarmonics = np.zeros(100, dtype = 'complex_')
for d in range (1, 101):
  FifthHarmonics[d-1] = np.abs((1 / (-1j * 5 * 2 * np.pi) * \
   (2 * np.exp(-1j * 5 * 2 * np.pi * (d/100)) - 2)))

fig, (plot21) = plt.subplots(1)
plot21.plot(FirstHarmonicIndices, FirstHarmonics, label = 'First')
plot21.plot(SecondHarmonicIndices, SecondHarmonics, label = 'Second')
plot21.plot(ThirdHarmonicIndices, ThirdHarmonics, label = 'Third')
plot21.plot(FourthHarmonicIndices, FourthHarmonics, label = 'Fourth')
plot21.plot(FifthHarmonicIndices, FifthHarmonics, label = 'Fifth')

plot21.set_title("First Five Harmonics vs Duty Cycle")
plot21.set_xlabel("Duty Cycle (%)")
plot21.set_ylabel("Harmonics Magnitude")
plot21.grid(True)
plot21.legend(title = 'Harmonics', loc = 'best')
```

Out[ ]: <matplotlib.legend.Legend at 0x7d355c7ee380>



This project focused on PWM signals with a set amplitude and frequency with varying duty cycles. The theory and derivations from these PWM signals can be extrapolated to find the Fourier Series expansion of signals with different amplitudes and frequencies. The derivations in this report are independent of an exact frequency or period, so the formula can be applied to all PWM signals switching between +1V and -1V (care must be taken when defining the duty cycle, D, which is the period that the signal is high divided by the fundamental period). However, a

change in amplitude would result in a different scaling factor to find the complex coefficients. A more general form of equation 24 would be:

$$\boxed{c_m = 2A\text{sinc}(m\pi D) \cdot De^{-jm\pi D}} \tag{34}$$

Where A is the absolute value of the amplitude of the signal. It is important to note that for this formula to be accurate, the absolute value of the high signal and the low signal of the PWM signal must be the same.

It is important to note that the Fourier Series can only be applied to periodic signals, such as a PWM, whereas the Fourier Transform can be applied to aperiodic signals.