UNIVERSIDAD DE VALPARAÍSO
Escuela de Ingeniería Civil Biomédica
CBM414 Procesamiento digital de señales biomédicas

Laboratory 1

Punto 1: 1+1
Punto 2: 1+0.5+0.5
Punto 3: 0.5+0.5

Bonificaciones:
- Inglés: 0.3
- Latex: 0.3

Total = 5 + 0.3+0.3=5.6

Maximiliano Antonio Gaete Pizarro
Andres Jesus Vega Araya

Problem 1:

Problem Statement

Implement the Fourier series for a square signal. To do this, create a function that takes as input arguments the period P, the time interval $t \in [-1, 1]$, and the number of coefficients $nfou$. The function should return the sum of the Fourier series (truncated to $nfou$ terms) of the square signal, i.e.,

$$\text{series} = \text{square\_signal\_fourier}(t, P, nfou)$$

The time interval should be subdivided by a number of samples $Nsample$. To do this, you can use the `linspace` function from numpy.

1. With $t \in [-1, 1]$, set $P = 2$ and $Nsample = 512$. Evaluate the Fourier series function in the interval t, sampled equidistantly, using $nfou = 10, 30, 100$.

   1 (a) (1 Point) For the three cases, plot the result of the Fourier series superimposed on the analytical signal and calculate the MSE between both signals.

   1 (b) (1 Point) Compare how the shape of the reconstructed signal varies. Include graphs and a brief explanation of how the number of coefficients affects the approximation of the square signal.

Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy import signal

# Senal cuadrada analitica
def senal_cuadrada(t, P):
    return signal.square(2 * np.pi * t / P, duty=0.5)
# Error cuadratico medio
def calcular_mse(senal_real, senal_aproximada):
    return np.mean((senal_real - senal_aproximada)**2)

def square_signal_fourier(t, P, nfou):
    series = np.zeros_like(t)
    for n in range(1, nfou + 1, 2):  # Only sum for n = 1, 3, 5, ...
        coef = 4 / (n * np.pi)
        series += coef * np.sin(2 * np.pi * n * t / P)
    return series

# Define parameters
P = 2
Nsample = 512
t = np.linspace(-1, 1, Nsample)

# Evaluate Fourier series for different nfou
nfou_values = [10, 30, 100]
series_results = [square_signal_fourier(t, P, nfou) for nfou in nfou_values]

# Plot results
plt.figure(figsize=(12, 8))
for i, nfou in enumerate(nfou_values):
    plt.subplot(3, 1, i+1)
    plt.plot(t, series_results[i], label=f'nfou = {nfou}')
    plt.plot(t, senal_cuadrada(t, P), color='red', label='Square Signal')
    plt.title(f'Fourier Series Truncated at {nfou} terms')
    plt.xlabel('Time t')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

The plots below show the Fourier series approximations for different values of $nfou$:

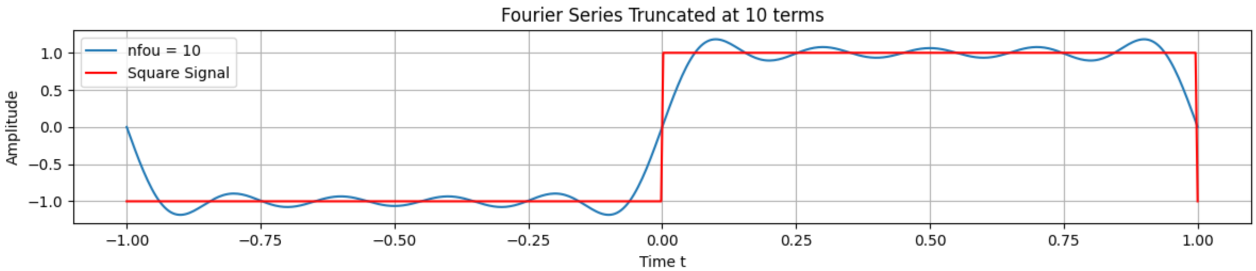

Figure 1: Fourier Series Truncated at 10 terms.

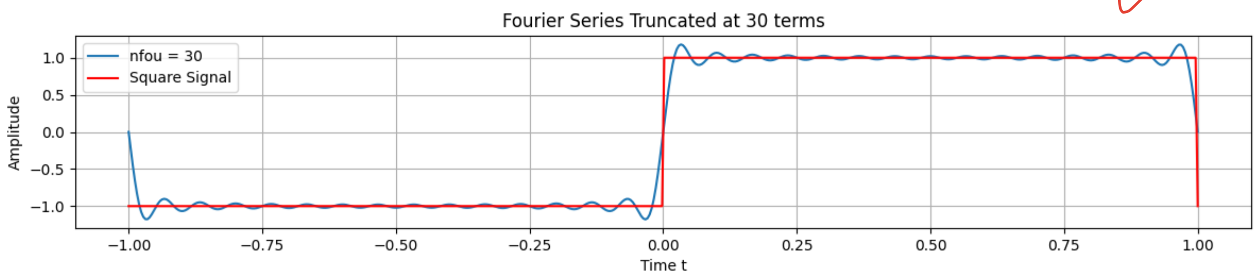

Figure 2: Fourier Series Truncated at 30 terms.

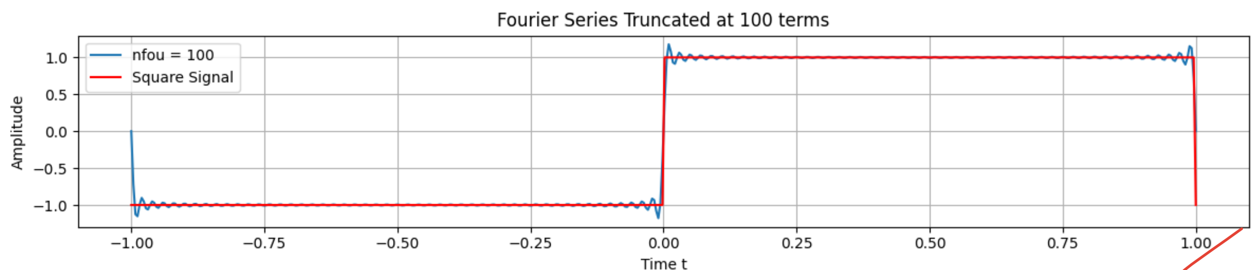

Figure 3: Fourier Series Truncated at 100 terms.

The Mean Squared Error (MSE) values for the different cases are:

```
mse_values = [calcular_mse(senal, serie) for serie in series_results]
mse_values
[0.04229492291155765, 0.01550770211169476, 0.006253270559774062]
```

Results and Discussion

The Mean Squared Error (MSE) decreases as $nfou$ increases:

- For $nfou = 10$: MSE ≈ 0.042.
- For $nfou = 30$: MSE ≈ 0.015.
- For $nfou = 100$: MSE ≈ 0.006.

Explanation: The Mean Squared Error (MSE) decreases as the number of Fourier coefficients $nfou$ increases because each additional term in the Fourier series contributes to a more accurate representation of the original square wave. The Fourier series approximates the square wave by summing sinusoidal functions with different frequencies and amplitudes. With a higher $nfou$, the series can capture more of the square wave's detail, particularly around the discontinuities (jumps between -1 and 1). This leads to a more precise approximation, thereby reducing the MSE. However, even as $nfou$ increases, some overshoot near the discontinuities, known as the Gibbs phenomenon, will persist, though it becomes less significant as more terms are added.

Problem 2:

Problem Statement

1. With $t \in [-1, 1]$, set $P = 2$ and $nfou = 50$. Evaluate the Fourier series function over the interval t, equispatially sampled, using $Nsamples = 64, 128, 512$.

1 (a) (2 Points) For all three cases, plot the Fourier series results superimposed on the analytical signal. Compare and analyze how the shape of the reconstructed signal varies. ↳ Faltó

0.5 (b) (0.5 Points) How does $Nsamples$ relate to the sampling frequency?

0.5 (c) (0.5 Points) Consider $Nsamples = 512$. Determine the maximum number of Fourier coefficients that can be used before encountering aliasing effects.

Python Implementation

```
# Define parameters
P = 2
nfou = 50

# Define different sample sizes
Nsamples_values = [64, 128, 512]

# Evaluate the Fourier series for different Nsamples
series_results = []
t_values = []
for Nsamples in Nsamples_values:
    t = np.linspace(-1, 1, Nsamples)
    t_values.append(t)
    series_results.append(square_signal_fourier(t, P, nfou))

# Plot the results
plt.figure(figsize=(12, 8))
for i, Nsamples in enumerate(Nsamples_values):
    plt.subplot(3, 1, i+1)
    plt.plot(t_values[i], series_results[i], label=f'Nsamples = {Nsamples}')
    plt.plot(t, senal_cuadrada(t, P), color='red', label='Square Signal')
    plt.title(f'Fourier Series Truncated at {nfou} terms with Nsamples = {Nsamples}')
    plt.xlabel('Time t')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

Results and Discussion

The following plots show the Fourier series approximations for different sample sizes $Nsamples = 64, 128, 512$:

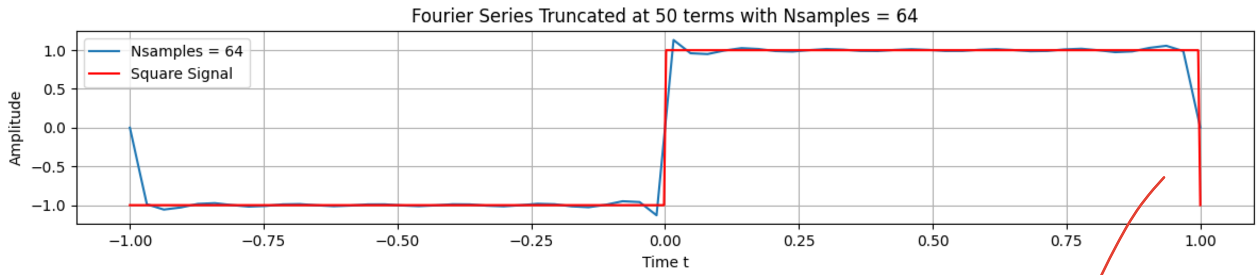

Figure 4: Fourier Series Truncated at 50 terms with $Nsamples = 64$.

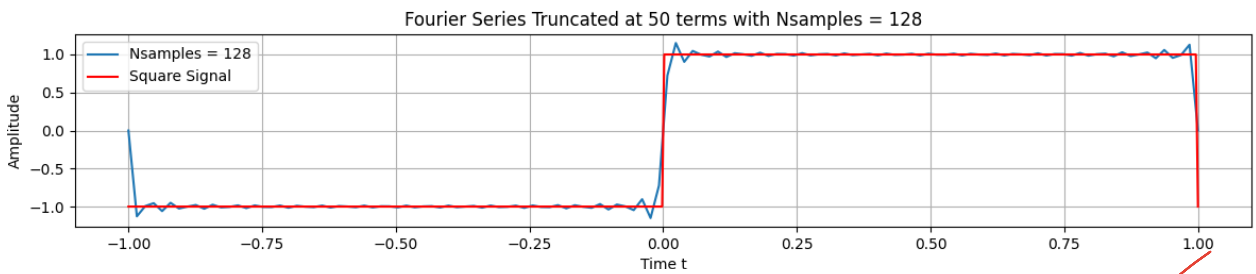

Figure 5: Fourier Series Truncated at 50 terms with $Nsamples = 128$.

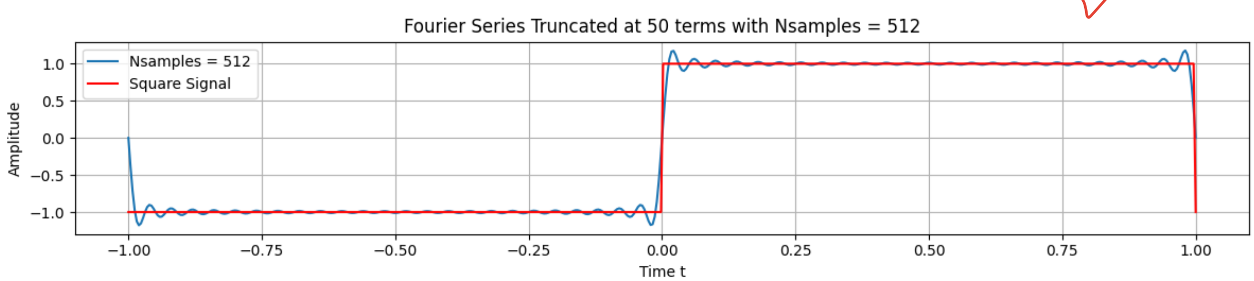

Figure 6: Fourier Series Truncated at 50 terms with $Nsamples = 512$.

Relationship between $Nsamples$ and Sampling Frequency $\boxed{0.5}$

- The sampling frequency f_s is directly related to the number of samples $Nsamples$ over the time interval:

$$f_s = \frac{Nsamples}{\text{duration of the interval}}$$

- With $Nsamples = 512$ and a time interval of $[-1, 1]$, the sampling frequency is $f_s = 256$ Hz.

Maximum Number of Fourier Coefficients to Avoid Aliasing $\boxed{0.5}$

- The maximum frequency that can be represented without aliasing is the Nyquist frequency, which is half the sampling frequency:

$$f_{\text{Nyquist}} = \frac{f_s}{2}$$

- The Fourier coefficients correspond to frequencies $\frac{n}{P}$. To avoid aliasing, the highest frequency should be less than or equal to the Nyquist frequency:

$$\frac{n_{\max}}{P} \leq f_{\text{Nyquist}}$$

- With $P = 2$, the maximum $n_{\max}$ is 255.

Explanation

- To avoid aliasing, the maximum number of Fourier coefficients that can be used is $nfou_{\max} = 255$. Since we use only odd terms (1, 3, 5,...), the maximum usable $nfou$ is 255.

Python Implementation 2

```
# Parameters
P = 2
Nsamples = 512
t = np.linspace(-1, 1, Nsamples)

# Determine the maximum number of coefficients before aliasing
nfou_max = 255  # Maximum number of odd terms before aliasing

# Evaluate the Fourier series using nfou_max
series_result = square_signal_fourier(t, P, nfou_max)

# Plot the result
plt.figure(figsize=(10, 6))
plt.plot(t, series_result, label=f'nfou = {nfou_max}')
plt.plot(t, senal_cuadrada(t, P), color='red', label='Square Signal')
plt.title(f'Fourier Series Truncated at {nfou_max} terms (Nsamples = {Nsamples})')
plt.xlabel('Time t')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```

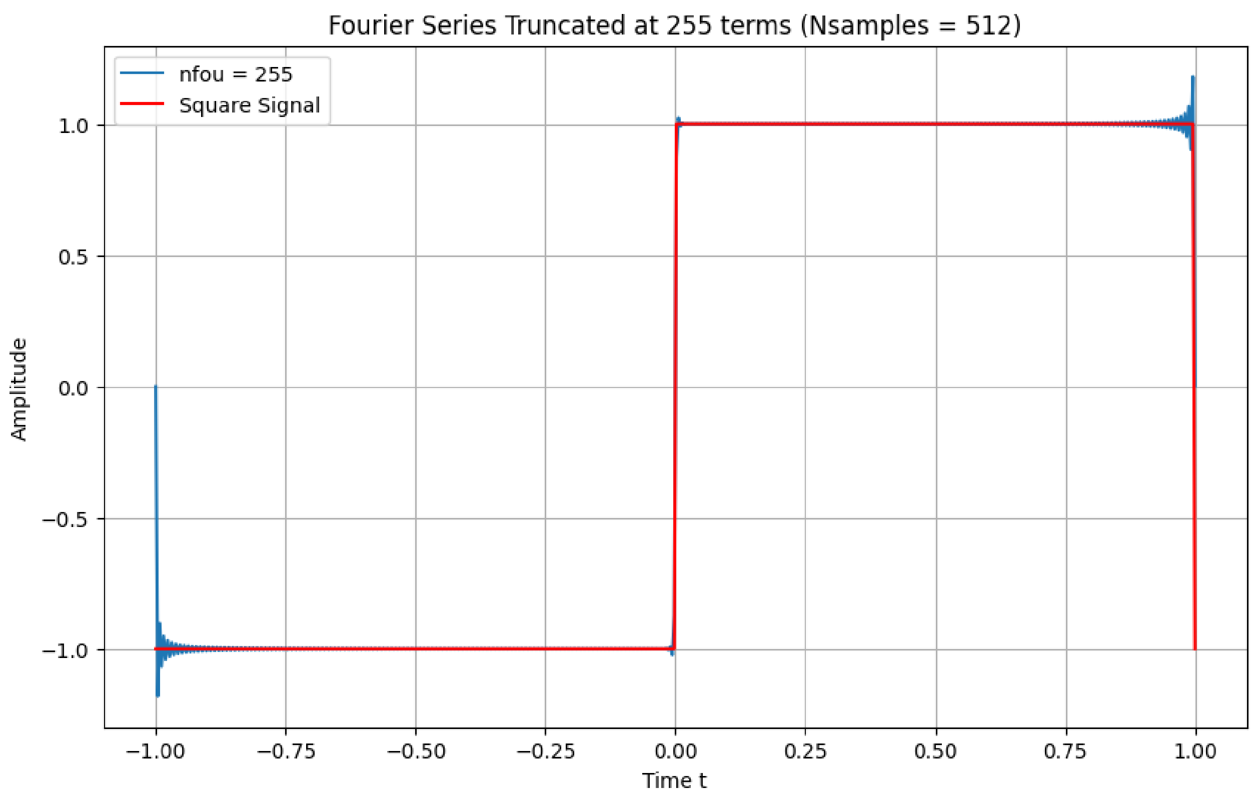

Figure 7: Fourier Series Truncated at 255 terms (Nsamples = 512).

Problem 3:

Problem Statement

1. With $t \in [-1, 1]$, set $nfou = 50$ and $Nsamples = 512$. Evaluate the Fourier series function over the interval t, equispatially sampled, using $P = 2, 1, 0.1$.

   0.5 (a) (1 Point) For all three cases, plot the Fourier series results superimposed on the analytical signal and calculate the MSE between both signals.

   0.5 (b) (1 Point) What happens to the MSE—does it increase, decrease, or remain the same? Explain this result.

Python Implementation

```
# Define parameters
nfou = 50
Nsamples = 512
t = np.linspace(-1, 1, Nsamples)

# Define different values of P
P_values = [2, 1, 0.1]

# Evaluate the Fourier series for different values of P
series_results = []
for P in P_values:
    series_results.append(square_signal_fourier(t, P, nfou))

serie = []
for P in P_values:
    serie.append(senal_cuadrada(t, P))

# Plot the results
plt.figure(figsize=(12, 8))
for i, P in enumerate(P_values):
    plt.subplot(3, 1, i+1)
    plt.plot(t, series_results[i], label=f'P = {P}')
    plt.plot(t, serie[i], color='red', label='Square Signal')
    plt.title(f'Fourier Series Truncated at {nfou} terms with P = {P}')
    plt.xlabel('Time t')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()

# Calculate MSE values
mse_values = [calcular_mse(senal, serie) for senal, serie in zip(square_signals,
    series_results)]

# Print MSE values
print(mse_values)  # [0.010169382571701559, 1.9891854477166597, 1.9823226326540
21]
```

serie

Results and Discussion

- For $P = 2$: MSE ≈ 0.0102 0.01012
- For $P = 1$: MSE ≈ 1.9892 0.01016
- For $P = 0.1$: MSE ≈ 1.9823 0 01016

Explanation: The increase in Mean Squared Error (MSE) as the period P decreases occurs because, with a lower P, the fundamental frequency of the square wave increases, leading to more rapid oscillations.

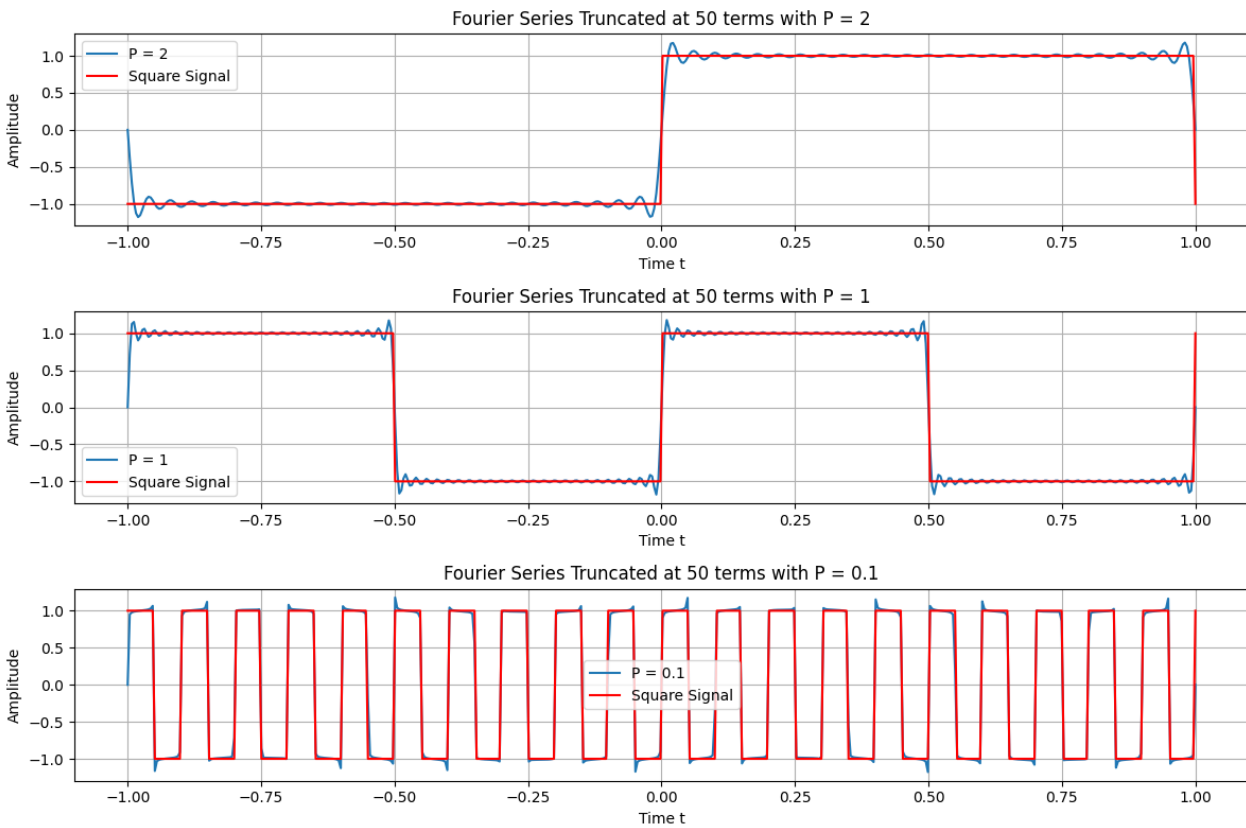

Figure 8: Fourier Series Truncated at 50 terms with $P = 2$, $P = 1$, and $P = 0.1$ respectively, superimposed on the analytical square signal.

A fixed number of Fourier terms ($nfou = 50$) is sufficient to capture the lower harmonics when P is large, but as P decreases, the signal contains higher frequency components that the Fourier series cannot represent accurately with just 50 terms. This limitation leads to a less accurate approximation of the square wave, resulting in a higher MSE. Essentially, the MSE rises as P decreases because the Fourier series is unable to adequately capture the increased frequency content of the signal with a limited number of terms.

Python Implementation 2

Esto no es del todo cierto. Los errores aumentan pero muy poco.

```
# Define parameters
nfou = 255
Nsamples = 512
t = np.linspace(-1, 1, Nsamples)

# Define different values of P
P_values = [2, 1, 0.1]

# Evaluate the Fourier series for different values of P
series_results = []
for P in P_values:
    series_results.append(square_signal_fourier(t, P, nfou))

serie = []
for P in P_values:
    serie.append(senal_cuadrada(t, P))

# Plot the results
plt.figure(figsize=(12, 8))
for i, P in enumerate(P_values):
```

```
    plt.subplot(3, 1, i+1)
    plt.plot(t, series_results[i], label=f'P = {P}')
    plt.plot(t, serie[i], color='red', label='Square Signal')
    plt.title(f'Fourier Series Truncated at {nfou} terms with P = {P}')
    plt.xlabel('Time t')
    plt.ylabel('Amplitude')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

Results

- For $P = 2$: MSE ≈ 0.0041
- For $P = 1$: MSE ≈ 1.9802
- For $P = 0.1$: MSE ≈ 1.9936

In summary, the Fourier series approximation becomes less accurate as the period of the square wave decreases, primarily due to the limitations in capturing high-frequency content with a finite number of Fourier terms. This leads to an increase in the MSE, even when the number of terms is increased significantly.