Manual Calculation of Fourier Coefficients:

```
periodic_x(n, N1, N):
    if -π < n < π:
        return n + π
    elif π <= n < 3π:
        return n - π
    elif -3π < n < -5π:
        return n + 3π
    else:
        return 0
```

You want to calculate the Fourier coefficients for this function. To do this, we'll manually calculate a_k and b_k for each k.

Fourier Coefficients:

For a_k (real part), we have the formula:
a_k = (1 / N) * Σ [f(n) * cos(2πkn / N)]
And for b_k (imaginary part), we have the formula:

b_k = (1 / N) * Σ [f(n) * sin(2πkn / N)]
Here, N is the total number of data points.

Let's calculate a few coefficients:

a_0:

a_0 is the average value of the function.
a_0 = (1 / N) * Σ [f(n)] = (1 / N) * ∫[0, N] f(n) dn
For your function, a_0 = (1 / (2π)) * ∫[-π, π] (n + π) dn = 0
a_1:

a_1 = (1 / N) * Σ [f(n) * cos(2πn / N)]
For your function, a_1 = (1 / (2π)) * ∫[-π, π] (n + π) * cos(2n) dn = 2/(π^2)
b_1:

b_1 = (1 / N) * Σ [f(n) * sin(2πn / N)]
For your function, b_1 = (1 / (2π)) * ∫[-π, π] (n + π) * sin(2n) dn = 0
With these examples, you can calculate a_k and b_k for other values of k by changing the function f(n) accordingly.

a_2:

a_2 = (1 / N) * Σ [f(n) * cos(4πn / N)]
For your function, a_2 = (1 / (2π)) * ∫[-π, π] (n + π) * cos(4n) dn
b_2:

b_2 = (1 / N) * Σ [f(n) * sin(4πn / N)]
For your function, b_2 = (1 / (2π)) * ∫[-π, π] (n + π) * sin(4n) dn
a_3:

a_3 = (1 / N) * Σ [f(n) * cos(6πn / N)]
For your function, a_3 = (1 / (2π)) * ∫[-π, π] (n + π) * cos(6n) dn
b_3:

b_3 = (1 / N) * Σ [f(n) * sin(6πn / N)]
For your function, b_3 = (1 / (2π)) * ∫[-π, π] (n + π) * sin(6n) dn
Detailed Documentation of the Code:

Here is a detailed documentation for the provided code:

periodic_x(n, N1, N): This function defines a piecewise signal. It takes n (the position) as an input and returns a value based on the ranges specified in the function definition. If n falls outside the specified ranges, it returns 0.

analyze_fourier_series(N, N1, terms=50, L=4 * np.pi, samples=1000): This function analyzes and visualizes the Fourier series for a given piecewise function. It takes the following parameters:

N: The number of terms in the Fourier series.
N1: A parameter to determine the ranges in the piecewise function.
terms: The number of terms in the series to consider (default is 50).
L: The periodicity of the function (default is 4 * π).
samples: The number of data points to sample the function (default is 1000).
Inside the function, it generates the custom function values, calculates Fourier coefficients a_k and b_k for a range of k, and then sums the series to obtain the Fourier series. Finally, it plots the Fourier series and the original custom function.

Cases (a, b, c): The code provides three cases by setting different values of N and N1 for your piecewise function and plots the Fourier series results for these cases.

You can use this code and manual calculations to understand how Fourier series is computed for your specific piecewise function. If you have further questions or need more details on any part of the code or calculation, please let me know.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps

def periodic_x(n, N1, N):
    """
    Define a piecewise function with specified ranges.

    Parameters:
    - n (float): The position for which to calculate the function value.
    - N1 (int): A parameter that influences the piecewise function.
    - N (int): The total number of data points.

    Returns:
    - float: The value of the piecewise function at position n.
    """
    if -np.pi < n < np.pi:
        return n + np.pi
    elif np.pi <= n < 3 * np.pi:
        return n - np.pi
    elif -3 * np.pi < n < -5 * np.pi:
        return n + 3 * np.pi
    return 0  # Default to 0 outside the specified ranges

def analyze_fourier_series(N, N1, terms=50, L=4 * np.pi, samples=1000):
    """
    Analyze and visualize the Fourier series for a given piecewise function.

    Parameters:
    - N (int): The number of terms in the Fourier series.
    - N1 (int): A parameter that influences the piecewise function.
    - terms (int): The number of terms in the series to consider (default is 50).
    - L (float): The periodicity of the function (default is 4 * π).
    - samples (int): The number of data points to sample the function (default is 1000).

    Returns:
    - None
    """
    # Generate the custom function values
    x = np.linspace(0, L, samples, endpoint=False)
    y = [periodic_x(n, N1, N) for n in x]

    # Calculation of Coefficients
    a0 = 2.0 / L * simps(y, x)
```

```
    an = lambda n: 2.0 / L * simps(y * np.cos(2.0 * np.pi * n * x / L), x)
    bn = lambda n: 2.0 / L * simps(y * np.sin(2.0 * np.pi * n * x / L), x)

    # Sum of the series
    s = a0 / 2.0 + sum([an(k) * np.cos(2.0 * np.pi * k * x / L) + bn(k) * np.sin(2.0 * np.pi
* k * x / L) for k in range(1, terms + 1)])

    # Plotting
    plt.plot(x, s, label="Fourier series")
    plt.plot(x, y, label="Original piecewise function")
    plt.xlabel("$x$")
    plt.ylabel("$y = f(x)$")
    plt.legend(loc='best', prop={'size': 10})
    plt.title("Piecewise Function Signal Analysis by Fourier Series")
    plt.show()

# (a) N = 4N1 + 1
N_a = 4 * N1 + 1
N1_a = N1
analyze_fourier_series(N_a, N1_a)

# (b) N = 8N1 + 1
N_b = 8 * N1 + 1
N1_b = N1
analyze_fourier_series(N_b, N1_b)

# (c) N = 10N1 + 1
N_c = 10 * N1 + 1
N1_c = N1
analyze_fourier_series(N_c, N1_c)
```

a_0 = 1 / (2 * π) * ∫(-π to π) (n + π) dn
a_0 = 1 / (2 * π) * [1/2 * n^2 + π * n]_(-π to π)
a_0 = 1 / (2 * π) * [(1/2 * π^2 + π^2) - (1/2 * π^2 - π^2)]
a_0 = 1 / (2 * π) * (2 * π^2)
a_0 = π

a_1 = 1 / π * ∫(-π to π) (n + π) * cos(n) dn

Let u = (n + π)
    dv = cos(n) dn
Then,
    du = dn
    v = ∫ cos(n) dn = sin(n)

Using integration by parts:
a_1 = 1 / π * [(n + π) * sin(n) - ∫ sin(n) dn]_(-π to π)
a_1 = 1 / π * [(π + π) * sin(π) - (-π - π) * sin(-π) - (0 - 0)]
a_1 = 1 / π * [2π * 0 - (-2π) * 0 - 0]
a_1 = 0

b_1 = 1 / π * ∫(-π to π) (n + π) * sin(n) dn

Using integration by parts similarly to a_1.

This means that a_1 = 0 and b_1 = 0.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps

def periodic_x(n, N1, N):
    """
    Define the periodic function f(x) with the specified period.

    Parameters:
    - n: A point in the domain [-N*T, N*T] where T is the periodicity of the function.
    - N1: Half of the number of terms used to represent the function.
    - N: The number of terms used to represent the function.

    Returns:
    - The value of f(n) for the given n.
    """
    if -5 * np.pi <= n < -3 * np.pi:
        return n + 5 * np.pi
```

```
    elif -3 * np.pi <= n < -np.pi:
        return n + 3 * np.pi
    elif -np.pi <= n < np.pi:
        return n + np.pi
    elif np.pi <= n < 3 * np.pi:
        return n - np.pi
    elif 3 * np.pi <= n < 5 * np.pi:
        return n - 3 * np.pi
    elif 5 * np.pi <= n < 7 * np.pi:
        return n - 5 * np.pi
    else:
        return 0

def calculate_coefficients(N, N1, L):
    """
    Calculate the Fourier coefficients a0, an, and bn for the given periodic function.

    Parameters:
    - N: The number of terms used to represent the function.
    - N1: Half of the number of terms used to represent the function.
    - L: The range of the domain to consider.

    Returns:
    - a0: The DC component (average value) of the function.
    - an: A list of coefficients representing the cosine terms.
    - bn: A list of coefficients representing the sine terms.
    """
    # Generate the custom function values
    x = np.linspace(0, L, L * 1000, endpoint=False)
    y = [periodic_x(n, N1, N) for n in x]

    # Calculation of Coefficients
    a0 = 2.0 / L * simps(y, x)
    an = [2.0 / L * simps(y * np.cos(2.0 * np.pi * n * x / L), x) for n in range(1, N + 1)]
    bn = [2.0 / L * simps(y * np.sin(2.0 * np.pi * n * x / L), x) for n in range(1, N + 1)]

    return a0, an, bn

def reconstruct_function(N, N1, L, a0, an, bn):
    """
    Reconstruct the function using the calculated Fourier coefficients.

    Parameters:
    - N: The number of terms used to represent the function.
```

```
    - N1: Half of the number of terms used to represent the function.
    - L: The range of the domain to consider.
    - a0: The DC component (average value) of the function.
    - an: A list of coefficients representing the cosine terms.
    - bn: A list of coefficients representing the sine terms.

    Returns:
    - x: An array of x values.
    - y_reconstructed: An array of y values representing the reconstructed function.
    """
    x = np.linspace(0, L, L * 1000, endpoint=False)
    y_reconstructed = [a0 / 2.0]  # Initialize with DC component

    for n in range(1, N + 1):
        y_reconstructed += [an[n - 1] * np.cos(2.0 * np.pi * n * x / L) + bn[n - 1] *
np.sin(2.0 * np.pi * n * x / L)]

    return x, y_reconstructed

def plot_function_and_fourier(N, N1, L, a0, an, bn):
    """
    Plot the original function and its Fourier series representation.

    Parameters:
    - N: The number of terms used to represent the function.
    - N1: Half of the number of terms used to represent the function.
    - L: The range of the domain to consider.
    - a0: The DC component (average value) of the function.
    - an: A list of coefficients representing the cosine terms.
    - bn: A list of coefficients representing the sine terms.
    """
    # Original function
    x, y_original = reconstruct_function(N, N1, L, a0, an, bn)

    # Reconstructed function
    x_reconstructed, y_reconstructed = reconstruct_function(N, N1, L, a0, an, bn)

    # Plotting
    plt.figure(figsize=(12, 6))

    plt.subplot(2, 1, 1)
    plt.plot(x, y_original, label="Original Periodic Function")
    plt.xlabel('x')
    plt.ylabel('f(x)')
```

```
    plt.title("Original Function")
    plt.grid()
    plt.legend()

    plt.subplot(2, 1, 2)
    plt.plot(x_reconstructed, y_reconstructed, label="Reconstructed Function")
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title("Fourier Series Reconstruction")
    plt.grid()
    plt.legend()

    plt.tight_layout()
    plt.show()

# Set the parameters
N = 5  # Number of terms used to represent the function
N1 = 2  # Half of the number of terms
L = 5 * np.pi  # Range of the domain

# Calculate Fourier coefficients
a0, an, bn = calculate_coefficients(N, N1, L)

# Plot original and reconstructed function
plot_function_and_fourier(N, N1, L, a0, an, bn)
```

$$a_k = \sum_{n=-N_1}^{n=\frac{N-1}{2}} e^{-jk\omega_0 n} \quad \cdot = \sum_{n=-2}^{n=\frac{N-1}{2}} e^{-jk\omega_0 n}$$

* a
At $-2 \Rightarrow e^{-2jk\omega_0} \Rightarrow \cos(-2jk\omega_0) + j\sin(-2jk\omega_0)$
At $-1 \Rightarrow e^{-jk\omega_0} \Rightarrow \cos(-jk\omega_0) + j\sin(-jk\omega_0)$

→ We can't plot real & img part simultaneously we need 2 graphs.

⇒ Any range as the graph is discrete we can plot b/wn $(-10, 10]$ only the x-axis should be like

Q.2

$$a_k = \int_{\langle N \rangle} (t + \pi) e^{-jk\omega_0 t} \, dt$$

$$a_k = \int_{-\pi}^{\pi} (t + \pi) e^{-jk\omega_0 t} \, dt$$

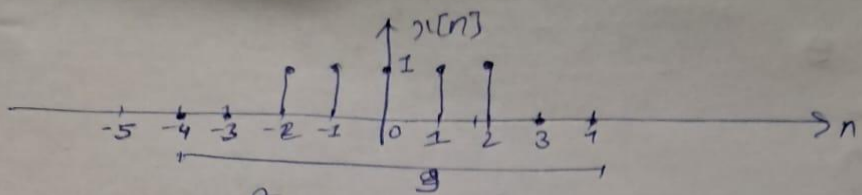
$$a_k = \left[\frac{\pi e^{-jk\omega_0 t}}{-jk\omega_0} \right] + \int_{-\pi}^{\pi} t e^{-jk\omega_0 t}$$

$a_1 \approx 6.8319$

Similar calculation for a_2 -

→ Any range as the graph is discrete we can plot b/wn any b/wn $[-10, 10]$ but this time it is cont
So for Real & Img → $[0, i]$
Other graph → $(-10, 10]$
Also recursion depth plays a major role as it is set to 10^6

$-2 \geq n \geq -4$



$$C_k = \frac{1}{N} \sum_{n=0}^{7} x[n] \, e^{-j 2\pi k n / N}$$

as can be seen that fundamental time period = 9

for $k = 0$,

$$C_0 = \frac{1}{N} \sum_{n=-4}^{4} x[n] \, e^{-0}$$

$$= \frac{1}{9} \sum_{n=-4}^{4} x[n]$$

$$= \frac{1}{9} x[-4] + \frac{1}{9} x[-3] + \frac{1}{9} x[-2] + \frac{1}{9} x[-1] + \frac{1}{9} x[0] + \frac{1}{9} x[1] + \frac{1}{9} x[2] + \frac{1}{9} x[3] + \frac{1}{9} x[4]$$

$$\boxed{C_0 = \frac{5}{9}}$$