In [ ]:
```python
import math
import numpy as np
import matplotlib.pyplot as plt
import sympy as sm
import scipy as scp
```

In [ ]:
```python
def gauleg(x1, x2, x, w, n):
    EPS = 3.0e-11
    m = (n + 1) // 2  # Find only half the roots because of symmetry
    xm = 0.5 * (x2 + x1)
    xl = 0.5 * (x2 - x1)
    for i in range(1, m + 1):
        z = math.cos(math.pi * (i - 0.25) / (n + 0.5))
        while True:
            p1 = 1.0
            p2 = 0.0
            for j in range(1, n + 1):
                # Recurrence relation
                p3 = p2
                p2 = p1
                p1 = ((2.0 * j - 1.0) * z * p2 - (j - 1.0) * p3) / j
            # Derivative
            pp = n * (z * p1 - p2) / (z * z - 1.0)
            z1 = z
            # Newton’s method
            z = z1 - p1 / pp
            if abs(z - z1) <= EPS:
                break
        x[i] = xm - xl * z
        x[n + 1 - i] = xm + xl * z
        # Weights
        w[i] = 2.0 * xl / ((1.0 - z * z) * pp * pp)
        w[n + 1 - i] = w[i]
```

In [ ]:
```python
def Gauss_Legendre_Quad(fs, weights: np.ndarray, zeros: np.ndarray, alpha: float = -1.0, beta: float = 1.0) :
    if beta <= alpha:
        print("Alpha is greater than (or equal to) beta: alpha: ", alpha, " beta: ", beta)
        return
    sum = 0
    scaled_zeros = np.zeros_like(zeros)
    for i in range(len(zeros)):
        scaled_zeros[i] = 0.5*(beta - alpha) * zeros[i] + 0.5*(beta + alpha)
    n = len(weights)
    x = sm.symbols('x')
    for i in range(n):
        sum += 0.5*(beta - alpha) * weights[i] * fs.subs(x, scaled_zeros[i])
    return sum
```

In [ ]:
```python
# For Legendre polynomials
def calculate_coeffs(func, leg_funcs):
    x = sm.symbols('x')
    coeffs = np.zeros(len(leg_funcs))
    for l in range(len(coeffs)):
        inte = sm.integrate(func*leg_funcs[l], (x, -1, 1))
        coeffs[l] = 0.5*(2*l + 1) * inte
    return coeffs

def calculate_leg_funcs(n: int):
    x = sm.symbols('x')
    leg_funcs = []
    for deg in range(n):
        leg = 0
        for k in range(n):
            leg += scp.special.binom(deg, k)**2.0 *(x - 1)**(deg - k) * (x+1)**k
        leg *= 2**(-deg)
        leg_funcs.append(leg)

    return leg_funcs

def calculate_L2_norm(func, coeffs, leg_funcs):
    approx = 0
    xspan = np.linspace(-1, 1, 1000)
    x = sm.symbols('x')
    for k in range(len(coeffs)):
        if not math.isnan(coeffs[k]):
            approx += coeffs[k] * leg_funcs[k]
    integrand = sm.lambdify(x, abs(approx - func)**2)
    inte = scp.integrate.trapz(integrand(xspan), xspan)

    inte = inte**(0.5)
    return inte
```

In [ ]:
```python
x = sm.symbols('x')
func = 1.0/(x+3)
# Input the number of quadrature points
ns = [16]
j = 50
errors = np.zeros(j)

leg_funcs = calculate_leg_funcs(16)
coeffs = calculate_coeffs(func, leg_funcs)
inte = calculate_L2_norm(func, coeffs, leg_funcs)

for n, N in enumerate(ns):
    xs = [0.0] * (N + 1)
    ws = [0.0] * (N + 1)

    # Call the gauleg function
```

```python
    gauleg(-1.0, 1.0, xs, ws, N)

    for k in range(len(errors)):
        errors[k] = inte*Gauss_Legendre_Quad(func, ws, xs, 0.5**(k+1), 0.5**(k))
    for j in range(len(errors)-1):
        errors[j] = errors[j] - errors[j+1]
```

In [ ]:

```python
plt.figure(figsize=(10, 6))

plt.scatter(range(len(errors)-1), errors[:-1])
plt.yscale('log')
plt.gca().set_facecolor((0.9, 0.9, 0.9))
plt.grid(True)
plt.gca().set_axisbelow(True)
plt.xlabel("k")
plt.title(r"Error in approximating $f(x) = \frac{1}{x+3}$ using 16 Legendre Polynomials")
plt.show()
```