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

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) :
    sum = 0
    n = len(weights)
    y = sm.symbols('y')
    for i in range(n):
        sum += weights[i] * fs.subs(y, zeros[i])
    return sum

y = sm.symbols('y')
x = sm.symbols('x')
funcs = [sm.exp(-(x - y)**2), x*sm.exp(-(x - y)**2)]
```

In [ ]:
```python
# Input the number of quadrature points
ns = [40, 80]
f_res = np.zeros((2*len(ns), 100))
xspan = np.linspace(-1, 1, 100)
# Allocate arrays x and w

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

    # Call the gauleg function
    gauleg(-1.0, 1.0, xs, ws, N)

    for i, fs in enumerate(funcs):
        # Calculate the integration
        res = Gauss_Legendre_Quad(fs, ws, xs)

        for k in range(len(xspan)):
            f_res[i*len(ns) + n, k] = res.subs(x, xspan[k])
```

In [ ]:
```python
fig, axs = plt.subplots(1, 2, figsize = (20, 10))

for i in range(len(axs)):
    for j in range(len(ns)):
        axs[i].plot(xspan, f_res[j + i*len(ns)], label = f"{ns[j]}")
        axs[i].legend()

    axs[i].grid(True)
    axs[i].set_facecolor("#E6E6E6")
    axs[i].set_title(f"{funcs[i]}")
```

In [ ]:
```python
fig, axs = plt.subplots(1, 2, figsize = (20, 10))

for i in range(len(axs)):
    axs[i].plot(xspan, np.abs(f_res[0 + i*len(ns)] - f_res[1 + i*len(ns)]))
    axs[i].grid(True)
    axs[i].set_yscale('log')
    axs[i].set_facecolor("#E6E6E6")
    axs[i].set_title(f"Error for {funcs[i]}")
```