Library Imports

In [ ]:
```
import numpy as np
import plotly.graph_objects as go
from scipy.integrate import quad
from scipy.special import roots_legendre
import matplotlib.pyplot as plt
import math
```

sources

https://pomax.github.io/bezierinfo/legendre-gauss.html

Method 1

Gaussian Legendre Quadrature

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i \cdot f(x_i)$$

In [ ]:
```
def f(x):
    return 1 / (2 - np.sqrt(x))

def gaussian_quadrature(func, a, b, n_points):
    x, w = roots_legendre(n_points)
    transformed_x = 0.5 * (b - a) * x + 0.5 * (a + b)
    fx = func(transformed_x)
    return 0.5 * (b - a) * np.sum(w * fx)
```

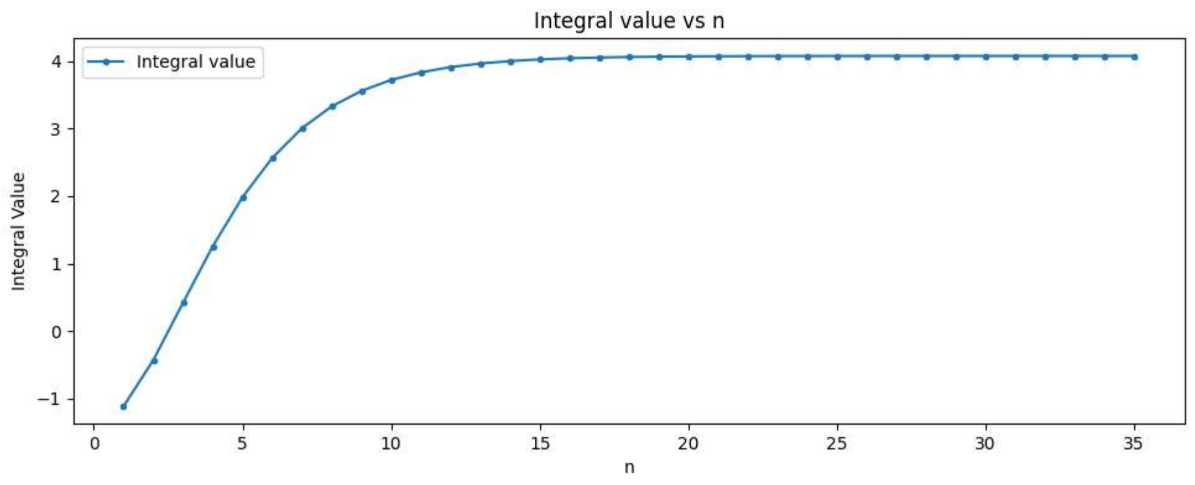
I am using a library to pull the roots/weights from the Legendre polynomial of n_th degree

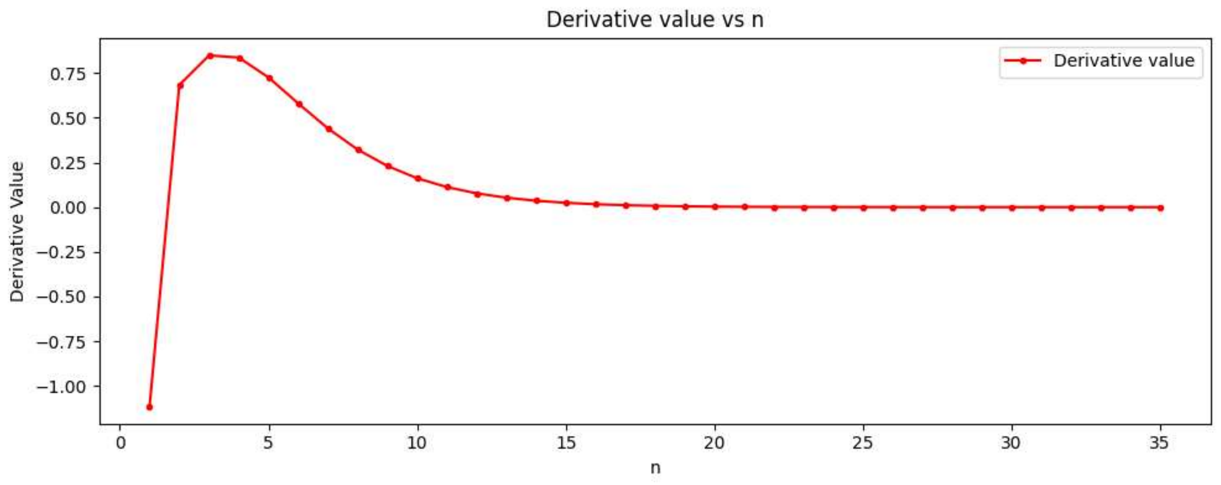
https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.roots_legendre.html

In [ ]:
```
def adaptive_quadrature(func, a, b, n_points):
    # Split points around the singularity
    split_point1 = 3.99
    split_point2 = 4.01

    # Integrate over the three subintervals
    integral1 = gaussian_quadrature(func, a, split_point1, n_points)
    integral2 = gaussian_quadrature(func, split_point1, split_point2, n_points)
    integral3 = gaussian_quadrature(func, split_point2, b, n_points)

    # Return the sum of the three integrals
    return integral1 + integral2 + integral3
```

```
integral_values = []
derivatives = []
n_points = 1
prev_integral = 0
first_iteration = True

while True:
    current_integral = adaptive_quadrature(f, 0, 5, n_points)

    if np.isinf(current_integral):
        n_points += 1
        continue

    derivative = current_integral - prev_integral
    integral_values.append(current_integral)
    derivatives.append(derivative)

    if not first_iteration and abs(derivative) < 1e-5:
        break

    prev_integral = current_integral
    first_iteration = False
    n_points += 1

# Generate ns based on the length of integral_values
ns = list(range(1, len(integral_values) + 1))

# Plotting
fig, axs = plt.subplots(2, 1, figsize=(10, 8))
axs[0].plot(ns, integral_values, '-o', markersize=3, label="Integral value")
axs[0].set_title("Integral value vs n")
axs[0].set_xlabel("n")
axs[0].set_ylabel("Integral Value")
axs[0].legend()

axs[1].plot(ns, derivatives, '-o', markersize=3, color="red", label="Derivative val
axs[1].set_title("Derivative value vs n")
axs[1].set_xlabel("n")
axs[1].set_ylabel("Derivative Value")
axs[1].legend()

plt.tight_layout()
plt.show()

# Printing the final value of n and the integral value
print(f"Final value of n: {n_points}")
print(f"Final integral value: {current_integral:.3f}")
```

```
c:\Users\AaronSnyder\.virtualenvs\eng_math-tu8MGJB8\lib\site-packages\ipykernel_laun
cher.py:2: RuntimeWarning: divide by zero encountered in true_divide
```





```
Final value of n: 70
Final integral value: 4.075
```

Method 2

In [ ]:
```python
# Define the function to be integrated
def f(x):
    return 1 / (2 - np.sqrt(x))

# Integrate the function from 0 to 3.99 and from 4.01 to 5
integral_value_1, _ = quad(f, 0, 3.99999)
integral_value_2, _ = quad(f, 4.00001, 5)

# Sum the two integrals
total_integral = integral_value_1 + integral_value_2

print(f"Value of I from 0 to 5: {total_integral:.5f}")
```

```
Value of I from 0 to 5: 4.07500
```