4. [**15 pt**] If $K = \max([A.B, C])$, solve the following using the method of Frobenius.
Here A, B, C, D are taken from your TUID.

$$\boxed{x\,y'' + 2y' + y = 0; \quad y(1) = -10; \quad y(5) = 2K \times (-1)^D}$$

(a) Compute your series solution using the nested recursion method, and plot this series-solution answer on $1 \le x \le 5$.
(b) Plot the "Bessel trick" solution on the same plot for comparison.

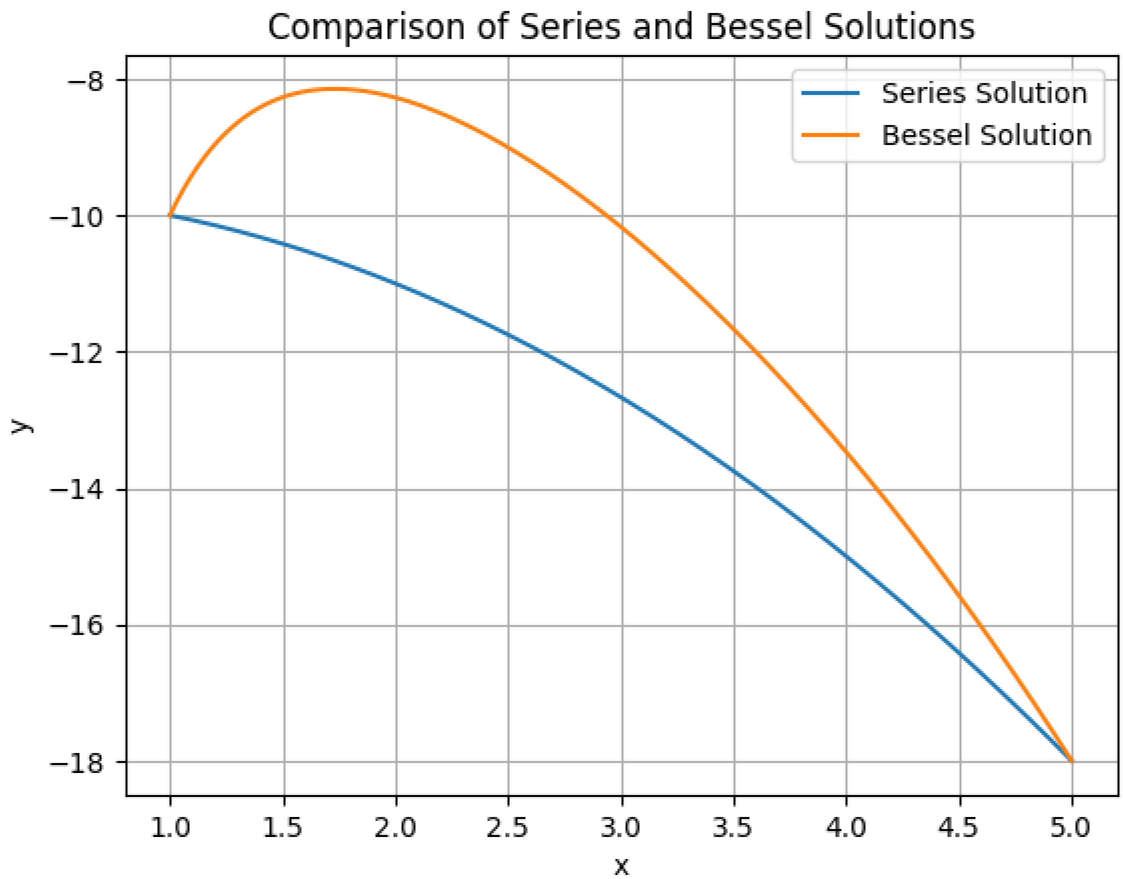
In [ ]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import scipy.special
from scipy.optimize import fsolve
```

In [ ]:
```
# Define u1 and u2 using Bessel functions
u1 = lambda x: 1/math.sqrt(x) * scipy.special.i0(2 * np.sqrt(x))  # Modified Bessel
u2 = lambda x: 1/math.sqrt(x) * scipy.special.k0(2 * np.sqrt(x))  # Modified Bessel

# Compute u1 and u2 at given boundary points (1 and 5)
u1_values = [u1(1), u1(5)]
u2_values = [u2(1), u2(5)]

# Solve the system of equations to determine coefficients using boundary conditions
CC = np.linalg.solve(np.array([[u1_values[0], u2_values[0]], [u1_values[1], u2_valu

# Define the solution using Bessel
def bessel_trick(x):
    return CC[0] * u1(x) + CC[1] * u2(x)  # Linear combination of the Bessel soluti

# Define the series solution to the differential equation using Frobenius method
def series(x, N=50):
    # Initial values for the indices of the Frobenius method (indicial roots)
    lambda_1, lambda_2 = 0, -1

    # Initialize coefficients arrays
    C1 = np.zeros(N)
    C2 = np.zeros(N)

    # Initial coefficients for the two solutions
    C1[0], C1[1] = 1, 0
    C2[0], C2[1] = 0, 1

    # Compute coefficients for lambda_1 using the recursive relation
    for k in range(2, N):
        denominator = k * (k-1) * (2*k)
        if denominator != 0:
            C1[k] = -C1[k-1] / denominator

    # Compute coefficients for lambda_2 using the recursive relation
    for k in range(3, N):
        denominator = (k-1) * (k-2) * (2*k-2)
        if denominator != 0:
```

```
            C2[k] = -C2[k-1] / denominator

    # Define the series solution using the coefficients
    def y(x_val, A, B):
        return sum([A * C1[k] * (x_val ** k) + B * C2[k] * (x_val ** (k+1)) for k i

    # Function to compute the discrepancies between boundary conditions and current
    def equations(vars):
        A, B = vars
        eq1 = y(1, A, B) - (-10)  # Boundary condition at x=1
        # print(eq1)
        eq2 = y(5, A, B) - (-18)  # Boundary condition at x=5
        # print(eq2)
        return [eq1, eq2]

    # Solve for coefficients A and B using the boundary conditions
    A, B = fsolve(equations, [1, 1])

    # Return the value of the solution at the given x
    return y(x, A, B)

# Define the x values for plotting the solutions
x = np.linspace(1, 5, 400)

# Compute the series and Bessel solutions for the given x values
y_vals_series = [series(val) for val in x]
y_vals_bessel = [bessel_trick(val) for val in x]
```

In [ ]:

```
# Plotting the results
plt.figure()
plt.plot(x, y_vals_series, label="Series Solution")
plt.plot(x, y_vals_bessel, label="Bessel Solution")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.title("Comparison of Series and Bessel Solutions")
plt.grid(True)
plt.show()
```



given the large discreptetency I think I did something wrong. I solved the the recursions multiple times and got the same solution. Not sure where I went wrong?