

#part 1

import numpy as np

import math

x\_values = [1.70, 1.74, 1.78, 1.82, 1.86]

y\_values = [0.9916, 0.9857, 0.9781, 0.9691, 0.9584]

diff = [

[y\_values[0],

y\_values[1] - y\_values[0],

y\_values[2] - 2 \* y\_values[1] + y\_values[0],

y\_values[3] - 3 \* y\_values[2] + 3 \* y\_values[1] - y\_values[0],

y\_values[4] - 4 \* y\_values[3] + 6 \* y\_values[2] - 4 \* y\_values[1] + y\_values[0]]

]

t = (1.72 - 1.70) / h

def derivative\_newton\_forward(t, h, diff):

result = diff[0][1]

if len(diff[0]) >= 3:

result += ((2 \* t - 1) / 2) \* diff[0][2]

if len(diff[0]) >= 4:

result += ((3 \* t\*\*2 - 6 \* t + 2) / 6) \* diff[0][3]

if len(diff[0]) >= 5:

result += ((4 \* t\*\*3 - 18 \* t\*\*2 + 22 \* t - 6) / 24) \* diff[0][4]

return result / h

approx\_cos\_1\_72 = derivative\_newton\_forward(t, h, diff)

print(f"Approximate cos(1.72): {approx\_cos\_1\_72}")

#part 2

import numpy as np

x\_values = [1.70, 1.74, 1.78, 1.82, 1.86]

y\_values = [0.9916, 0.9857, 0.9781, 0.9691, 0.9584]

h = x\_values[1] - x\_values[0]

diff = [[

y\_values[-1],

y\_values[-1] - y\_values[-2],

y\_values[-1] - 2\*y\_values[-2] + y\_values[-3],

y\_values[-1] - 3\*y\_values[-2] + 3\*y\_values[-3] - y\_values[-4],

y\_values[-1] - 4\*y\_values[-2] + 6\*y\_values[-3] - 4\*y\_values[-4] + y\_values[-5]

]]

t = (1.84 - x\_values[-1]) / h

def derivative\_newton\_backward(t, h, diff):

result = diff[0][1]

if len(diff[0]) >= 3:

result += ((2 \* t + 1) / 2) \* diff[0][2]

if len(diff[0]) >= 4:

result += ((3 \* t\*\*2 + 6 \* t + 2) / 6) \* diff[0][3]

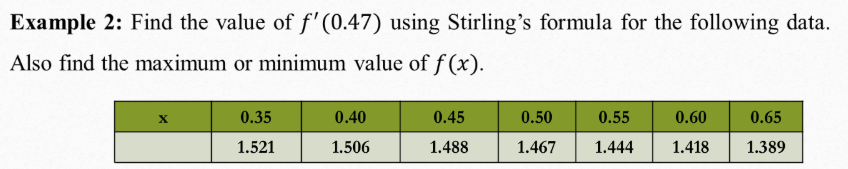
if len(diff[0]) >= 5:

result += ((4 \* t\*\*3 + 18 \* t\*\*2 + 22 \* t + 6) / 24) \* diff[0][4]

return result / h

approx\_cos\_1\_84 = derivative\_newton\_backward(t, h, diff)

print(f"Approximate cos(1.84): {approx\_cos\_1\_84:.6f}")



import numpy as np

x\_values = [0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65]

y\_values = [1.521, 1.506, 1.488, 1.467, 1.444, 1.418, 1.389]

h = x\_values[1] - x\_values[0] # uniform interval = 0.05

n = len(x\_values)

# Build the difference table

diff\_table = np.zeros((n, n))

diff\_table[:, 0] = y\_values

for j in range(1, n):

for i in range(n - j):

diff\_table[i][j] = diff\_table[i+1][j-1] - diff\_table[i][j-1]

# Find central value index near 0.47

mid\_index = 2 # x = 0.45 (3rd value) is closest to 0.47

x0 = x\_values[mid\_index]

p = (0.47 - x0) / h

# Apply Stirling’s formula for the first derivative:

dy\_dx = (1 / h) \* (

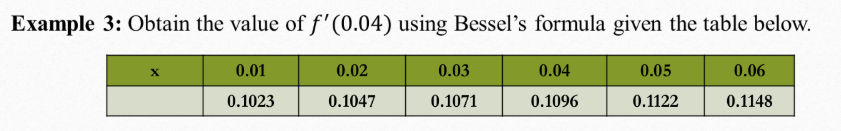
(diff\_table[mid\_index][1] + diff\_table[mid\_index - 1][1]) / 2

+ p \* (diff\_table[mid\_index - 1][2] / 2 + diff\_table[mid\_index - 1][2] / 2) # approximate central second difference

+ (p\*\*2 - 1) \* (diff\_table[mid\_index - 1][3] + diff\_table[mid\_index - 2][3]) / 4

)

print(f"Approximate derivative at x = 0.47 using Stirling's formula: {dy\_dx:.6f}")



import numpy as np

x\_values = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06]

y\_values = [0.1023, 0.1047, 0.1071, 0.1096, 0.1122, 0.1148]

h = x\_values[1] - x\_values[0] # Assuming equal spacing

n = len(x\_values)

# Build forward difference table

diff\_table = np.zeros((n, n))

diff\_table[:, 0] = y\_values

for j in range(1, n):

for i in range(n - j):

diff\_table[i][j] = diff\_table[i + 1][j - 1] - diff\_table[i][j-1]

mid\_index = 2 # y at x=0.03

p = (0.04 - x\_values[mid\_index]) / h

# Bessel’s first derivative formula components:

dy\_dx = (1 / h) \* (

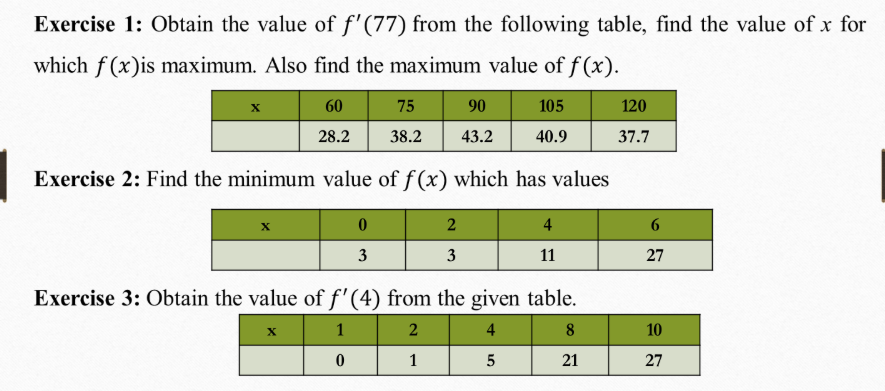
diff\_table[mid\_index][1] + diff\_table[mid\_index + 1][1]

) / 2

dy\_dx += (p - 0.5) \* diff\_table[mid\_index][2]

dy\_dx += ((3 \* p\*\*2 - 1) / 12) \* (diff\_table[mid\_index][3] + diff\_table[mid\_index + 1][3]) / 2

print(f"Approximate derivative at x = 0.04 using Bessel's formula: {dy\_dx:.6f}")



# Exercise 1:

import numpy as np

x\_values = [60, 75, 90, 105, 120]

y\_values = [28.2, 38.2, 43.2, 40.9, 37.7]

h = x\_values[1] - x\_values[0] # step size

n = len(x\_values)

# Build forward difference table

diff\_table = np.zeros((n, n))

diff\_table[:, 0] = y\_values

for j in range(1, n):

for i in range(n - j):

diff\_table[i][j] = diff\_table[i+1][j-1] - diff\_table[i][j-1]

# Stirling’s formula around 75

mid\_index = 1 # x = 75

p = (77 - x\_values[mid\_index]) / h

fprime\_77 = (1/h) \* (

(diff\_table[mid\_index][1] + diff\_table[mid\_index-1][1]) / 2

+ (p/2) \* diff\_table[mid\_index-1][2]

+ ((2\*p\*\*2 - 1)/6) \* ((diff\_table[mid\_index][3] + diff\_table[mid\_index-1][3]) / 2)

)

print(f"Approximate f'(77) = {fprime\_77:.6f}")

print(f"Maximum value of f(x) = {max(y\_values)} at x = {x\_values[y\_values.index(max(y\_values))]}")

#Exercise 2:

x\_values = [0, 2, 4, 6]

y\_values = [3, 3, 11, 27]

min\_val = min(y\_values)

x\_min = x\_values[y\_values.index(min\_val)]

print(f"Minimum value of f(x) = {min\_val} at x = {x\_min}")

#Exercise 3:

import numpy as np

x\_values = [1, 2, 4, 8, 10]

y\_values = [0, 1, 5, 21, 27]

# Lagrange interpolation derivative

def lagrange\_derivative(x\_vals, y\_vals, x0):

n = len(x\_vals)

result = 0

for i in range(n):

Li\_prime = 0

for j in range(n):

if j != i:

prod = 1

for k in range(n):

if k != i and k != j:

prod \*= (x0 - x\_vals[k])

denom = np.prod([x\_vals[i] - x\_vals[m] for m in range(n) if m != i])

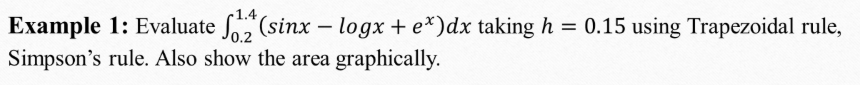
Li\_prime += prod / denom

result += y\_vals[i] \* Li\_prime

return result

fprime\_4 = lagrange\_derivative(x\_values, y\_values, 4)

print(f"Approximate f'(4) = {fprime\_4:.6f}")



#part 1

import numpy as np

def f(x):

return np.sin(x) - np.log(x) + np.exp(x)

a = 0.2

b = 1.4

h = 0.15

x = np.arange(a, b + h, h)

y = f(x)

print("x values:", x)

print("f(x) values:", y)

t=(h/2)\*(y[0]+2\*(y[1]+y[2]+y[3]+y[4]+y[5]+y[6]+y[7])+y[8])

s1=(h/3)\*(y[0]+4\*(y[1]+y[3]+y[5]+y[7])+2\*(y[2]+y[4]+y[6])+y[8])

s2=(3\*h/8)\*(y[0]+3\*(y[1]+y[2]+y[4]+y[5]+y[7])+2\*(y[3]+y[6])+y[8])

print("Trapezoidal value:", t)

print("Simpson's one-third value:", s1)

print("Simpson's three-eight value:", s2)

#part 2

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

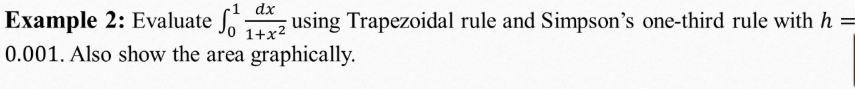
plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

plt.legend()

plt.show()



#part 1

import numpy as np

def f(x):

return 1/(1+(x\*\*2))

a = 0

b = 1

h = 0.001

x = np.arange(a, b + h, h)

y = f(x)

n=len(x)

print("length:", n)

print("x values:", x)

print("f(x) values:", y)

t1=0

for i in range (1,n-1, 1):

t1 += y[i]

trap=(h/2)\*(y[0]+2\*t1+y[n-1])

print("Trapezoidal value:", trap)

#part 2

s1=0

s2=0

for i in range (1,n-1, 2):

s1 += y[i]

for i in range (2,n-2, 2):

s2 += y[i]

sim1=(h/3)\*(y[0]+4\*s1+2\*s2+y[n-1])

print("Simpson's one-third value:", sim1)

#part 3

s3 = 0

s4 = 0

for i in range(1, n - 1):

if i % 3 == 0:

s4 += y[i]

else:

s3 += y[i]

sim3 = (3 \* h / 8) \* (y[0] + 3 \* s3 + 2 \* s4 + y[n - 1])

print("Simpson's three-eighth value:", sim3)

#part 4

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

plt.legend()

plt.show()



#part 1

import numpy as np

def f(x):

return x\*np.exp(x)

a = 0

b = 1

h = 0.001

x = np.arange(a, b + h, h)

y = f(x)

n=len(x)

print("length:", n)

print("x values:", x)

print("f(x) values:", y)

t1=0

for i in range (1,n-1, 1):

t1 += y[i]

trap=(h/2)\*(y[0]+2\*t1+y[n-1])

s1=0

s2=0

for i in range (1,n-1, 2):

s1 += y[i]

for i in range (2,n-2, 2):

s2 += y[i]

sim1=(h/3)\*(y[0]+4\*s1+2\*s2+y[n-1])

s3 = 0

s4 = 0

for i in range(1, n - 1):

if i % 3 == 0:

s4 += y[i]

else:

s3 += y[i]

sim3 = (3 \* h / 8) \* (y[0] + 3 \* s3 + 2 \* s4 + y[n - 1])

print("Trapezoidal value:", trap)

print("Simpson's one-third value:", sim1)

print("Simpson's three-eighth value:", sim3)

#part 2

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

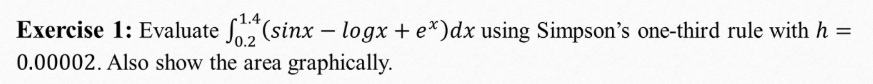
plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

plt.legend()

plt.show()



#part 1

import numpy as np

def f(x):

return np.sin(x) - np.log(x) + np.exp(x)

a = 0.2

b = 1.4

h = 0.00002

x = np.arange(a, b + h, h)

y = f(x)

n=len(x)

print("length:", n)

print("x values:", x)

print("f(x) values:", y)

t1=0

for i in range (1,n-1, 1):

t1 += y[i]

trap=(h/2)\*(y[0]+2\*t1+y[n-1])

s1=0

s2=0

for i in range (1,n-1, 2):

s1 += y[i]

for i in range (2,n-2, 2):

s2 += y[i]

sim1=(h/3)\*(y[0]+4\*s1+2\*s2+y[n-1])

s3 = 0

s4 = 0

for i in range(1, n - 1):

if i % 3 == 0:

s4 += y[i]

else:

s3 += y[i]

sim3 = (3 \* h / 8) \* (y[0] + 3 \* s3 + 2 \* s4 + y[n - 1])

print("Trapezoidal value:", trap)

print("Simpson's one-third value:", sim1)

print("Simpson's three-eighth value:", sim3)

#part 2

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

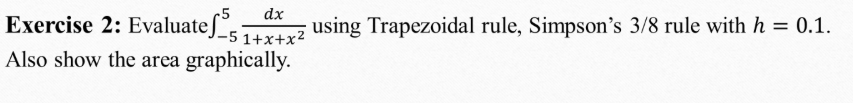
plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

plt.legend()

plt.show()



#part 1

import numpy as np

def f(x):

return 1/(1+x+x\*\*2)

a = -5.0

b = 5.0

h = 0.1

x = np.arange(a, b + h, h)

y = f(x)

n=len(x)

print("length:", n)

print("x values:", x)

print("f(x) values:", y)

t1=0

for i in range (1,n-1, 1):

t1 += y[i]

trap=(h/2)\*(y[0]+2\*t1+y[n-1])

s1=0

s2=0

for i in range (1,n-1, 2):

s1 += y[i]

for i in range (2,n-2, 2):

s2 += y[i]

sim1=(h/3)\*(y[0]+4\*s1+2\*s2+y[n-1])

s3 = 0

s4 = 0

for i in range(1, n - 1):

if i % 3 == 0:

s4 += y[i]

else:

s3 += y[i]

sim3 = (3 \* h / 8) \* (y[0] + 3 \* s3 + 2 \* s4 + y[n - 1])

print("Trapezoidal value:", trap)

print("Simpson's one-third value:", sim1)

print("Simpson's three-eighth value:", sim3)

#part 2

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

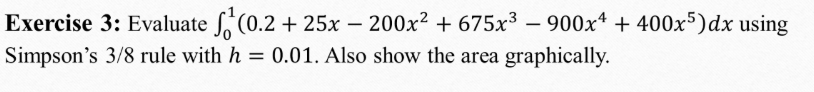
plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

plt.legend()

plt.show()



import numpy as np

import matplotlib.pyplot as plt

# Define the function

def f(x):

return 0.2 + 25\*x - 200\*x\*\*2 + 675\*x\*\*3 - 900\*x\*\*4 + 400\*x\*\*5

# Limits and step size

a = 0.0

b = 1.0

h = 0.01

# Generate x and y values

x = np.arange(a, b + h, h)

y = f(x)

n = len(x)

# Trapezoidal Rule

t1 = 0

for i in range(1, n-1, 1):

t1 += y[i]

trap = (h/2) \* (y[0] + 2\*t1 + y[n-1])

# Simpson's 1/3 Rule

s1 = 0

s2 = 0

for i in range(1, n-1, 2):

s1 += y[i]

for i in range(2, n-2, 2):

s2 += y[i]

sim1 = (h/3) \* (y[0] + 4\*s1 + 2\*s2 + y[n-1])

# Simpson's 3/8 Rule

s3 = 0

s4 = 0

for i in range(1, n-1):

if i % 3 == 0:

s4 += y[i]

else:

s3 += y[i]

sim3 = (3\*h/8) \* (y[0] + 3\*s3 + 2\*s4 + y[n-1])

print("Trapezoidal value:", trap)

print("Simpson's one-third value:", sim1)

print("Simpson's three-eighth value:", sim3)

# Plot the function and shaded area

plt.figure(figsize=(8, 5))

plt.plot(x, y, 'bo-', label='f(x)')

plt.fill\_between(x, y, alpha=0.3, color='skyblue')

plt.title('Area of the function')

plt.xlabel('Value of x')

plt.ylabel('Value of f(x)')

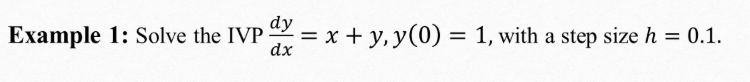
plt.legend()

plt.show()

print("length:", n)

print("x values:", x)

print("f(x) values:", y)



# Given initial values

x = 0

y = 1

h = 0.1

# First derivative

def dy\_dx(x, y):

return x + y

# Second derivative

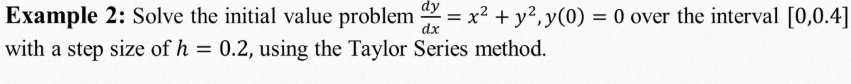
def d2y\_dx2(x, y):

return x + y + 1

# Apply Taylor Series

y\_next = y + h \* dy\_dx(x, y) + (h\*\*2 / 2) \* d2y\_dx2(x, y)

print(f"Approximate value of y(0.1) using Taylor Series: {y\_next:.3f}")



# Given initial values

x = 0

y = 0

h = 0.4

# First derivative

def dy\_dx(x, y):

return x\*\*2 + y\*\*2

# Second derivative

def d2y\_dx2(x, y):

return 2\*x + (2\*y\*(x\*\*2 + y\*\*2))

# Third derivative

def d3y\_dx3(x, y):

return 2 + 4\*x\*y + (6\*y\*(x\*\*2 + y\*\*2))

# Apply Taylor Series

y\_next = y + h \* dy\_dx(x, y) + (h\*\*2 / 2) \* d2y\_dx2(x, y) +(h\*\*3/6) \* d3y\_dx3(x, y)

print(f"Approximate value of y(0.4) using Taylor Series: {y\_next:.3f}")

# Euler's Method

# Given initial values

x = 0

y = 1

h = 0.1

# First derivative

def dy\_dx(x, y):

return x + y

x1=0.1

y1 = y + h \* dy\_dx(x, y)

def dy1\_dx(x1, y1):

return x1 + y1

y2 = y1 + h \* dy1\_dx(x1, y1)

print(f"Approximate value of y(2) using Euler's Method: {y1:.3f}")

#Alternative for n

# Initial values

x = 0

y = 1

h = 0.1

n\_steps = 2

# Store results

x\_vals = [x]

y\_vals = [y]

# Define first derivative

def dy\_dx(x, y):

return x + y

# Loop to calculate y values

for i in range(n\_steps):

y\_new = y + h \* dy\_dx(x, y)

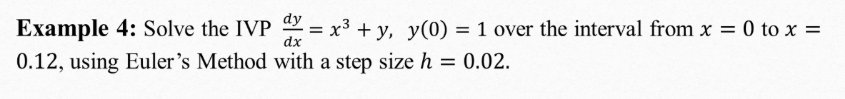
x += h

y = y\_new

x\_vals.append(x)

y\_vals.append(y)

print(f"Approximate value of y(2) using Taylor Series: {y\_vals[-1]:.3f}")



import numpy as np

a=0

b=0.12

x = np.arange(a, b+h, h)

y = 1

h = 0.02

x

n=len(x)

n

x = 0

y = 1

h = 0.02

n\_steps = 7 # To reach x = 0.12

# Store results

x\_vals = [x]

y\_vals = [y]

# Define first derivative

def dy\_dx(x, y):

return x\*\*3 + y

# Loop to calculate y values

for i in range(n\_steps):

y\_new = y + h \* dy\_dx(x, y)

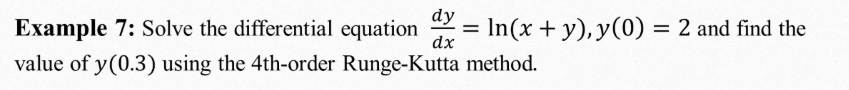
x += h

y = y\_new

x\_vals.append(x)

y\_vals.append(y)

print(f"Approximate value of y(2) using Taylor Series: {y\_vals[-1]:.3f}")



import math as m

# Function definition

def f(x, y):

return m.log(x + y)

# Initial conditions

x0 = 0

y0 = 2

h = 0.15

# Step 1: from x0 to x1

k1 = h \* f(x0, y0)

k2 = h \* f(x0 + h/2, y0 + k1/2)

k3 = h \* f(x0 + h/2, y0 + k2/2)

k4 = h \* f(x0 + h, y0 + k3)

y1 = y0 + (1/6)\*(k1 + 2\*k2 + 2\*k3 + k4)

# Step 2: from x1 to x2

x1 = x0 + h

x2 = x1 + h

k1 = h \* f(x1, y1)

k2 = h \* f(x1 + h/2, y1 + k1/2)

k3 = h \* f(x1 + h/2, y1 + k2/2)

k4 = h \* f(x2, y1 + k3)

y2 = y1 + (1/6)\*(k1 + 2\*k2 + 2\*k3 + k4)

print(f"Approximate value of y(0.3) using RK4: {y2:.6f}")