4 Code contains syntax errors

List of arguements should be seperated by commas.

The code will show syntax error for:

a. print x.

b. Name Error for 'ppolyfit

c. Attribute error

1 1. Import numpy as np

def lagrange_interpolation (x_points, y_points):

def L(k, x):

"""

Compute the kth lagrange basis polynomial at x."""

terms = [(x-x_points[m]) / (x_points[k] - x_points[m] for m in range (len(x_points)) if m != k]

return np.prod(terms, axis=0)

def p(x):

"""

compute the lagrange polynomial at x."""

return sum (y_point[k] * L(k, x) for k in range(len(x_points)))

return P

data_points = [(1,1), (2,4), (3,9), (4,16)]

x_points, y_points = zip(*data_points)

lagrange_poly = lagrange_interpolation (x_points, y_points)

print("lagrange polynomial at x=2.5:", lagrange_poly(2.5))

2. import numpy as np

def newton_interpolation(x, y):

def divided_diff(x, y):

n = len(x)

coef = np.zeros((n,n))

coef[:,0] = y

for j in range (n-1):

for i in range (n-j):

coef[i,j] = (coef[i+1

return coef

def P(xi, coef, x):

result, prod = coef[0,0],1 for

for i in range (1, len(x)):

prod *= (xi - x[i-1])

result += coef[0,i] * prod

return result

coef = divided_diff(x,y)

return lambda xi: P(xi,coef,x)

x, y = [1,2,3,4], [1,4,9,16]

p = newton_interpolation(x, y)

print("newton polynomial at x=2.5:", P(2.5))

J 1. import numby as np

```
def powe-iteration (A, num-simulations: int):
A= np.array (A)
b-k = np.random.rand (A.shape [1])
for _ in range (num-simulations):
b-k1= norm = np.linalg.norm (b-k1)
b-k = b-k1 / b-k1.norm
eigenvalue = np.dot( b-k.T, np.dot (A ,bk)) / np.dot (b-k.T, b-k)
return eigenvalue, b-k
A=[[4,1,1], [1,3,-1], [1,-1, 2]]
dominant-eigenvalue, dominant-eigenvector = power-Iteration (A,1000)
print ("Dominant Eigenvalue :", dominant-eigenvalue
print ("Dominant Eigenvalue:", dominant-eigenvector.
```

2. def qr-algorithm (A, num-iteration: int):

```
A = np.array (A)
n = A.shape [0]
Q= np.eye(n)
for _ in range (num-iterations):
Q-1, R 1 = np.linalg.qr [A]
A= np.dot (Q, Q.i)
Eigenvalue = np.diag(A)
eigenvalue= Q
return eigenvalue, eigenvectors.
A = [[4,1,1], [1,3,-1], [1,-1,2]]
eigenvalues, eigenvectors = qr-algorithm
print ("eigenvalues:", eigenvalue)
print ("Eigenvectors:", eigenvectors)
```

3. Comparison.

Both methods effectively identifies the dominant eigenvalue, and the QR algorithm additionally provided all eigenvalues and their corresponding eigenvectors, demonstrating a comprehensive solution compared to the power iteration method.

```
k import numpy as np
def gradient_descent ( learning_rate =0.1, initial_guess =(0,0), num_iterations =1000):
x, y = initial_guess
for _ in range (num_iterations):
grad_x = 2*x - y+1
grad_y = 2*y - x - 1
x_ = learing_rate * grad_x
y_ = learnig_rate * grad_y
return x, y
min_x, min_y = gradient_descent ()
print (f"minimum found at: x = {min_x} y = {min_y}")
```

CAT II: NUMERICAL ANALYSIS

INTE/MG/2928/09/22

EMMANUEL KIPKEMOI BIEGON

a $x^2 - x - 2 = 0$

Initial guesses [1,3]

$f(x) = x^2 - x - 2$

Points = $a = 1$

$b = 3$

$f(a) = 1^2 - 1 - 2 = -2$

$f(b) = 3^2 - 1 - 2 = 4$

formula = $x = b - \frac{f(b) \cdot (a-b)}{f(a) - f(b)}$

Iteration 1

$x = 3 - \frac{4 \cdot (1-3)}{-2-4} = 0.6667$

$= \frac{3 - 4 + 12}{-6}$

$f(x) = -2.222$

$= 0$

$a = 1$

$b = 0.6667$.

$fx = f(0.6667) = 0.6667^2 - 0.667 - 2$

Iteration 2

$x = 0.6667 - \frac{-2.222 \cdot (1 - 0.6667)}{-2 - (-2.222)}$

$= -3.33$

$fx = 4^2 - 4 - 2 = 8$

$a = 4$

$b = 0.6667$

Iteration 3

$x = 0.6667 - \frac{8(4 - 0.6667)}{-2 - 8}$

$= 3.3334$

$f(x) = 3.3334^2 - 3.3334 - 2 = 5.5556$

$a = 4$

$b = 3.3334$

b 1. Differentiation - NumPy

```
Import numby as np
Import matplotlip.pyplot as plt
def f(x):
return x**2 - 3*x + 2
def numerical_derivative(f, x, h=1e-5):
return (f(x+h) - f(x-h)) / (2*h)
x = np.linespace(0, 3, 100)
y = f(x)
dy_dx = numerical_derivatives(f, x)
Plt.plot(x, y, label = ("f(x)"))
plt.plot(x, dy_dx, label ("f'(x)"))
plt.legend()
plt.xlabel("x)
plt.ylabel("y")
Plt.title("Numerical Differentiation")
plt.show()
```