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()
```

2. Numerical Integration - SciPy

```
from scipy.integrate import quad
def integrand(x):
return x**2 - 3*x + 2
integral, error = quad(integrand, 0, 3)
print(f'integral results: {integral}')
print(f'Error estimate: {Error}')
```

3 Curve Fitting - SciPy

```
from scipy.optimize import curve_fit
def model_func(x, a, b, c):
return a*x**2 + b*x + c
np.random.seed(0)
x_data = np.linspace(0, 10, 100)
y_data = model_func(x_data, 2, -3, 1) + np.random.normal(0, 1, size=x_data.size)
params, covariance = curve_fit(model_func, x_data, y_data)
Plt.scatter(x_data, y_data, label='Data')
plt.plot(x_data, model_func(x_data, *params), label='fitted curve', color='blue')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Curve fitting')
plt.show()
print(f'fitted parameters: {params}')
```

4. Linear Regression - Scikit-learn

```
from sklearn.linear_model import LinearRegression.
X_data = np.linspace(0, 10, 100).reshape(-1,1)
y_data = 3*X_data.flatten() + 7 + np.random.normal(0,1, size = X_data.shape[0])
model = LinearRegression()
model.fit(X_data, y_data)
y_pred = model.predict(X_data)
plt.scatter(X_data, y_data, label = "Data")
plt.plot(X_data, y_pred, label = 'Linear Regression', color blue
plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression')
plt.show()
```

c) Given points:

$(x_1, y_1) = (2.00, 7.2)$

$(x_2, y_2) = (4.25, 7.1)$

2. Interpolation at $x = 40$

$$y = y_1 + \frac{(x - x_1)(y_2 - y_1)}{x_2 - x_1}$$

$$= 7.2 + \frac{(4.0 - 2).(7.1 - 7.2)}{4.25 - 2.0}$$

$$= 7.1111$$

value of y at $x = 40 = 7.1111$.

d) Iteration 1 $f(x) = x^3 - 0.165x^2 + 3.993 \times 10^{-4}$

$x_0 = 0.5$ Derivative:

$$f'(x) = 3x^2 - 0.33x$$

Newton's method $= x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

Iteration 1

let's assume $x_0 = 5$

$f(0.5) = 0.5^3 - 0.165 \times 0.5^2 + 3.993 \times 10^{-4} = 0.0841493.$

$f'(0.5) = 3 \times 0.5^3 - 0.33 \times 0.5 = 0.585.$

$$x_1 = 0.5 - \frac{0.0841493}{0.585} = 0.5 - 0.143 = 0.357$$

Absolute relative error $= \left|\frac{0.357 - 0.5}{0.357}\right| \times 100\%$

$= 40.15\%$

Iteration 2

$x_1 = 0.357$

$f(0.357) = 0.357^3 - 0.165 \times 0.357^2 + 3.993 \times 10^{-4} = 0.03527.$

$f'(0357) = 0357^3 - 0.33 \times 0.357 = 0.264$

$$x_2 = 0.357 - \frac{0.03527}{0.264} = 0.357$$

Absolute relative error $= \left|\frac{0.224 - 0357}{0.224}\right| \times 100$

$= 59.38\%$

Iteration 3

$x_2 = 0.224$

$f(0.224) = 0.224^3 - 0.165 \times 0.224^2 + 3.993 \times 10^{-4} = 0.0090.$

$f'(0.224) = 3 \times 0.224^2 - 0.33 \cdot 0224 = -0.059$

$$x_3 = 0.224 - \frac{0.0090}{-0059} = 0.377$$

Absolute relative error $= \left|\frac{0.377 - 0.224}{0.377}\right| \times 100\%$

$= 40.68\%$

2 Import numpy as np

```
import matplotlib.pyplot as plt
def compute_fft(signal, sampling_rate):
N = len(signal)
fft_values = np.fft.fft(signal)
fft_frequencies = np.fft.fftfreq(N, d=1/sampling_rate)
positive_freqs = fft_frequencies[:N//2]
positive_fft = np.abs(fft_values[:N//2])
return positive_freqs, positive_fft
f1 = 50
f2 = 120
sampling_rate = 1000
duration = 1
t = np.linspace(0, duration, int(sampling_rate * duration), endpoint=False)
signal = np.sin(2 * np.pi * f1 * t) + np.sin(2 * np.pi * f2 * t)
frequencies, fft_magnitude = compute_fft(signal, sampling_rate)
plt.figure(figsize=(10,6))
plt.plot(frequencies, fft_magnitude)
plt.title('FFT of the signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plot.xlim(0,200)
plt.grid()
plt.show()
```