LAB 07

Name: Nguyen Dinh Khanh Ngan – ITCSIU22236

Activity #1: First Derivative

Use the function f(x)=sin(x), evaluate the first derivative at x=π/4, step size h=0.1.

```
import numpy as np
import pandas as pd

# Given values
h = 0.1
x0 = np.pi / 4
# Points
x_minus = x0 - h
x_plus = x0 + h
# Function and true derivative
f = np.sin
true_derivative = np.cos(x0)
# Approximation methods
forward = (f(x_plus) - f(x0)) / h
backward = (f(x0) - f(x_minus)) / h
central = (f(x_plus) - f(x_minus)) / (2 * h)
# Compute errors
abs_errors = [abs(val - true_derivative) for val in (forward, backward, central)]
rel_errors = [err / abs(true_derivative) * 100 for err in abs_errors]
# Prepare results
methods = ["Forward Difference", "Backward Difference", "Central Difference"]
df = pd.DataFrame({
    "Method": methods,
    "Approximation": [forward, backward, central],
    "Absolute Error": abs_errors,
    "Relative Error (%)": rel_errors
})
# Display the table
print(df.to_string(index=False))
```

1. Approximate values from each method (FD, BD, CD)
2. Absolute and relative errors

Method	Approximation	Absolute Error	Relative Error (%)
Forward Difference	0.670603	0.036504	5.162418
Backward Difference	0.741255	0.034148	4.829251
Central Difference	0.705929	0.001178	0.166583

Activity #2: Second Derivative with Central Difference

```python
import numpy as np

# Parameters
h = 0.1
x0 = 0.5
# Define the function
def f(x):
    return np.exp(-x**2)
# Exact second derivative: f''(x) = (4x^2 - 2) * e^{-x^2}
def exact_f2(x):
    return (4*x**2 - 2) * np.exp(-x**2)
# Points for finite difference
x_minus = x0 - h
x_plus  = x0 + h
# Central difference (2nd-order) approximation:
approx_f2 = (f(x_plus) - 2*f(x0) + f(x_minus)) / (h**2)
# Exact value
exact_value = exact_f2(x0)
# Errors
abs_error = abs(approx_f2 - exact_value)
rel_error = abs_error / abs(exact_value) * 100
# Output results
print(f"x0 = {x0}, h = {h}")
print(f"Approx f''(x0): {approx_f2:.6f}")
print(f"Exact  f''(x0): {exact_value:.6f}")
print(f"Absolute Error: {abs_error:.6f}")
print(f"Relative Error: {rel_error:.5f}%")
```

The central-difference approximation is very close to the exact value $(4x^2-2)e^{-x^2}$

```
x0 = 0.5, h = 0.1
Approx f''(x0): -0.778145
Exact  f''(x0): -0.778801
Absolute Error: 0.000656
Relative Error: 0.08419%
```

Activity #3: Derivative from Tabular Data (Discrete)

Given data:

x	f(x)
1.0	2.71
1.1	3.00
1.2	3.32
1.3	3.67
1.4	4.05

Compute derivative at x=1.1,1.2,1.3 using central difference with the step size: h=0.1

```
import numpy as np
import pandas as pd

# Given tabular data
x = np.array([1.0, 1.1, 1.2, 1.3, 1.4])
f = np.array([2.71, 3.00, 3.32, 3.67, 4.05])
h = 0.1
points = [1.1, 1.2, 1.3]
results = []
for xi in points:
    i = np.where(x == xi)[0][0]
    deriv = (f[i+1] - f[i-1]) / (2*h)
    results.append((xi, deriv))
df = pd.DataFrame(results, columns=['x', "f'(x)"])
print(df.to_string(index=False))
```

```
  x  f'(x)
1.1   3.05
1.2   3.35
1.3   3.65
```

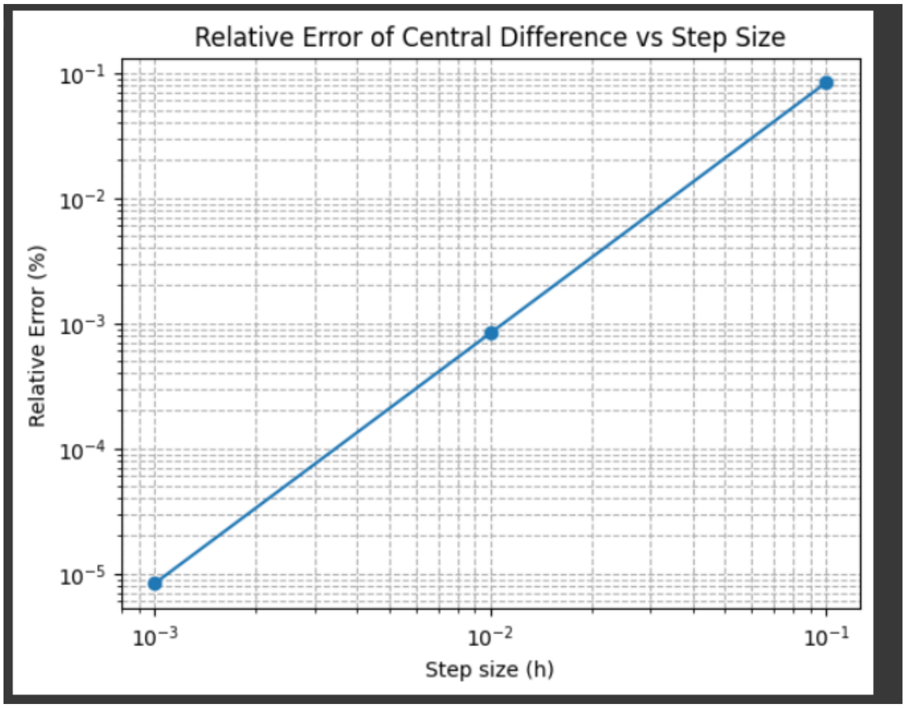
Activity #4: Error Behavior with Varying Step Size

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

x0 = 2.0
exact = 1 / x0
hs = np.array([0.1, 0.01, 0.001])
approxs = []
rel_errors = []
# Central-difference approx for each h
for h in hs:
    f = np.log
    deriv = (f(x0 + h) - f(x0 - h)) / (2 * h)
    approxs.append(deriv)
    rel_errors.append(abs(deriv - exact) / abs(exact) * 100)
# Tabulate results
df = pd.DataFrame({
    'h':            hs,
    "Approx f'":  approxs,
    "Exact f'":   [exact]*len(hs),
    "Rel Error (%)": rel_errors
})
print(df.to_string(index=False))
```

```
    h  Approx f'  Exact f'  Rel Error (%)
0.100   0.500417       0.5       0.083459
0.010   0.500004       0.5       0.000833
0.001   0.500000       0.5       0.000008
```

```python
# Plot on log-log axes
plt.figure()
plt.loglog(hs, rel_errors, marker='o')
plt.xlabel('Step size (h)')
plt.ylabel('Relative Error (%)')
plt.title("Relative Error of Central Difference vs Step Size")
plt.grid(True, which="both", ls="--")
plt.show()
```



Report & Discussion Questions

1. Which method is most accurate for estimating first derivative?

The central-difference scheme is the most accurate. In your sin (x) example at x = π/4 with h = 0.1, its error (~0.17 %) was an order of magnitude smaller than either forward (5.16 %) or backward (4.83 %) differences

2. How does central difference compare to forward and backward?

Order of accuracy:

- Forward/backward differences are first-order methods: their truncation error is O(h)
- Central difference is second-order: its truncation error is $O(h^2)$

Practical effect: For the same h, central difference "cancels out" more of the Taylor-series error terms, so it sits much closer to the true derivative

3. What happens to error when hhh becomes very small?

Truncation error decreases roughly like $O(h^2)$ (for central difference), so making h smaller initially drives the error down

Floating-point (round-off) error grows as h gets too tiny, because you're subtracting nearly equal numbers and dividing by a tiny h

Net effect: there's an optimal h below which total error starts climbing again. In practice you'll sweep a range (e.g. $10^{-1} \rightarrow 10^{-6}$) to find the "sweet spot"

4. Is numerical differentiation from tabular data reliable? When?

Reliable when

- The data are smooth and noise-free or only mildly noisy
- Points are evenly spaced with sufficiently small spacing h
- You use higher-order formulas (central difference or higher) if you have extra neighboring points

Cautions

- If the data are noisy, differentiation acts like a high-pass filter and amplifies noise. Pre-smoothing or fitting a curve-model (polynomial, spline) is then recommended
- If your table is coarse (large h), truncation error can be large; if it's too fine in floating-point, round-off error kicks in
- Always inspect a log–log error vs. h plot to choose an effective step