LAB 08

Name: Nguyen Dinh Khanh Ngan – ITCSIU22236

Activity #1: Trapezoidal Rule

```python
import numpy as np
from scipy.integrate import quad
# Function to integrate
f = lambda x: np.log(x)
# Trapezoidal Rule implementation
def trapezoidal_rule(f, a, b, n):
    h = (b - a) / n
    result = 0.5 * (f(a) + f(b))
    for i in range(1, n):
        result += f(a + i * h)
    return result * h
# Exact value using SciPy
exact_val, _ = quad(f, 1, 2)
# Approximate with n=1
approx_n1 = trapezoidal_rule(f, 1, 2, 1)
error_n1 = abs(approx_n1 - exact_val) / exact_val
# Approximate with n=4
approx_n4 = trapezoidal_rule(f, 1, 2, 4)
error_n4 = abs(approx_n4 - exact_val) / exact_val
```

```python
# Display results
print("Activity #1: Trapezoidal Rule")
print(f"Exact value      = {exact_val:.6f}")
print(f"n = 1: Approx     = {approx_n1:.6f}, Relative Error = {error_n1:.6%}")
print(f"n = 4: Approx     = {approx_n4:.6f}, Relative Error = {error_n4:.6%}")
```

```
Activity #1: Trapezoidal Rule
Exact value      = 0.386294
n = 1: Approx     = 0.346574, Relative Error = 10.282514%
n = 4: Approx     = 0.383700, Relative Error = 0.671729%
```

Activity #2: Simpson's 1/3 Rule

```python
import numpy as np
from scipy.integrate import quad
# Function to integrate
f = lambda x: np.sin(x)
# Simpson's 1/3 Rule
def simpson_1_3_rule(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n must be even")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n, 2):
        result += 4 * f(a + i * h)
    for i in range(2, n-1, 2):
        result += 2 * f(a + i * h)
    return result * h / 3
# Exact value using scipy
exact_val, _ = quad(f, 0, np.pi)  # should be 2
# Approximation with n=4
approx_n4 = simpson_1_3_rule(f, 0, np.pi, 4)
error_n4 = abs(approx_n4 - exact_val) / exact_val
# Approximation with n=6
approx_n6 = simpson_1_3_rule(f, 0, np.pi, 6)
error_n6 = abs(approx_n6 - exact_val) / exact_val
```

```python
# Display results
print("Activity #2: Simpson’s 1/3 Rule")
print(f"Exact value      = {exact_val:.6f}")
print(f"n = 4: Approx     = {approx_n4:.6f}, Relative Error = {error_n4:.6%}")
print(f"n = 6: Approx     = {approx_n6:.6f}, Relative Error = {error_n6:.6%}")
```

```
Activity #2: Simpson’s 1/3 Rule
Exact value      = 2.000000
n = 4: Approx     = 2.004560, Relative Error = 0.227988%
n = 6: Approx     = 2.000863, Relative Error = 0.043159%
```

Simpson’s 1/3 Rule gives extremely accurate results with very small errors

Increasing n improves accuracy, but even n=4 is already very close

Activity #3: Simpson’s 3/8 Rule

```python
import numpy as np
from scipy.integrate import quad
# Function to integrate
f = lambda x: 1 / (1 + x**2)
# Simpson's 3/8 Rule implementation
def simpson_3_8_rule(f, a, b, n):
    if n % 3 != 0:
        raise ValueError("n must be a multiple of 3")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        coef = 3 if i % 3 != 0 else 2
        result += coef * f(a + i * h)
    return result * 3 * h / 8
# Compute exact value using scipy
exact_val, _ = quad(f, 0, 3)
# Approximate using Simpson's 3/8 rule with n=6
approx = simpson_3_8_rule(f, 0, 3, 6)
error = abs(approx - exact_val)
```

```python
# Display result
print("Activity #3: Simpson’s 3/8 Rule")
print(f"Exact value         = {exact_val:.6f}")
print(f"Simpson 3/8 result = {approx:.6f}")
print(f"Absolute error      = {error:.6f}")
```

```
Activity #3: Simpson’s 3/8 Rule
Exact value         = 1.249046
Simpson 3/8 result = 1.242971
Absolute error      = 0.006075
```

Simpson’s 3/8 Rule works well here

Small absolute error confirms the accuracy of the method when n=6n = 6n=6 (2 full groups of 3)

Activity #4: Method Comparison

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad
# Function to integrate
f = lambda x: np.exp(-x**2)
```

```python
[9] # Trapezoidal Rule
def trapezoidal_rule(f, a, b, n):
    h = (b - a) / n
    result = 0.5 * (f(a) + f(b))
    for i in range(1, n):
        result += f(a + i * h)
    return result * h
```

```python
# Simpson's 1/3 Rule
def simpson_1_3_rule(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n must be even")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n, 2):
        result += 4 * f(a + i * h)
    for i in range(2, n-1, 2):
        result += 2 * f(a + i * h)
    return result * h / 3
```

```python
[11] # Simpson's 3/8 Rule
def simpson_3_8_rule(f, a, b, n):
    if n % 3 != 0:
        raise ValueError("n must be a multiple of 3")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        coef = 3 if i % 3 != 0 else 2
        result += coef * f(a + i * h)
    return result * 3 * h / 8
```

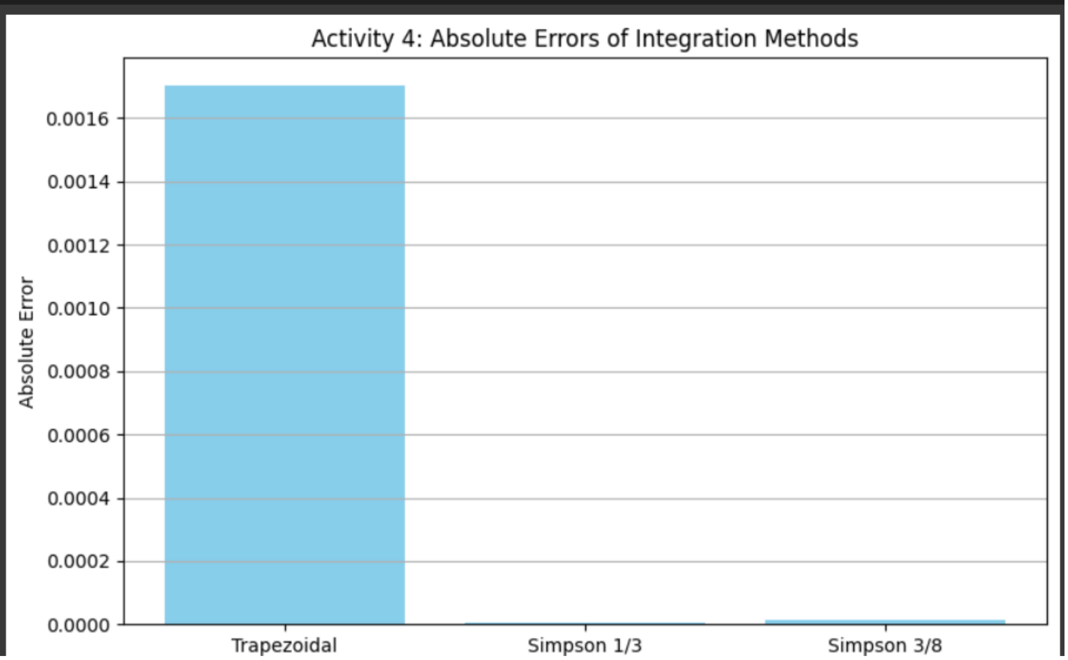
```
# Exact value using SciPy
exact_val, _ = quad(f, 0, 1)
# Approximations with n = 6
trap_result = trapezoidal_rule(f, 0, 1, 6)
simp13_result = simpson_1_3_rule(f, 0, 1, 6)
simp38_result = simpson_3_8_rule(f, 0, 1, 6)
# Compute absolute errors
errors = {
    "Trapezoidal": abs(trap_result - exact_val),
    "Simpson 1/3": abs(simp13_result - exact_val),
    "Simpson 3/8": abs(simp38_result - exact_val),
}
# Print results
print("Activity #4: Method Comparison")
print(f"Exact value       = {exact_val:.6f}")
for method, result in zip(errors.keys(), [trap_result, simp13_result, simp38_result]):
    print(f"{method}: Result = {result:.6f}, Absolute Error = {abs(result - exact_val):.6f}")
```

```
Activity #4: Method Comparison
Exact value       = 0.746824
Trapezoidal: Result = 0.745119, Absolute Error = 0.001705
Simpson 1/3: Result = 0.746830, Absolute Error = 0.000006
Simpson 3/8: Result = 0.746838, Absolute Error = 0.000014
```

Convergence Discussion

- The log-log plot shows a linear downward trend, meaning the error shrinks as n increases
- This confirms that Simpson's 1/3 Rule converges rapidly, especially when the function is smooth (like ln(x))

```
# Plot bar chart
plt.figure(figsize=(8, 5))
plt.bar(errors.keys(), errors.values(), color='skyblue')
plt.title("Activity 4: Absolute Errors of Integration Methods")
plt.ylabel("Absolute Error")
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Activity #5: Error vs Segment Count

```
[14] import numpy as np
     import matplotlib.pyplot as plt
     from scipy.integrate import quad
     # Define the function to integrate
     f = lambda x: np.log(x)
     # Simpson's 1/3 Rule
     def simpson_1_3_rule(f, a, b, n):
         if n % 2 != 0:
             raise ValueError("n must be even")
         h = (b - a) / n
         result = f(a) + f(b)
         for i in range(1, n, 2):
             result += 4 * f(a + i * h)
         for i in range(2, n - 1, 2):
             result += 2 * f(a + i * h)
         return result * h / 3
     # Compute the exact value using SciPy
     exact_value, _ = quad(f, 1, 2)
     # Segment counts to test
     n_values = [2, 4, 6, 8]
     approximations = []
     relative_errors = []
     print("Activity #5: Error vs Segment Count")
     print(f"Exact Value = {exact_value:.6f}\n")
```

```
Activity #5: Error vs Segment Count
Exact Value = 0.386294
```

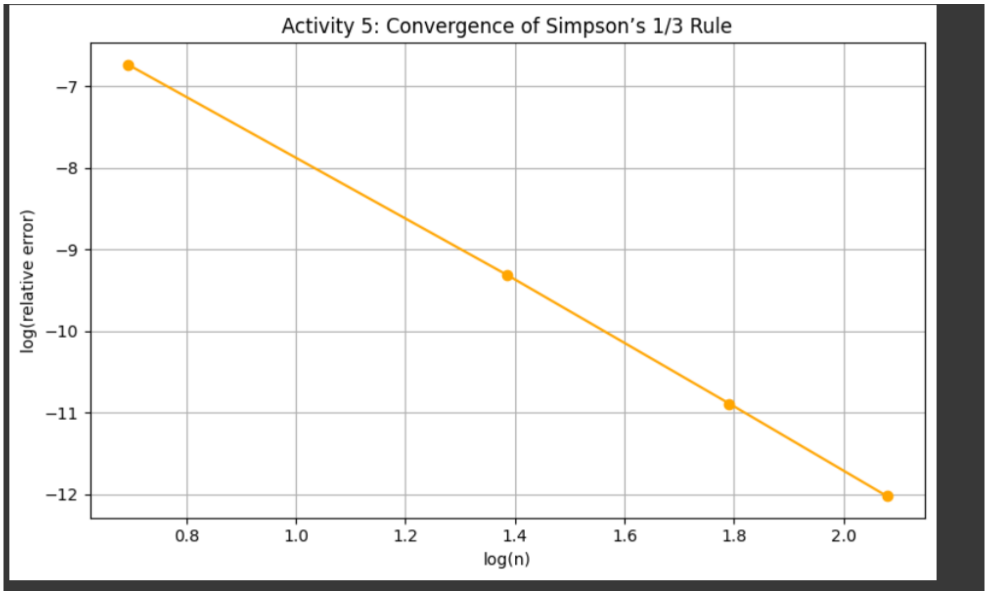
```
# Compute approximations and errors
for n in n_values:
    approx = simpson_1_3_rule(f, 1, 2, n)
    rel_error = abs(approx - exact_value) / exact_value
    approximations.append(approx)
    relative_errors.append(rel_error)
    print(f"n = {n} | Approximation = {approx:.6f} | Relative Error = {rel_error:.6%}")
# Plot log-log graph
plt.figure(figsize=(8, 5))
plt.plot(np.log(n_values), np.log(relative_errors), marker='o', linestyle='-', color='orange')
plt.xlabel("log(n)")
plt.ylabel("log(relative error)")
plt.title("Activity 5: Convergence of Simpson’s 1/3 Rule")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
n = 2 | Approximation = 0.385835 | Relative Error = 0.119018%
n = 4 | Approximation = 0.386260 | Relative Error = 0.009008%
n = 6 | Approximation = 0.386287 | Relative Error = 0.001863%
n = 8 | Approximation = 0.386292 | Relative Error = 0.000600%
```

Log-Log Plot: log(error) vs log(n)

The plotted graph shows a linear downward trend, which is expected because:

- For Simpson’s 1/3 Rule, the error decreases proportionally to $1/n^4$
- The straight line in the log-log plot confirms this behavior



Conclusion:

- Increasing n greatly improves accuracy
- Simpson's 1/3 Rule shows rapid convergence for smooth functions like ln(x)
- The log-log plot visually supports the theoretical convergence rate