Name: Phan Tran Thanh Huy

ID: ITCSIU22056

Lab 8

Activity 1: Trapezoidal Rule

Code:

```python
import numpy as np
from math import log
from scipy.integrate import quad

def f(x):
    return np.log(x)

def trapezoidal_rule(f, a, b, n):
    h = (b - a) / n
    total = 0.5 * (f(a) + f(b))
    for i in range(1, n):
        total += f(a + i * h)
    return total * h

a = 1
b = 2

T1 = trapezoidal_rule(f, a, b, n=1)
T4 = trapezoidal_rule(f, a, b, n=4)

exact_value, _ = quad(f, a, b)

def relative_error(approx, exact):
    return abs((approx - exact) / exact) * 100

print("Trapezoidal Rule with n=1:", T1)
print("Trapezoidal Rule with n=4:", T4)
print("Exact Value:", exact_value)
print("Relative Error (n=1):", relative_error(T1, exact_value), "%")
print("Relative Error (n=4):", relative_error(T4, exact_value), "%")

```

Result:

```
PS E:\Homework\TMC\Lab8> & C:/Python312/python.exe e:/Homework/TMC/Lab8/p1.py
Trapezoidal Rule with n=1: 0.34657359027997264
Trapezoidal Rule with n=4: 0.38369950940944236
Exact Value: 0.38629436111989063
Relative Error (n=1): 10.28251376094776 %
Relative Error (n=4): 0.6717291194532671 %
```

Activity 2: Simpson's 1/3 Rule

Code:

```python
import numpy as np
from math import sin, pi
from scipy.integrate import quad

def f(x):
    return np.sin(x)

def simpson_rule(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n must be even for Simpson's 1/3 Rule")

    h = (b - a) / n
    x = [a + i * h for i in range(n + 1)]
    y = [f(xi) for xi in x]

    result = y[0] + y[-1]
    for i in range(1, n):
        if i % 2 == 0:
            result += 2 * y[i]
        else:
            result += 4 * y[i]

    return result * h / 3

exact_value, _ = quad(f, 0, pi)

S4 = simpson_rule(f, 0, pi, 4)
S6 = simpson_rule(f, 0, pi, 6)

def relative_error(approx, exact):
    return abs((approx - exact) / exact) * 100

print("Simpson's Rule with n=4:", S4)
print("Simpson's Rule with n=6:", S6)
print("Exact Value:", exact_value)
print("Relative Error (n=4):", relative_error(S4, exact_value), "%")
print("Relative Error (n=6):", relative_error(S6, exact_value), "%")
```

Result:

```
PS E:\Homework\TMC\Lab8> & C:/Python312/python.exe e:/Homework/TMC/Lab8/p2.py
Simpson's Rule with n=4: 2.0045597549844207
Simpson's Rule with n=6: 2.0008631896735363
Exact Value: 2.0
Relative Error (n=4): 0.22798774922103693 %
Relative Error (n=6): 0.04315948367681344 %
```

Activity 3: Simpson's 3/8 Rule

Code:

```python
import numpy as np
from math import pi
from scipy.integrate import quad

def f(x):
    return 1 / (1 + x**2)

def simpson_38(f, a, b, n):
    if n % 3 != 0:
        raise ValueError("n must be a multiple of 3 for Simpson's 3/8 Rule")

    h = (b - a) / n
    x = [a + i * h for i in range(n + 1)]
    y = [f(xi) for xi in x]

    result = y[0] + y[-1]

    for i in range(1, n):
        if i % 3 == 0:
            result += 2 * y[i]
        else:
            result += 3 * y[i]

    return (3 * h / 8) * result

a = 0
b = 3
n = 6

S38 = simpson_38(f, a, b, n)

exact_value = np.arctan(b) - np.arctan(a)

def relative_error(approx, exact):
    return abs((approx - exact) / exact) * 100

print("Simpson's 3/8 Rule with n=6:", S38)
print("Exact Value (arctan):", exact_value)
print("Relative Error:", relative_error(S38, exact_value), "%")
```

Result:

```
PS E:\Homework\TMC\Lab8> & C:/Python312/python.exe e:/Homework/TMC/Lab8/p3.py
Simpson's 3/8 Rule with n=6: 1.2429708222811668
Exact Value (arctan): 1.2490457723982544
Relative Error: 0.4863672934437991 %
```

Activity 4: Method Comparison

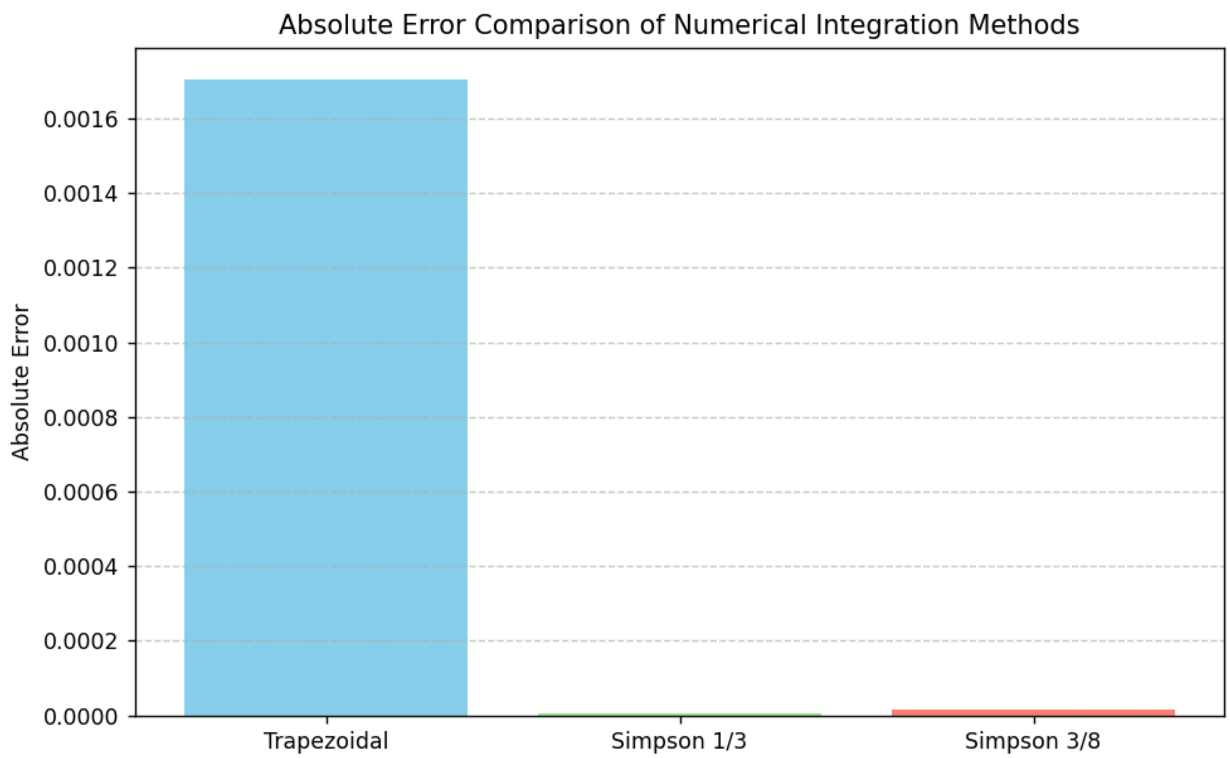
Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from math import exp
from scipy.integrate import quad

def f(x):
    return np.exp(-x**2)

# Trapezoidal Rule
def trapezoidal(f, a, b, n):
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        result += 2 * f(a + i * h)
    return (h / 2) * result

# Simpson's 1/3 Rule
def simpson_13(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n must be even for Simpson's 1/3 Rule")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        result += 4 * f(a + i * h) if i % 2 != 0 else 2 * f(a + i * h)
    return (h / 3) * result

# Simpson's 3/8 Rule
def simpson_38(f, a, b, n):
    if n % 3 != 0:
        raise ValueError("n must be a multiple of 3 for Simpson's 3/8 Rule")
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        if i % 3 == 0:
            result += 2 * f(a + i * h)
        else:
            result += 3 * f(a + i * h)
    return (3 * h / 8) * result

a, b = 0, 1
n = 6
exact_value, _ = quad(f, a, b)

trap_val = trapezoidal(f, a, b, n)
simp13_val = simpson_13(f, a, b, n)
simp38_val = simpson_38(f, a, b, n)

errors = {
    "Trapezoidal": abs(trap_val - exact_value),
    "Simpson 1/3": abs(simp13_val - exact_value),
    "Simpson 3/8": abs(simp38_val - exact_value)
}

print("{:<15} {:<20} {:<20}".format("Method", "Approximation", "Absolute Error"))
print("-" * 55)
print("{:<15} {:<20.10f} {:<20.10f}".format("Trapezoidal", trap_val, errors["Trapezoidal"]))
print("{:<15} {:<20.10f} {:<20.10f}".format("Simpson 1/3", simp13_val, errors["Simpson 1/3"]))
print("{:<15} {:<20.10f} {:<20.10f}".format("Simpson 3/8", simp38_val, errors["Simpson 3/8"]))

plt.figure(figsize=(8, 5))
plt.bar(errors.keys(), errors.values(), color=['skyblue', 'lightgreen', 'salmon'])
plt.title('Absolute Error Comparison of Numerical Integration Methods')
plt.ylabel('Absolute Error')
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Result:

```
PS E:\Homework\TMC\Lab8> & C:/Python312/python.exe e:/Homework/TMC/Lab8/p4.py
Method          Approximation       Absolute Error
-------------------------------------------------------
Trapezoidal     0.7451194124        0.0017047204
Simpson 1/3     0.7468303915        0.0000062587
Simpson 3/8     0.7468380575        0.0000139247
```

Plot:



Activity 5: Error vs Segment Count

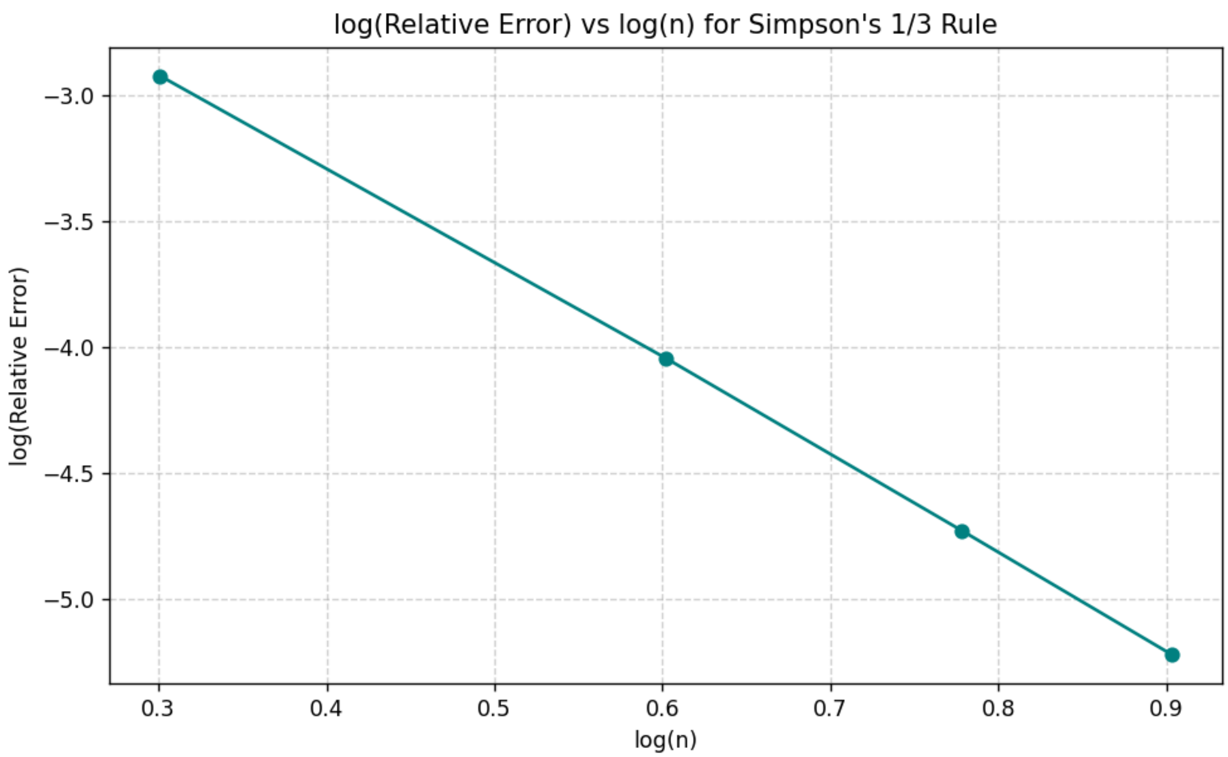
Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from math import log
from scipy.integrate import quad

def f(x):
    return np.log(x)

def simpson_13(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n must be even for Simpson's 1/3 Rule")
    h = (b - a) / n
    x = np.linspace(a, b, n+1)
    y = f(x)
    result = y[0] + y[-1] + 4 * sum(y[1:-1:2]) + 2 * sum(y[2:-2:2])
    return (h / 3) * result

a, b = 1, 2
exact, _ = quad(f, a, b)

ns = [2, 4, 6, 8]
approximations = []
relative_errors = []

for n in ns:
    approx = simpson_13(f, a, b, n)
    rel_error = abs((approx - exact) / exact)
    approximations.append(approx)
    relative_errors.append(rel_error)

print(f"{'n':<5}{'Approximation':<20}{'Relative Error':<20}")
print("-" * 45)
for n, approx, err in zip(ns, approximations, relative_errors):
    print(f"{n:<5}{approx:<20.10f}{err:<20.10f}")

log_n = np.log10(ns)
log_error = np.log10(relative_errors)

plt.figure(figsize=(8, 5))
plt.plot(log_n, log_error, marker='o', linestyle='-', color='teal')
plt.title("log(Relative Error) vs log(n) for Simpson's 1/3 Rule")
plt.xlabel("log(n)")
plt.ylabel("log(Relative Error)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

Result:

```
PS E:\Homework\TMC\Lab8> & C:/Python312/python.exe e:/Homework/TMC/Lab8/p5.py
n    Approximation       Relative Error
---------------------------------------------
2    0.3858346022        0.0011901778
4    0.3862595628        0.0000900824
6    0.3862871633        0.0000186330
8    0.3862920435        0.0000059997
```

Plot:



Discuss convergence behavior

Simpson’s 1/3 Rule demonstrates rapid convergence when applied to the integral above. As the number of segments n increases from 2 to 8, the relative error decreases dramatically, confirming the method's fourth-order accuracy. This is further supported by the linear trend in the log-log plot of error versus segment count, with a slope close to –4. The results show that Simpson’s 1/3

Rule achieves high accuracy with relatively few segments for smooth functions, making it an efficient and reliable method for numerical integration.