Homework 0

소유자	Hyejun Park
생성 일시	@2025년 10월 20일 오후 10:05

Exercise 0.2

$$f(x) = 2x^2 + 3x^3 + 8x^5$$

- As $x \to 0 : O(x^2)$
- As $x \to \infty : O(x^5)$
- $f(x) = 2x^2 + O(x^3)$

Exercise 0.3

(a) Compute the exact integral

$$\int_0^{\frac{\pi}{2}} (2 + cosx)dx$$

$$= 2x + sinx|_0^{\frac{\pi}{2}}$$

$$= (2 * \pi/2 + sin\pi/2) - 0$$

$$= \pi + 1$$

(b) Implement `convergence_template.py`

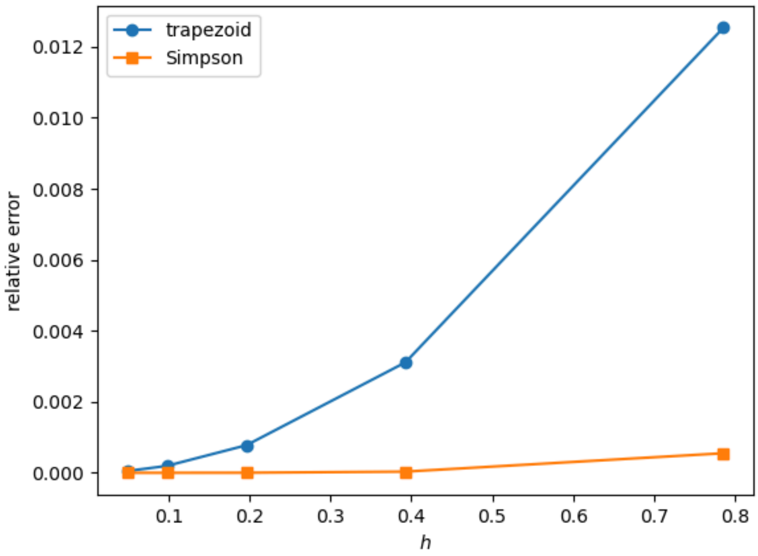
```
"""
Numerical Mathematics for Engineering II WS 25/26
Homework 00 Exercise 0.3
Revision on convergence curves and tables
"""

# Import necessary libraries
```

```
import numpy as np
import scipy.integrate # library to calculate integrate
import matplotlib.pyplot as plt

# Methods for Integrating Functions given fixed samples.

#    trapezoid            -- Use trapezoidal rule to compute integral.
#    cumulative_trapezoid -- Use trapezoidal rule to cumulatively compute in
tegral.
#    simpson              -- Use Simpson's rule to compute integral from sample
s.

def f(x):
    return 2 + np.cos(x)

# interval
a = 0
b = np.pi/2

def approximate_integral(n,rule):
    h = (b-a)/n # step size
    x = np.linspace(a, b, n+1) # range [a,b] => chop with n sections

    if(rule == "trapezoid"):
        integral = scipy.integrate.trapezoid(f(x), x)
    elif(rule == "simpson"):
        integral = scipy.integrate.simpson(f(x), x=x)

    # i got pi + 1, when i evaluated the integral
    exact = np.pi + 1
    relative_error = np.abs((integral - exact)/exact)     # E_rel = (|I_approx - I_
exact|) / I_exact

    return relative_error



if __name__ == '__main__':
```

```
N = 5
error_t = np.zeros(N)
error_s = np.zeros(N)
for k in range(N):
    error_t[k] = approximate_integral(2**(k+1), "trapezoid")
    error_s[k] = approximate_integral(2**(k+1), "simpson")

h = (b-a) / (2**np.arange(1,N+1)) # N+1 due to dimension error

plt.plot(h, error_t, "o-", label="trapezoid")  # using dot line
plt.plot(h, error_s, "s-", label="Simpson")     # using square line
plt.xlabel("$h$")
plt.ylabel("relative error")
plt.legend()
plt.show()
# loglog slope = p
plt.loglog(h, error_t, "x-", label="trapezoid")
plt.loglog(h, error_s, "x-", label="Simpson")
plt.loglog(h, h**2, "--", label="$h^2$")   # Trapezoid rule → O(h^2)
plt.loglog(h, h**4, "--", label="$h^4$")   # Simpson rule → O(h^4)

plt.xlabel("$h$")
plt.ylabel("relative error")
plt.legend()
plt.show()

### Convergence tables
# E(h) = C * h^p  ⇒  log(E(h)) = log(C) + p * log(h)
# p = [log(E_i/E_(i+1)) / (log(h_i /h_(i+1)) ]
rate_t = np.log(error_t[:-1] / error_t[1:]) / np.log(h[:-1] / h[1:])
rate_s = np.log(error_s[:-1] / error_s[1:]) / np.log(h[:-1] / h[1:])

print("")
print(f"   h     |   E_T    |     p_T     |  E_S    |    p_S    ")
print(f"{h[0]:.2e} | {error_t[0]:.2e} |     --     | {error_s[0]:.2e} | -- ")
for i in range(1,N):
    print(f"{h[i]:.2e} | {error_t[i]:.2e} | {rate_t[i-1]:.4e} | {error_s[i]:.2e} | {ra
```

```
te_s[i-1]:.4e}")
    print("")
```

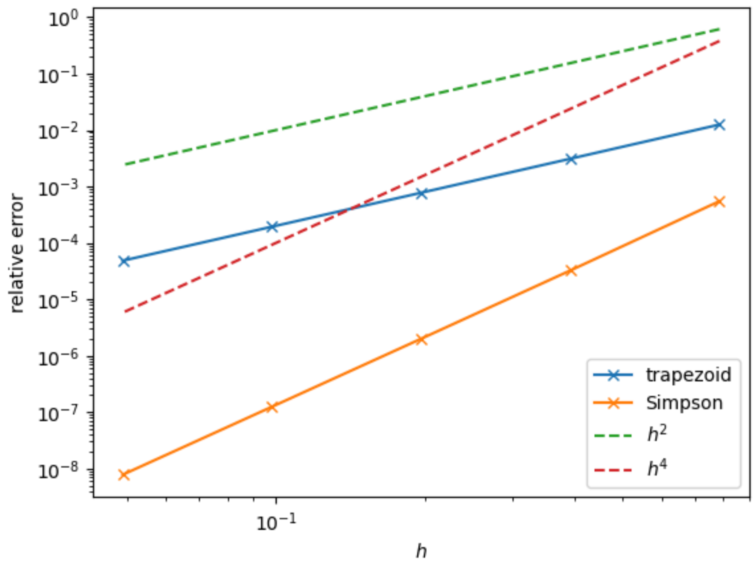
(C) Which rule is more accurate? Can you easily read the convergence order on the curve?



- Trapezoidal Rule : error decrease $\propto h^2$
- Simpson's Rule : error decrease $\propto h^4$

Thus, Simpson's rule is more accurate for the same n (smaller h)

(d) You should get straight lines for the curves h2 and h4, explain why. Which convergence order do you observe for the Trapezoidal and for the Simpson's rule?



- Plot $E(h) \propto h^p \rightarrow log(E) = log(C) + plog(h)$
- slope = p = convergence order
  - h^2 : slope 2
  - h^4 : slope 4

→ Thus, its linear due to the slope, and simpson's rule (slope 4) is lower than trapezodial

(e) compute the table of order from the errors.

```
(base) ➜  25Wise-Num2Eng git:(main) × python homework/homework0/convergence_template.py
   h     |   E_T    |    p_T     |   E_S    |    p_S
7.85e-01 | 1.25e-02 |     --     | 5.50e-04 | --
3.93e-01 | 3.11e-03 | 2.0113e+00 | 3.25e-05 | 4.0824e+00
1.96e-01 | 7.76e-04 | 2.0028e+00 | 2.00e-06 | 4.0200e+00
9.82e-02 | 1.94e-04 | 2.0007e+00 | 1.25e-07 | 4.0050e+00
4.91e-02 | 4.85e-05 | 2.0002e+00 | 7.79e-09 | 4.0012e+00
```

Exercise 0.4

(a) central difference

$$D_c^1 f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

approximates f′ with second order accuracy

$$D_c^1 f(x) = f'(x) + h^2 C_2$$

with $|C_2| \leq \frac{1}{6_{xi \in |x-h, x+h|}} max|f'''(\xi)|$

Taylor Expansions

Expand f(x+h) and f(x-h) :

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(\xi_1)}{6}h^3$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(\xi_2)}{6}h^3$$

Compute CD

$$D_c^1 f(x) = \frac{f(x+h) - f(x-h)}{2h}$$

$$= \frac{2f'(x)h) + \frac{h^3}{6}(f'''(\xi_1) + f'''(\xi_2))}{2h}$$

$$= f'(x) + \frac{h^2}{12}(f'''(\xi_1) + f'''(\xi_2))$$

Bound the Error Term

$$C_2 = \frac{1}{12}(f'''(\xi_1) + f'''(\xi_2))$$

$$|C_2| \leq \frac{1}{12} * 2_{\xi \in [x-h, x+h]} |f'''(\xi)| = \frac{1}{6}_{\xi \in [x-h, x+h]} * max|f'''(\xi)|$$

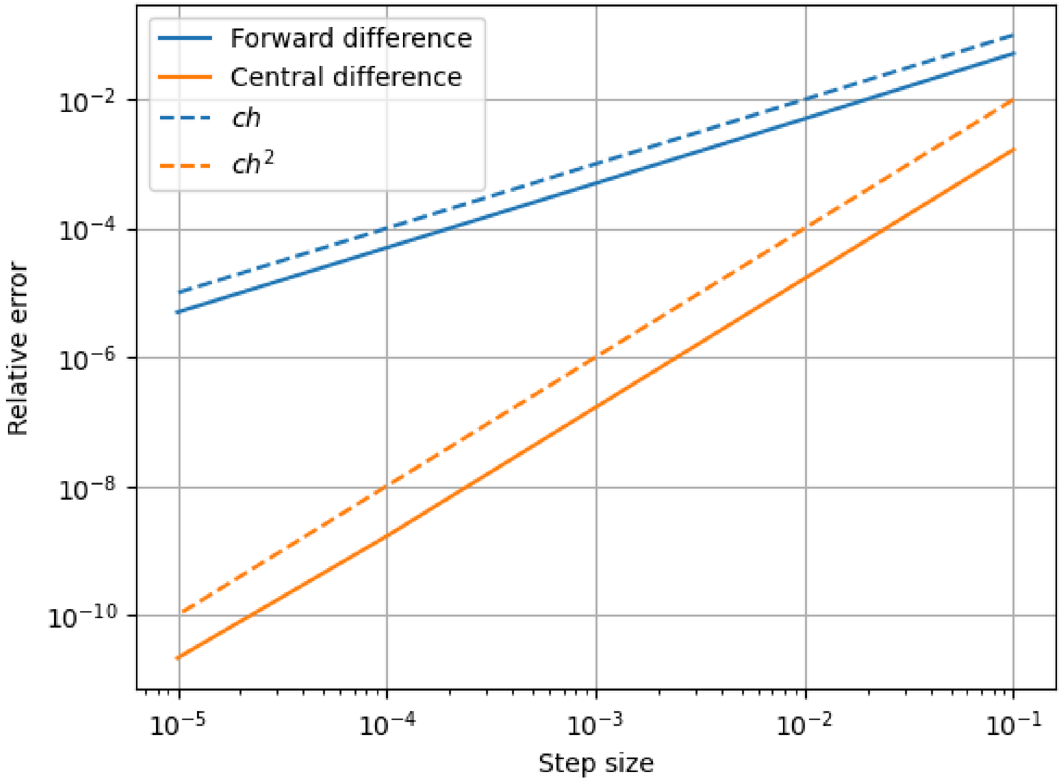
$$D_c^1 f(x) = f'(x) + h^2 C_2$$

with $|C_2| \leq \frac{1}{6}_{\xi \in [x-h, x+h]} * max|f'''(\xi)|$

(b) implement FD.py

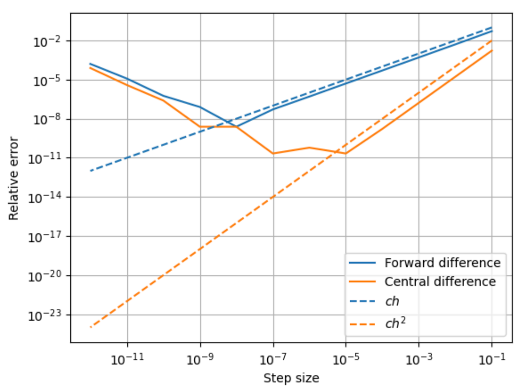
```
(base) ➜ 25Wise-Num2Eng git:(main) × python homework/homework0/F
D_template.py
Error of (FD) for a linear function: 8.882e-16
Error of (CD) for a quadratic function: 2.220e-16
```

(c)



(d)

h → -1 ~5



h→ -1 ~-12

When h→ -1 ~ -12, error increase again (because h is too small → round-off error)