ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

*****

LẬP TRÌNH CÁC THUẬT TOÁN PHƯƠNG PHÁP TÍNH

Bài tập:

Xây dựng các chương trình thuật toán trong phương pháp tính để giải quyết các bài tập bằng ngôn ngữ lập trình Python

Giảng viên giảng dạy: Thầy Lê Phê Đô

Người thực hiện:

Lê Trung Kiên 21021602 K66-ĐACLC2

Hà Nội, 2024

Khái quát

Đây là nhiệm vụ liên quan đến thi cuối kì: Lập trình các phương pháp đại số đã học trong chương trình để giải quyết các bài toán, cụ thể là các bài có đuôi là đuôi của mã sinh viên (số cuối mã sinh viên là "2").

Chương trình được lập trình trên nền tảng Google Colab, để chạy được, có thể mở file đính kèm đuôi .ipynb trên Google Colab để chạy. Trong file code, em đã phân chia các phần từ to đến nhỏ, từ chương đến các dạng, các bài và được đánh tiêu đề rất rõ ràng, đồng thời có cả ảnh đề bài.

Trong quá trình lập trình, em đã comment# giải thích chi tiết trong từng đoạn code và đồng thời kết quả cũng được in ra ngay phía dưới đoạn code. Có một số bài ở chương 7 kết quả là bảng thống kê, vì bảng hơi to so với chiều rộng khổ giấy nên nó bị nhảy dòng. Để kiểm tra lại kết quả thầy có thể chạy lại đoạn code để kiểm chứng ạ

Ở trang tiếp theo là trình bày toàn bộ đề bài + codes theo từng chương của em ạ!

#Thư viện

```
import numpy as np
import matplotlib.pyplot as plt
import math
import sympy as sp
from tabulate import tabulate
```

#CHƯƠNG 2: CÁC PP LẶP GIẢI PT

Phương pháp chia đôi

Bài 2

2. Let $f(x) = 3(x+1)(x-\frac{1}{2})(x-1)$. Use the Bisection method on the following intervals to find p_3.

a. $[-2, 1.5]$ **b.** $[-1.25, 2.5]$

```
import numpy as np

F = lambda x: 3 * (x + 1) * (x - 1/2) * (x - 1)

eps = 1e-10
#n: so lan lap
def Bisection_method(a, b, n, eps):
  cnt = 0
  while(1):
    p = (a + b) / 2
    if abs(F(p)) < eps:
      return p, cnt
    elif F(a) * F(p) < 0:
      b = p
    else:
      a = p
    cnt+= 1

#Cau a [a, b] = [-2, 1.5]
res, cnt = Bisection_method(-2, 1.5, 3, eps)

print('Giá trị nghiệm xấp xỉ là: ', res)
print('So lan lap: ', cnt)

#Cau b [a, b] = [-1.25, 2.5]
res, cnt = Bisection_method(-1.25, 2.5, 3, eps)
print('Giá trị nghiệm xấp xỉ là: ', res)
print('So lan lap: ', cnt)
```

```
Giá trị nghiệm xấp xỉ là:  -0.9999999999890861
So lan lap:  36
```

```
Giá trị nghiệm xấp xỉ là:  1.0000000000218279
So lan lap:  34
```

Bài 12

12. Find an approximation to $\sqrt{3}$ correct to within 10^{-4} using the Bisection Algorithm. [*Hint:* Consider $f(x) = x^2 - 3$.]

```
import numpy as np

# căn 3 ~ 1.7, ta chọn đoạn [1, 2]
a = 1
b = 2
eps = 1e-10
F = lambda x: x** 2 - 3

def bisection(a, b, eps):
  p_old = 0
  cnt = 0
  while (True):
    p = (a + b) / 2
    if(abs(p - p_old)/ abs(p) < eps):
      break
    else:
      p_old = p
    if F(a) * F(p) < 0:
      b = p
    else:
      a = p
    cnt+= 1
  return p, cnt

res, cnt = bisection(a, b, eps)
print('Kết quả chính xác: ', np.sqrt(3))
print('Kết quả xấp xỉ: ', res)
print('Số lần lặp: ', cnt)
print('Error', np.abs(res - np.sqrt(3)))

Kết quả chính xác:  1.7320508075688772
Kết quả xấp xỉ:  1.7320508075645193
Số lần lặp:  32
Error 4.3578474162586645e-12
```

PHƯƠNG PHÁP ĐIỂM BẤT ĐỘNG

Bài 2

1. Use algebraic manipulation to show that each of the following functions has a fixed point at p precisely when $f(p) = 0$, where $f(x) = x^4 + 2x^2 - x - 3$.

a. $g_1(x) = \left(3 + x - 2x^2\right)^{1/4}$

b. $g_2(x) = \left(\dfrac{x + 3 - x^4}{2}\right)^{1/2}$

c. $g_3(x) = \left(\dfrac{x + 3}{x^2 + 2}\right)^{1/2}$

d. $g_4(x) = \dfrac{3x^4 + 2x^2 + 3}{4x^3 + 4x - 1}$

2. **a.** Perform four iterations, if possible, on each of the functions g defined in Exercise 1. Let $p_0 = 1$ and $p_{n+1} = g(p_n)$, for $n = 0, 1, 2, 3$.

b. Which function do you think gives the best approximation to the solution?

```
#khoi tao
p0 = 1

F = lambda x: x** 4 + 2 * x** 2 - x - 3

g1 = lambda x: (3 + x - 2* x** 2)** (1/ 4)
g2 = lambda x: ((x + 3 - x** 4)/ 2)** (1/ 2)
g3 = lambda x: ((x + 3)/ (x** 2 + 2)/ 2)** (1/ 2)
g4 = lambda x: (3* x** 4 + 2* x** 2 + 3)/ (4* x** 3 + 4* x - 1)

def fixed_point_iteration(p0, g, n = 4):
  for i in range (4):
    p = g(p0)
    p0 = p
  return p

res1 = fixed_point_iteration(p0, g1)
res2 = fixed_point_iteration(p0, g2)
res3 = fixed_point_iteration(p0, g3)
res4 = fixed_point_iteration(p0, g4)

print('Nghiệm xấp xỉ sau 4 lần lặp')
print('Cau a:')
print('g1: ', res1)
print('Sai so: ', abs(F(res1)))
print()

print('Cau b: ')
print('g2: ', res2)
print('Sai so: ', abs(F(res2)))
print()

print('Cau c: ')
print('g3: ', res3)
print('Sai so: ', abs(F(res3)))
print()
```

```python
print('Cau d: ')
print('g4: ', res4)
print('Sai so: ', abs(F(res4)))

print()
print('Dựa vào kiểm tra F(p), ta thấy phương trình d cho xấp xỉ tốt nhất')
```

```
Nghiệm xấp xỉ sau 4 lần lặp
Cau a:
g1:  1.1078205295102599
Sai so:  0.14710524433896754

Cau b:
g2:  0.9875064291508866
Sai so:  1.0862140575309671

Cau c:
g3:  0.8421732238188103
Sai so:  1.9206180427968524

Cau d:
g4:  1.124123029704334
Sai so:  1.7141843500212417e-13

Dựa vào kiểm tra F(p), ta thấy phương trình d cho xấp xỉ tốt nhất
```

Bài 12

12. For each of the following equations, use the given interval or determine an interval $[a, b]$ on which fixed-point iteration will converge. Estimate the number of iterations necessary to obtain approximations accurate to within 10^{-5}, and perform the calculations.

a. $2 + \sin x - x = 0$ use $[2, 3]$

b. $x^3 - 2x - 5 = 0$ use $[2, 3]$

c. $3x^2 - e^x = 0$

d. $x - \cos x = 0$

```python
#khoi tao
a = 2
b = 3
eps = 1e-10

g1 = lambda x: 2 + np.sin(x)
g2 = lambda x: (2* x + 5)** (1/ 3)
g3 = lambda x: np.log(3 * x** 2)
g4 = lambda x: np.cos(x)

def fixed_point_iteration(a, b, g, eps):
  p_old = (a + b)/ 2
  cnt = 0
  while (True):
```

```
    p = g(p_old)
    if(abs(p - p_old)/ abs(p) < eps):
      break
    else:
      p_old = p
    cnt+= 1
  return np.round(p, 10), cnt

print('Cau a: ')
res, count = fixed_point_iteration(a, b, g1, eps)
print('Ket qua xap xi: ', res)
print('So lan lap: ', count)

print()
print('Cau b: ')
res, count = fixed_point_iteration(a, b, g2, eps)
print('Ket qua xap xi: ', res)
print('So lan lap: ', count)

print()
print('Cau c: ')
res, count = fixed_point_iteration(a, b, g3, eps)
print('Ket qua xap xi: ', res)
print('So lan lap: ', count)

print()
print('Cau d: ')
res, count = fixed_point_iteration(a, b, g4, eps)
print('Ket qua xap xi: ', res)
print('So lan lap: ', count)
```

```
Cau a:
Ket qua xap xi:  2.5541959529
So lan lap:  108

Cau b:
Ket qua xap xi:  2.0945514816
So lan lap:  12

Cau c:
Ket qua xap xi:  3.7330790283
So lan lap:  35

Cau d:
Ket qua xap xi:  0.7390851332
So lan lap:  55
```

PHƯƠNG PHÁP LẶP NEWTON, ĐIỂM SAI, DÂY CUNG

Bài 2

2. Let $f(x) = -x^3 - \cos x$ and $p_0 = -1$. Use Newton's method to find p_2. Could $p_0 = 0$ be used?

```
import sympy as sp
#find p2
p0 = -1
x = sp.symbols('x')
F = -x** 3 - sp.cos(x)
eps = 1e-10

cnt = 0
#n: so lan lap
def Newton_method(p0, F, n, eps):
  cnt = 0
  while(1):
    if sp.diff(F, x, 1).subs(x, p0).evalf() == 0:
      return 'Error!'
    p = p0 - F.subs(x, p0).evalf() / sp.diff(F, x, 1).subs(x,
p0).evalf()
    if (abs(p - p0)/ abs(p) < eps):
      return p, cnt
    p0 = p
    cnt+= 1

p2, cnt = Newton_method(p0, F, 2, eps)
print('Câu a')
print('Giá trị của nghiệm xấp xỉ = ', p2)
print('Số lần lặp: ', cnt)
print()

#Kiem tra voi p0 = 0
p0 = 0
print('Câu b')
print('Kiểm tra với p0 = 0')
if Newton_method(p0, F, 2, eps) == 'Error!':
  print('Không dùng được vì f’(p0) = 0 hoặc f(p0).f’’(p0) < 0')
else:
  p2, cnt= Newton_method(p0, F, 2, eps)
  print('Giá trị của p2 = ', p2)
  print('Số lần lặp: ', cnt)
```

```
Câu a
Giá trị của nghiệm xấp xỉ =  -0.865474033101614
Số lần lặp:  4

Câu b
Kiểm tra với p0 = 0
Không dùng được vì f’(p0) = 0 hoặc f(p0).f’’(p0) < 0
```

Bài 12

12. Use all three methods in this Section to find solutions to within 10^{-7} for the following problems.

a. $x^2 - 4x + 4 - \ln x = 0$ for $1 \leq x \leq 2$ and for $2 \leq x \leq 4$

b. $x + 1 - 2\sin \pi x = 0$ for $0 \leq x \leq 1/2$ and for $1/2 \leq x \leq 1$

```
import sympy as sp
eps = 1e-10
x = sp.Symbol('x')
F1 = x**2 - 4*x + 4 - sp.log(x)
F2 = x+1 - 2*sp.sin(sp.pi * x)

def Newton_method(p0, F, eps):
  cnt = 0
  while(1):
    p = p0 - F.subs(x, p0).evalf() / sp.diff(F, x, 1).subs(x,
p0).evalf()
    if abs(p - p0)/ abs(p) < eps:
      return p, cnt
    cnt+= 1
    p0 = p

def Secant_method(f, x0, x1, eps):
    cnt = 0
    while(1):
        f_x0 = f.subs(x, x0).evalf()
        f_x1 = f.subs(x, x1).evalf()
        x_n = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        if abs(x_n - x1) < eps:
            return x_n, cnt
        cnt+= 1
        x0 = x1
        x1 = x_n

def false_position(f, a, b, eps):
    x = sp.Symbol('x')
    cnt = 0
    while(1):
        f_a = f.subs(x, a).evalf()
        f_b = f.subs(x, b).evalf()

        # Áp dụng công thức phương pháp dây cung
        x_next = (a*f_b - b*f_a) / (f_b - f_a)

        f_x_next = f.subs(x, x_next).evalf()

        if abs(f_x_next) < eps:
            return x_next, cnt
        cnt+= 1

        if f_a * f_x_next < 0:
            b = x_next
```

```
        else:
            a = x_next


print('Câu a')
# print('F1 = x^2 - 4x + 4 - lnx')
print(' - x thuoc [1, 2]')
print('Newton method: ', Newton_method(1.5, F1, eps)[0])
print('Số lần lặp: ', Newton_method(1.5, F1, eps)[1])
print('Secant method: ', Secant_method(F1, 1, 2, eps)[0])
print('Số lần lặp: ', Secant_method(F1, 1, 2, eps)[1])
print('False position method: ', false_position(F1, 1, 2, eps)[0])
print('Số lần lặp: ', false_position(F1, 1, 2, eps)[1])

print()

print(' - x thuoc [2, 4]')
print('Newton method: ', Newton_method(3, F1, eps)[0])
print('Số lần lặp: ', Newton_method(3, F1, eps)[1])
print('Secant method: ', Secant_method(F1, 2, 4, eps)[0])
print('Số lần lặp: ', Secant_method(F1, 2, 4, eps)[1])
print('False position method: ', false_position(F1, 2, 4, eps)[0])
print('Số lần lặp: ', false_position(F1, 2, 4, eps)[1])


print()
print('Câu b')
# print(' F2 = x + 1 - 2sin(pi.x)')
print(' - x thuoc [0, 1/2]')
print('Newton method: ', Newton_method(0.25, F2, eps)[0])
print('Số lần lặp: ', Newton_method(0.25, F2, eps)[1])
print('Secant method: ', Secant_method(F2, 0, 1/2, eps)[0])
print('Số lần lặp: ', Secant_method(F2, 0, 1/2, eps)[1])
print('False position method: ', false_position(F2, 0, 1/2, eps)[0])
print('Số lần lặp: ', false_position(F2, 0, 1/2, eps)[1])

print()
print(' - x thuoc [1/2, 1]')
print('Newton method: ', Newton_method(0.75, F2, eps)[0])
print('Số lần lặp: ', Newton_method(0.75, F2, eps)[1])
print('Secant method: ', Secant_method(F2, 1/2, 1, eps)[0])
print('Số lần lặp: ', Secant_method(F2, 1/2, 1, eps)[1])
print('False position method: ', false_position(F2, 1/2, 1, eps)[0])
print('Số lần lặp: ', false_position(F2, 1/2, 1, eps)[1])
```

```
Câu a
 - x thuoc [1, 2]
Newton method:  1.41239117202388
Số lần lặp:  4
Secant method:  1.41239117202388
```

```
Số lần lặp:  7
False position method:  1.41239117205790
Số lần lặp:  15

 - x thuoc [2, 4]
Newton method:  3.05710354999474
Số lần lặp:  3
Secant method:  3.05710354999474
Số lần lặp:  8
False position method:  3.05710354994707
Số lần lặp:  23

Câu b
 - x thuoc [0, 1/2]
Newton method:  0.206035119570966
Số lần lặp:  4
Secant method:  0.206035119570966
Số lần lặp:  8
False position method:  0.206035119575608
Số lần lặp:  14

 - x thuoc [1/2, 1]
Newton method:  0.681974808738647
Số lần lặp:  4
Secant method:  0.681974808738647
Số lần lặp:  7
False position method:  0.681974808725777
Số lần lặp:  19
```

Bài 22

22. Use Maple to determine how many iterations of Newton's method with $p_0 = \pi/4$ are needed to find a root of $f(x) = \cos x - x$ to within 10^{-100}.

```python
from mpmath import mp

# Thiết lập độ chính xác cho tính toán
mp.dps = 100

# Phương trình cần giải
def f(x):
    return mp.cos(x) - x

# Đạo hàm của hàm số f
def f_prime(x):
    return -mp.sin(x) - 1

# Điểm bắt đầu
x0 = mp.pi / 4
```

```python
# Độ chính xác mong muốn
tolerance = mp.power(10, -100)

# Số lần lặp
num_iterations = 0

# Áp dụng phương pháp Newton
while(1):
    x_next = x0 - f(x0) / f_prime(x0)
    num_iterations += 1

    if mp.fabs(x_next - x0)/ mp.fabs(x_next) < tolerance:
        break

    x0 = x_next

print("Số lần lặp cần thiết:", num_iterations)
```

```
Số lần lặp cần thiết: 7
```

#CHƯƠNG 3: GIẢI HPT

KHỬ GAUSS

```python
import numpy as np

def gauss_jordan_elimination(A, b):
    n = len(b)
    # Nối ma trận A và vector b thành ma trận tổng hợp [A | b]
    augmented_matrix = np.hstack((A.astype(float),
np.expand_dims(b.astype(float), axis=1)))

    # Quá trình khử
    for i in range(n):
        # Chọn pivot
        pivot_row = i
        for j in range(i + 1, n):
            if abs(augmented_matrix[j, i]) >
abs(augmented_matrix[pivot_row, i]):
                pivot_row = j

        # Hoán đổi hàng để đưa pivot về vị trí chính giữa
        augmented_matrix[[pivot_row, i]] = augmented_matrix[[i,
pivot_row]]

        # Chia hàng pivot cho giá trị của pivot để đưa về 1
        pivot_value = augmented_matrix[i, i]
        augmented_matrix[i] /= pivot_value

        # Loại bỏ phần tử pivot từ các hàng khác
```

```
        for j in range(n):
            if j != i:
                factor = augmented_matrix[j, i]
                augmented_matrix[j] -= factor * augmented_matrix[i]

    # Trả về ma trận kết quả
    return augmented_matrix[:, :-1]

# Ví dụ:
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 10]])
b = np.array([1, 2, 3])

solution = gauss_jordan_elimination(A, b)
print("Nghiệm của hệ phương trình:")
print(solution)
```

```
Nghiệm của hệ phương trình:
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [-0. -0.  1.]]
```

PHƯƠNG PHÁP A = LU

Doolittle

Bài 2

1–5 DOOLITTLE'S METHOD

Show the factorization and solve by Doolittle's method.

1. $$4x_1 + 5x_2 = 14$$
$$12x_1 + 14x_2 = 36$$

2. $$2x_1 + 9x_2 = 82$$
$$3x_1 - 5x_2 = -62$$

```python
def doolittle(A, b):
    n = len(A)
    L = np.zeros((n, n))  # Khởi tạo ma trận L với kích thước nxn chứa toàn số 0
    U = np.zeros((n, n))  # Khởi tạo ma trận U với kích thước nxn chứa toàn số 0

    for k in range(n):
        # Tính toán các phần tử của ma trận U từ cột k trở đi
        U[k, k:] = A[k, k:] - np.dot(L[k, :k], U[:k, k:])
        # Tính toán các phần tử của ma trận L từ hàng k trở đi
        L[k:, k] = (A[k:, k] - np.dot(L[k:, :k], U[:k, k])) / U[k, k]

    return L, U  # Trả về ma trận L và U

def forward_substitution(L, b):
    n = len(b)
    y = np.zeros(n)  # Khởi tạo vector y chứa toàn số 0

    for i in range(n):
        # Tính toán từng phần tử của vector y bằng phép thế tiến
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
```

```python
    return y  # Trả về vector y

def back_substitution(U, y):
    n = len(y)
    x = np.zeros(n)  # Khởi tạo vector x chứa toàn số 0

    for i in range(n - 1, -1, -1):
        # Tính toán từng phần tử của vector x bằng phép thế lùi
        x[i] = (y[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i, i]

    return x  # Trả về vector x

# Ví dụ:
A = np.array([[2, 9],
              [3, -5]])
b = np.array([82, -62])

# Phân rã ma trận A thành L và U
L, U = doolittle(A, b)

# Giải hệ phương trình Ly = b bằng phép thế tiến
y = forward_substitution(L, b)

# Giải hệ phương trình Ux = y bằng phép thế lùi
x = back_substitution(U, y)

print("Ma trận L:")
print(L)
print("\nMa trận U:")
print(U)
print("\nNghiệm của hệ phương trình:")
print(x)
```

```
Ma trận L:
[[1.  0. ]
 [1.5 1. ]]

Ma trận U:
[[  2.    9. ]
 [  0.  -18.5]]

Nghiệm của hệ phương trình:
[-4. 10.]
```

Cholesky

Bài 12

$$
\begin{aligned}
\textbf{12.}\quad 4x_1 + 2x_2 + 4x_3 \phantom{{}+ 2x_4} &= 20 \\
2x_1 + 2x_2 + 3x_3 + 2x_4 &= 36 \\
4x_1 + 3x_2 + 6x_3 + 3x_4 &= 60 \\
2x_2 + 3x_3 + 9x_4 &= 122
\end{aligned}
$$

```python
import numpy as np

def cholesky_decomposition(A):
    # Lấy kích thước của ma trận A
    n = len(A)
    # Tạo ma trận L ban đầu với các giá trị bằng 0
    L = np.zeros((n, n))

    # Duyệt qua từng phần tử trong ma trận L
    for i in range(n):
        for j in range(i + 1):
            if i == j:
                # Tính giá trị đường chéo L[i, j]
                L[i, j] = np.sqrt(A[i, i] - np.sum(L[i, :i]**2))
            else:
                # Tính các giá trị ngoài đường chéo
                L[i, j] = (A[i, j] - np.sum(L[i, :j] * L[j, :j])) /
L[j, j]

    return L

def forward_substitution(L, b):
    # Lấy kích thước của vector b
    n = len(b)
    # Tạo vector y ban đầu với các giá trị bằng 0
    y = np.zeros(n)

    # Thực hiện thế tiến (forward substitution)
    for i in range(n):
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]

    return y
```

```
def back_substitution(L_transpose, y):
    # Lấy kích thước của vector y
    n = len(y)
    # Tạo vector x ban đầu với các giá trị bằng 0
    x = np.zeros(n)

    # Thực hiện thế lùi (back substitution)
    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(L_transpose[i, i + 1:], x[i + 1:])) /
L_transpose[i, i]

    return x

# Ví dụ:
A = np.array([[4, 2, 4, 0],
              [2, 2, 3, 2],
              [4, 3, 6, 3],
              [0, 2, 3, 9]])
b = np.array([20, 36, 60, 122])

# Thực hiện phân rã Cholesky trên ma trận A
L = cholesky_decomposition(A)
# Giải hệ phương trình Ly = b bằng phương pháp thế tiến
y = forward_substitution(L, b)
# Giải hệ phương trình L^T x = y bằng phương pháp thế lùi
x = back_substitution(L.T, y)

# In ra kết quả ma trận L và nghiệm x
print("Ma trận L (phân rã Cholesky):")
print(L)
print("\nNghiệm của hệ phương trình:")
print(x)
```

```
Ma trận L (phân rã Cholesky):
[[2. 0. 0. 0.]
 [1. 1. 0. 0.]
 [2. 1. 1. 0.]
 [0. 2. 1. 2.]]

Nghiệm của hệ phương trình:
[ 6. -2.  0. 14.]
```

Gauss-seidel

```
import numpy as np

def gauss_seidel(A, b, initial_guess, tolerance=1e-10,
max_iterations=1000):
    n = len(b)  # Số lượng ẩn trong hệ phương trình
```

```python
    x = initial_guess.copy()  # Sao chép giá trị ước lượng ban đầu
    x_new = np.zeros_like(x)  # Tạo vector x_new với các giá trị ban đầu là 0, cùng kích thước với x

    for k in range(max_iterations):  # Lặp lại tối đa số lần lặp cho phép
        for i in range(n):  # Duyệt qua từng ẩn
            # Tính giá trị mới cho x_new[i] dựa trên công thức Gauss-Seidel
            x_new[i] = (b[i] - np.dot(A[i, :i], x_new[:i]) - np.dot(A[i, i + 1:], x[i + 1:])) / A[i, i]

        # Kiểm tra điều kiện hội tụ
        if np.linalg.norm(x_new - x) < tolerance:
            return x_new, k  # Trả về nghiệm và số lần lặp nếu hội tụ

        x = x_new.copy()  # Cập nhật giá trị của x để sử dụng trong lần lặp tiếp theo

    raise ValueError("Không hội tụ sau {} lần lặp.".format(max_iterations))  # Ném lỗi nếu không hội tụ sau số lần lặp cho phép

# Ví dụ:
A = np.array([[10, -1, 2],  # Ma trận hệ số
              [-1, 11, -1],
              [2, -1, 10]])
b = np.array([6, 25, -11])  # Vector hằng số
initial_guess = np.zeros(len(b))  # Ước lượng ban đầu là vector các giá trị 0

solution, count = gauss_seidel(A, b, initial_guess)  # Gọi hàm Gauss-Seidel để tìm nghiệm
print("Nghiệm của hệ phương trình:")  # In ra nghiệm của hệ phương trình
print(solution)
print('Số bước lặp: ')  # In ra số bước lặp đã thực hiện
print(count)
```

```
Nghiệm của hệ phương trình:
[ 1.04326923  2.26923077 -1.08173077]
Số bước lặp:
9
```

JACOBI

```python
def jacobi(A, b, initial_guess, tolerance=1e-10, max_iterations=1000):
    n = len(b)  # Số lượng ẩn trong hệ phương trình
    x = initial_guess.copy()  # Sao chép giá trị ước lượng ban đầu
```

```python
    x_new = np.zeros_like(x)  # Tạo vector x_new với các giá trị ban đầu là 0, cùng kích thước với x

    for k in range(max_iterations):  # Lặp lại tối đa số lần lặp cho phép
        for i in range(n):  # Duyệt qua từng ẩn
            # Tính giá trị mới cho x_new[i] dựa trên công thức Jacobi
            x_new[i] = (b[i] - np.dot(A[i, :i], x[:i]) - np.dot(A[i, i + 1:], x[i + 1:])) / A[i, i]

        # Kiểm tra điều kiện hội tụ
        if np.linalg.norm(x_new - x) < tolerance:
            return x_new, k  # Trả về nghiệm và số lần lặp nếu hội tụ

        x = x_new.copy()  # Cập nhật giá trị của x để sử dụng trong lần lặp tiếp theo

    raise ValueError("Không hội tụ sau {} lần lặp.".format(max_iterations))  # Ném lỗi nếu không hội tụ sau số lần lặp cho phép

# Ví dụ:
A = np.array([[10, -1, 2],  # Ma trận hệ số
              [-1, 11, -1],
              [2, -1, 10]])
b = np.array([6, 25, -11])  # Vector hằng số
initial_guess = np.zeros(len(b))  # Ước lượng ban đầu là vector các giá trị 0

solution, count = jacobi(A, b, initial_guess)  # Gọi hàm Jacobi để tìm nghiệm
print("Nghiệm của hệ phương trình:")  # In ra nghiệm của hệ phương trình
print(solution)
print('Số bước lặp: ')  # In ra số bước lặp đã thực hiện
print(count)
```

```
Nghiệm của hệ phương trình:
[ 1.04326923  2.26923077 -1.08173077]
Số bước lặp:
18
```

#CHƯƠNG 4: NỘI SUY

Nội suy Lagrange

Bài 2

2. For the given functions $f(x)$, let $x_0 = 1$, $x_1 = 1.25$, and $x_2 = 1.6$. Construct interpolation polynomials of degree at most one and at most two to approximate $f(1.4)$, and find the absolute error.

a. $f(x) = \sin \pi x$

b. $f(x) = \sqrt[3]{x-1}$

c. $f(x) = \log_{10}(3x - 1)$

d. $f(x) = e^{2x} - x$

```
x0 = 1
x1 = 1.25
x2 = 1.6

#Cau a
Fa = lambda x: np.sin(np.pi * x)
#Cau c
Fc = lambda x: np.log10(3 * x - 1)
#Cau b
Fb = lambda x: (x - 1)** (1/ 3)
#Cau d
Fd = lambda x: np.exp(2 *x) - x

#Noi suy Lagrange bac nhat
def Lagrange_of_degree_one(x0, x1, F, x):
  return F(x0) * (x - x1) / (x0 - x1) + F(x1) * (x - x0) / (x1 - x0)
#Noi duy Lagrange bac hai
def Lagrange_of_degree_two(x0, x1, x2, F, x):
  sum = 0
  F0 = F(x0)
  F1 = F(x1)
  F2 = F(x2)
  X = np.array([x0, x1, x2])
  Y = np.array([F0, F1, F2])
  for i in range(0, 3):
    X_without_i = np.concatenate((X[:i], X[i+1:]))
    a = np.prod((x - X_without_i))
    b = np.prod(X[i] - X_without_i)
    tmp = a/ b
    sum+= tmp * Y[i]
  return sum

print('Câu a')
print('Nội suy Lagrange bậc nhất: f(1.4) = ',
Lagrange_of_degree_one(x0, x1, Fa, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fa(1.4) -
Lagrange_of_degree_one(x0, x1, Fa, 1.4)))
print('Nội suy Lagrange bậc hai: f(1.4) = ',
Lagrange_of_degree_two(x0, x1, x2, Fa, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fa(1.4) -
Lagrange_of_degree_two(x0, x1, x2, Fa, 1.4)))

print()
print('Câu b')
```

```python
print('Nội suy Lagrange bậc nhất: f(1.4) = ',
Lagrange_of_degree_one(x0, x1, Fb, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fb(1.4) -
Lagrange_of_degree_one(x0, x1, Fb, 1.4)))
print('Nội suy Lagrange bậc hai: f(1.4) = ',
Lagrange_of_degree_two(x0, x1, x2, Fb, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fb(1.4) -
Lagrange_of_degree_two(x0, x1, x2, Fb, 1.4)))

print()
print('Câu c')
print('Nội suy Lagrange bậc nhất: f(1.4) = ',
Lagrange_of_degree_one(x0, x1, Fc, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fc(1.4) -
Lagrange_of_degree_one(x0, x1, Fc, 1.4)))
print('Nội suy Lagrange bậc hai: f(1.4) = ',
Lagrange_of_degree_two(x0, x1, x2, Fc, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fc(1.4) -
Lagrange_of_degree_two(x0, x1, x2, Fc, 1.4)))

print()
print('Câu d')
print('Nội suy Lagrange bậc nhất: f(1.4) = ',
Lagrange_of_degree_one(x0, x1, Fd, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fd(1.4) -
Lagrange_of_degree_one(x0, x1, Fd, 1.4)))
print('Nội suy Lagrange bậc hai: f(1.4) = ',
Lagrange_of_degree_two(x0, x1, x2, Fd, 1.4))
print('Sai số tuyệt đối: E = ', abs(Fd(1.4) -
Lagrange_of_degree_two(x0, x1, x2, Fd, 1.4)))
```

```
Câu a
Nội suy Lagrange bậc nhất: f(1.4) =  -1.1313708498984756
Sai số tuyệt đối: E =  0.18031433360332205
Nội suy Lagrange bậc hai: f(1.4) =  -0.9182280617406016
Sai số tuyệt đối: E =  0.03282845455455197

Câu b
Nội suy Lagrange bậc nhất: f(1.4) =  1.0079368399158983
Sai số tuyệt đối: E =  0.271130540187821
Nội suy Lagrange bậc hai: f(1.4) =  0.8169446700381561
Sai số tuyệt đối: E =  0.08013837031007875

Câu c
Nội suy Lagrange bậc nhất: f(1.4) =  0.5223143127300315
Sai số tuyệt đối: E =  0.01716433441012566
Nội suy Lagrange bậc hai: f(1.4) =  0.507122062831104
Sai số tuyệt đối: E =  0.0019720845111981244

Câu d
```

```
Nội suy Lagrange bậc nhất: f(1.4) =  13.658556677767166
Sai số tuyệt đối: E =  1.3860900933298819
Nội suy Lagrange bậc hai: f(1.4) =  15.269763314888284
Sai số tuyệt đối: E =  0.2251165437912359
```

Bài 12

12. Use the Lagrange interpolating polynomial of degree three or less and four-digit chopping arithmetic to approximate $\cos 0.750$ using the following values. Find an error bound for the approximation.

$$\cos 0.698 = 0.7661 \quad \cos 0.733 = 0.7432 \quad \cos 0.768 = 0.7193 \quad \cos 0.803 = 0.6946$$

The actual value of $\cos 0.750$ is 0.7317 (to four decimal places). Explain the discrepancy between the actual error and the error bound.

```python
from sympy import Matrix
X = np.array([0.698, 0.733, 0.768, 0.803])
Y = np.array([0.7551, 0.7432, 0.7193, 0.6946])

x = sp.symbols('x')
F = sp.cos(x)
#Exact Value cos0.75 = 0.7317

#Chopping 4 digits
def chopping_4_digits(x):
  x = x* np.power(10, 4)
  x = int(x)/ np.power(10, 4)
  return x


def Lagrange_of_degree_three(X, Y, x):
  sum = 0
  for i in range(0, 4):
    X_without_i = np.concatenate((X[:i], X[i+1:]))
    a = np.prod((x - X_without_i))
    b = np.prod(X[i] - X_without_i)
    tmp = a/ b
    sum+= tmp * Y[i]
  return sum

def Lagrange_error_bound(X):
  n = 3
  #Dao ham cap n + 1 cua cosx
  F = lambda x: np.cos(x)
  #tinh (n+1)!
  s = 1
  for i in range(1, n + 2):
    s *= i
  x = np.linspace(min(X), max(X), 10)
```

```
  M = 1
  m = 0
  for i in x:
    m = max(m, abs(np.prod(i - X)))
  return M * m/ s

res = chopping_4_digits(Lagrange_of_degree_three(X, Y, 0.75))
print('Giá trị xấp xỉ: ', res)
print('Sai sô´tuyệt đôí: ', chopping_4_digits(abs(0.7317 - res)))
# print('Error_bound: ', Lagrange_error_bound(X))
# print('Lí do Error bound << Sai sô´ tuyệt đô´i là do khoảng (a, b)
khá bé, !0.105')
```

```
Giá trị xấp xỉ:  0.7323
Sai sô´tuyệt đôí:  0.0005
```

Nội suy Newton

Bài 2

2. Use Eq. (3.10) or Algorithm 3.2 to construct interpolating polynomials of degree one, two, and three for the following data. Approximate the specified value using each of the polynomials.

a. $f(0.43)$ if $f(0) = 1$, $f(0.25) = 1.64872$, $f(0.5) = 2.71828$, $f(0.75) = 4.48169$

b. $f(0)$ if $f(-0.5) = 1.93750$, $f(-0.25) = 1.33203$, $f(0.25) = 0.800781$, $f(0.5) = 0.687500$

```
#Tinh toan bang sai phan
def newton_divided_difference(x, y):
    n = len(y)
    coef = np.zeros([n, n])
    coef[:,0] = y

    for j in range(1, n):
        for i in range(n-j):
            coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j] -
x[i])

    return coef

#n la bac da thuc
def newton_polynomial(coef, X, x, n):
  p = coef[0][0]
  for i in range(1, n+1):
    p+= coef[0][i] * np.prod(x - X[:i])
  return p


#Cau a
X = np.array([0, 0.25, 0.5, 0.75])
Y = np.array([1, 1.64872, 2.71828, 4.48169])
coef = newton_divided_difference(X, Y)
```

```
print(coef)

print('Cau a')
print('Đa thức bậc nhất, f(0.43) = ', newton_polynomial(coef, X, 0.43,
1))
print('Đa thức bậc hai, f(0.43) = ', newton_polynomial(coef, X, 0.43,
2))
print('Đa thức bậc ba, f(0.43) = ', newton_polynomial(coef, X, 0.43,
3))
print()

#Cau b
X = np.array([-0.5, -0.25, 0.25, 0.5])
Y = np.array([1.93750, 1.33203, 0.800781, 0.687500])
coef = newton_divided_difference(X, Y)
print(coef)

print('Cau b')
print('Đa thức bậc nhất, f(0) = ', newton_polynomial(coef, X, 0, 1))
print('Đa thức bậc hai, f(0) = ', newton_polynomial(coef, X, 0, 2))
print('Đa thức bậc ba, f(0) = ', newton_polynomial(coef, X, 0, 3))
```

```
[[1.          2.59488     3.36672     2.91210667]
 [1.64872     4.27824     5.5508      0.        ]
 [2.71828     7.05364     0.          0.        ]
 [4.48169     0.          0.          0.        ]]
Cau a
Đa thức bậc nhất, f(0.43) =  2.1157984
Đa thức bậc hai, f(0.43) =  2.376382528
Đa thức bậc ba, f(0.43) =  2.3606047340800003

[[ 1.9375     -2.42188     1.81250933 -1.00001067]
 [ 1.33203    -1.062498    0.81249867  0.        ]
 [ 0.800781   -0.453124    0.          0.        ]
 [ 0.6875      0.          0.          0.        ]]
Cau b
Đa thức bậc nhất, f(0) =  0.7265600000000001
Đa thức bậc hai, f(0) =  0.9531236666666667
Đa thức bậc ba, f(0) =  0.984374
```

Nội suy spline

Bài 2

2. Determine the clamped cubic spline s that interpolates the data $f(0) = 0$, $f(1) = 1$, $f(2) = 2$ and satisfies $s'(0) = s'(2) = 1$.

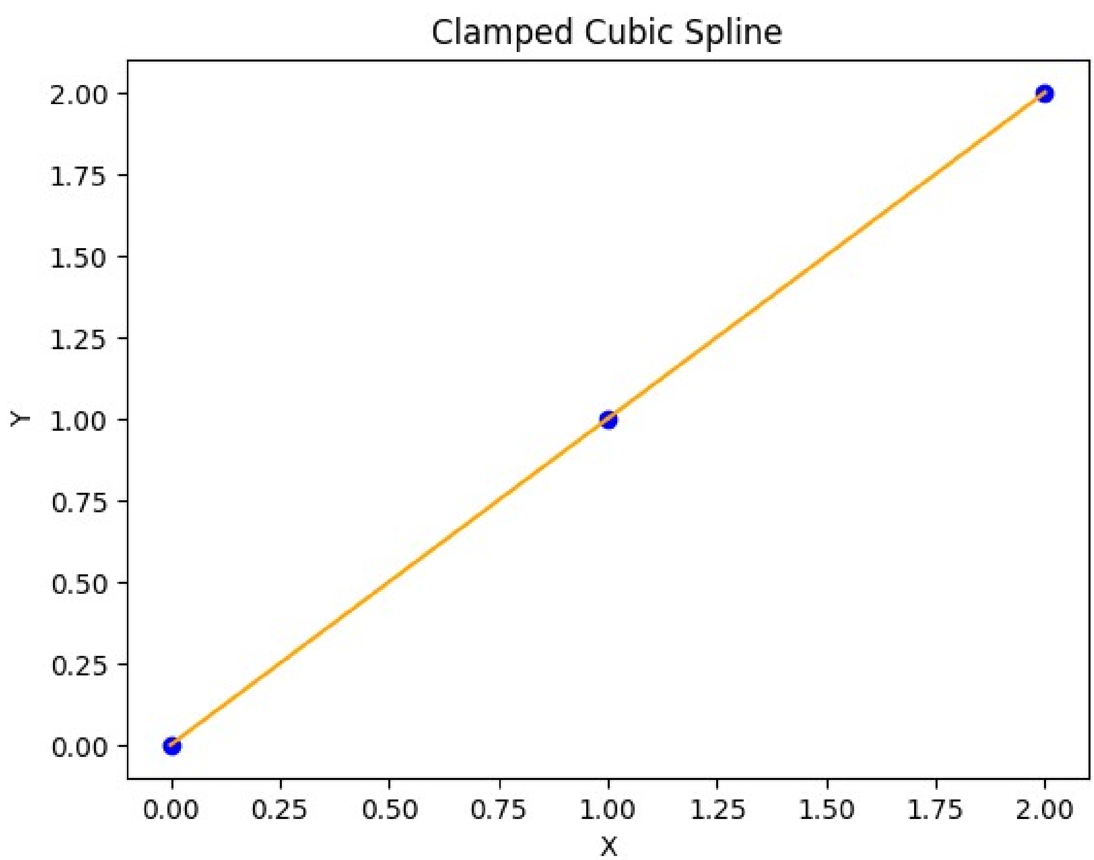
```
#Natural Cubic Spline

def Natural_cubic_spline(X, Y, x):
```

```python
    n = len(X)  # Số lượng điểm dữ liệu
    h = np.diff(X)  # Tính khoảng cách giữa các điểm X (h_i = X_{i+1} - X_i)

    # Khởi tạo ma trận A và vector b
    A = np.zeros((n, n))
    b = np.zeros(n)

    # Điều kiện biên tự nhiên: c_0 = 0 và c_n = 0
    A[0, 0] = 1
    A[n - 1, n - 1] = 1

    # Thiết lập các phương trình cho các điểm nội suy
    for i in range(1, n - 1):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 3 * ((Y[i + 1] - Y[i]) / h[i] - (Y[i] - Y[i - 1]) / h[i - 1])

    # Giải hệ phương trình tuyến tính A*c = b để tìm các hệ số c
    c = np.linalg.solve(A, b)

    # Tính các hệ số a, b, c, d cho các đoạn spline
    a = Y[:-1]
    b = (Y[1:] - Y[:-1]) / h - h * (2 * c[:-1] + c[1:]) / 3
    d = np.diff(c) / (3 * h)
    c = c[:-1]

    # In ra các phương trình spline cho từng đoạn
    for i in range(0, n - 1):
        print(f'{a[i]} + {b[i]} * (x - {X[i]}) + {c[i]} * (x - {X[i]})**2 + {d[i]} * (x - {X[i]})**3, x thuộc [{X[i]}, {X[i + 1]}]')

    # Tìm đoạn spline chứa x và tính giá trị nội suy tại x
    i = np.searchsorted(X, x) - 1
    i = np.clip(i, 0, n - 1)

    dx = x - X[i]
    y = a[i] + b[i] * dx + c[i] * dx ** 2 + d[i] * dx ** 3
    return y

def Clamped_cubic_spline(X, Y, x, a, c):
    n = len(X)  # Số lượng điểm dữ liệu
    h = np.diff(X)  # Tính khoảng cách giữa các điểm X (h_i = X_{i+1} - X_i)

    # Khởi tạo ma trận A và vector b
    A = np.zeros((n, n))
    b = np.zeros(n)
```

```
    # Điều kiện biên kẹp (clamped boundary conditions)
    A[0, 0] = 2 * h[0]
    A[n - 1, n - 1] = 2 * h[-1]
    A[0, 1] = h[0]
    A[n - 1, n - 2] = h[-1]
    b[0] = 3 * (Y[1] - Y[0]) / h[0] - 3 * a
    b[n - 1] = 3 * c - 3 * (Y[-1] - Y[-2]) / h[-1]

    # Thiết lập các phương trình cho các điểm nội suy
    for i in range(1, n - 1):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 3 * ((Y[i + 1] - Y[i]) / h[i] - (Y[i] - Y[i - 1]) / h[i
- 1])

    # Giải hệ phương trình tuyến tính A*c = b để tìm các hệ số c
    c = np.linalg.solve(A, b)

    # Tính các hệ số a, b, c, d cho các đoạn spline
    a = Y[:-1]
    b = (Y[1:] - Y[:-1]) / h - h * (2 * c[:-1] + c[1:]) / 3
    d = np.diff(c) / (3 * h)
    c = c[:-1]



    # Tìm đoạn spline chứa x và tính giá trị nội suy tại x
    i = np.searchsorted(X, x) - 1
    i = np.clip(i, 0, n - 1)

    dx = x - X[i]
    y = a[i] + b[i] * dx + c[i] * dx ** 2 + d[i] * dx ** 3

    # Vẽ đồ thị các đoạn spline
    for i in range(0, n - 1):
        x_vals = np.linspace(X[i], X[i + 1], 100)
        y_vals = a[i] + b[i] * (x_vals - X[i]) + c[i] * (x_vals -
X[i])**2 + d[i] * (x_vals - X[i])**3
        plt.plot(x_vals, y_vals, color='orange')

    plt.scatter(X, Y, color='blue')
    plt.title('Clamped Cubic Spline')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()

    # In ra các phương trình spline cho từng đoạn
    for i in range(0, n - 1):
```

```
        print(f'{a[i]} + {b[i]} * (x - {X[i]}) + {c[i]} * (x - {X[i]})**2 + {d[i]} * (x - {X[i]})**3, x thuộc [{X[i]}, {X[i + 1]}]')

    return y

X = np.array([0, 1, 2])
Y = np.array([0, 1, 2])

print('Natural Cubic Spline')
print('f(1.5) = ', Natural_cubic_spline(X, Y, 1.5))

print()
print('Clamped Cubic Spline')
print('f(1.5) = ', Clamped_cubic_spline(X, Y, 1.5, 1, 1))
```

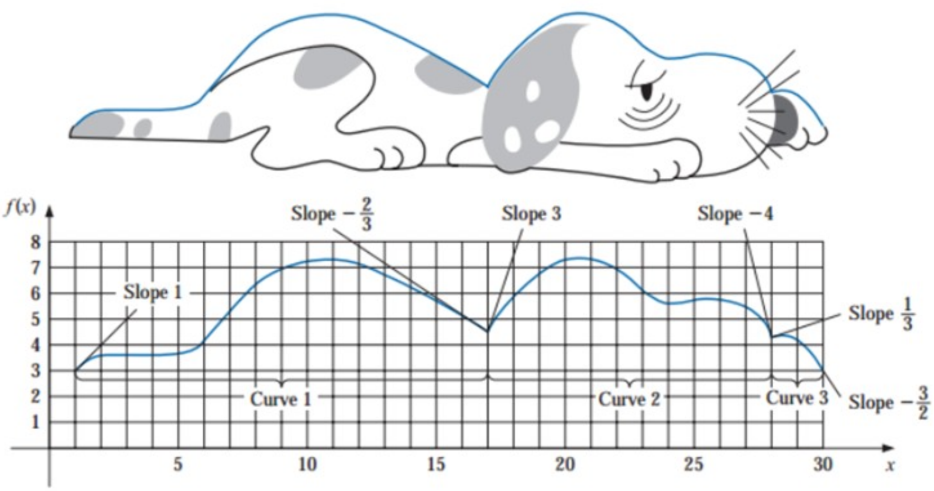
```
Natural Cubic Spline
0 + 1.0 * (x - 0) + 0.0 * (x - 0)**2 + 0.0 * (x - 0)**3, x thuộc [0, 1]
1 + 1.0 * (x - 1) + 0.0 * (x - 1)**2 + 0.0 * (x - 1)**3, x thuộc [1, 2]
f(1.5) =  1.5

Clamped Cubic Spline
```



```
0 + 1.0 * (x - 0) + 0.0 * (x - 0)**2 + 0.0 * (x - 0)**3, x thuộc [0,
1]
1 + 1.0 * (x - 1) + 0.0 * (x - 1)**2 + 0.0 * (x - 1)**3, x thuộc [1,
2]
f(1.5) =  1.5
```

Bài 32

32. The upper portion of this noble beast is to be approximated using clamped cubic spline interpolants. The curve is drawn on a grid from which the table is constructed. Use Algorithm 3.5 to construct the three clamped cubic splines.



Curve 1				Curve 2				Curve 3			
i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$	i	x_i	$f(x_i)$	$f'(x_i)$
0	1	3.0	1.0	0	17	4.5	3.0	0	27.7	4.1	0.33
1	2	3.7		1	20	7.0		1	28	4.3	
2	5	3.9		2	23	6.1		2	29	4.1	
3	6	4.2		3	24	5.6		3	30	3.0	−1.5
4	7	5.7		4	25	5.8					
5	8	6.6		5	27	5.2					
6	10	7.1		6	27.7	4.1	−4.0				
7	13	6.7									
8	17	4.5	−0.67								

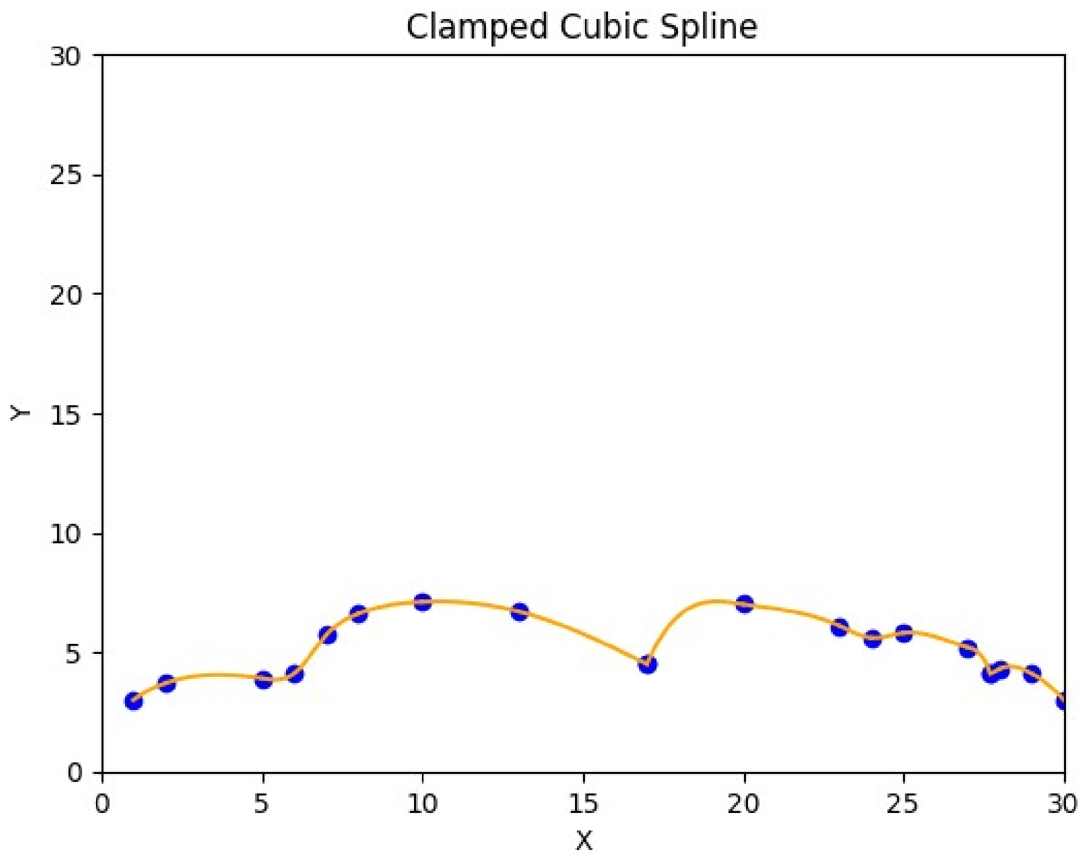
```python
def Clamped_cubic_spline(X, Y, x, a, c):
    # Số lượng điểm dữ liệu
    n = len(X)
    # Tính toán khoảng cách giữa các điểm dữ liệu
    h = np.diff(X)

    # Khởi tạo ma trận và vector cho hệ phương trình tuyến tính
    A = np.zeros((n, n))
    b = np.zeros(n)

    # Điều kiện biên kẹp
    A[0, 0] = 2 * h[0]
    A[n - 1, n - 1] = 2 * h[-1]
```

```
    A[0, 1] = h[0]
    A[n - 1, n - 2] = h[-1]
    b[0] = 3 * (Y[1] - Y[0]) / h[0] - 3 * a
    b[n - 1] = 3 * c - 3 * (Y[-1] - Y[-2]) / h[-1]

    # Tính toán các hệ số c của các đoạn spline
    for i in range(1, n - 1):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 3 * ((Y[i + 1] - Y[i]) / h[i] - (Y[i] - Y[i - 1]) / h[i - 1])
    c = np.linalg.solve(A, b)

    # Tính toán các hệ số a, b, d
    a = Y[:-1]
    b = (Y[1:] - Y[:-1]) / h - h * (2 * c[:-1] + c[1:]) / 3
    d = np.diff(c) / (3 * h)
    c = c[:-1]

    # Tìm đoạn nội suy chứa x và tính giá trị nội suy
    i = np.searchsorted(X, x) - 1
    i = np.clip(i, 0, n - 1)
    dx = x - X[i]
    y = a[i] + b[i] * dx + c[i] * dx ** 2 + d[i] * dx ** 3

    # Vẽ đồ thị các đoạn nội suy
    plt.ylim(0, 30)
    plt.xlim(0, 30)
    for i in range(0, n - 1):
        x_range = np.linspace(X[i], X[i + 1], 100)
        spline_values = a[i] + b[i] * (x_range - X[i]) + c[i] * (x_range - X[i]) ** 2 + d[i] * (x_range - X[i]) ** 3
        plt.plot(x_range, spline_values, color='orange', label='y = a0 + a1 * x')
    plt.scatter(X, Y, color='blue')
    plt.title('Clamped Cubic Spline')
    plt.xlabel('X')
    plt.ylabel('Y')
    # plt.show()
    # In ra các phương trình spline cho từng đoạn
    for i in range(0, n - 1):
        print(f'{a[i]} + {b[i]} * (x - {X[i]}) + {c[i]} * (x - {X[i]})**2 + {d[i]} * (x - {X[i]})**3, x thuộc [{X[i]}, {X[i + 1]}]')
        print()

#Curve 1
X = np.array([1, 2, 5, 6, 7, 8, 10, 13, 17])
a = 1
c = -0.67
```

```python
Y = np.array([3, 3.7, 3.9, 4.1, 5.7, 6.6, 7.1, 6.7, 4.5])

res = Clamped_cubic_spline(X, Y, 2.5, a, c)

#Curve 2
X = np.array([17, 20, 23, 24, 25, 27, 27.7])
Y = np.array([4.5, 7, 6.1, 5.6, 5.8, 5.2, 4.1])
a = 3
c = -4
res = Clamped_cubic_spline(X, Y, 22, a, c)

#Curve 3
X = np.array([27.7, 28, 29, 30])
Y = np.array([4.1, 4.3, 4.1, 3])
a = 0.33
c = -1.5
res = Clamped_cubic_spline(X, Y, 28, a, c)
```

```
3.0 + 1.0 * (x - 1) + -0.3615519340638753 * (x - 1)**2 +
0.06155193406387549 * (x - 1)**3, x thuộc [1, 2]

3.7 + 0.46155193406387596 * (x - 2) + -0.17689613187224884 * (x -
2)**2 + 0.015089236468837468 * (x - 2)**3, x thuộc [2, 5]

3.9 + -0.19241547251100544 * (x - 5) + -0.04109300365271162 * (x -
5)**2 + 0.43350847616371674 * (x - 5)**3, x thuộc [5, 6]

4.1 + 1.0259239486747216 * (x - 6) + 1.2594324248384388 * (x - 6)**2 +
-0.6853563735131599 * (x - 6)**3, x thuộc [6, 7]

5.7 + 1.4887196778121194 * (x - 7) + -0.7966366957010408 * (x - 7)**2
+ 0.20791701788892067 * (x - 7)**3, x thuộc [7, 8]

6.6 + 0.5191973400768 * (x - 8) + -0.17288564203427884 * (x - 8)**2 +
0.01914348599793941 * (x - 8)**3, x thuộc [8, 10]

7.1 + 0.05737660391495769 * (x - 10) + -0.05802472604664237 * (x -
10)**2 + -0.00184841767870708 * (x - 10)**3, x thuộc [10, 13]

6.7 + -0.34067902968998787 * (x - 13) + -0.07466048515500609 * (x -
13)**2 + 0.005582560644375 7605 * (x - 13)**3, x thuộc [13, 17]

4.5 + 3.0 * (x - 17.0) + -1.1007084510629728 * (x - 17.0)**2 +
0.12616207628025017 * (x - 17.0)**3, x thuộc [17.0, 20.0]

7.0 + -0.1978746468110818 * (x - 20.0) + 0.034750235459278814 * (x -
20.0)**2 + -0.022930673285194974 * (x - 20.0)**3, x thuộc [20.0, 23.0]

6.1 + -0.6085014127556733 * (x - 23.0) + -0.17162582410747595 * (x -
23.0)**2 + 0.2801272368631492 * (x - 23.0)**3, x thuộc [23.0, 24.0]
```

```
5.6 + -0.11137135038117751 * (x - 24.0) + 0.6687558864819717 * (x - 24.0)**2 + -0.35738453610079396 * (x - 24.0)**3, x thuộc [24.0, 25.0]

5.8 + 0.1539868142803839 * (x - 25.0) + -0.40339772182041034 * (x - 25.0)**2 + 0.08820215734010924 * (x - 25.0)**3, x thuộc [25.0, 27.0]

5.2 + -0.4011781849199465 * (x - 27.0) + 0.1258152222202451 * (x - 27.0)**2 + -2.568002126658778 * (x - 27.0)**3, x thuộc [27.0, 27.7]

4.1 + 0.33 * (x - 27.7) + 2.262046204620452 * (x - 27.7)**2 + -3.7994132746607767 * (x - 27.7)**3, x thuộc [27.7, 28.0]

4.3 + 0.6613861386138598 * (x - 28.0) + -1.1574257425742553 * (x - 28.0)**2 + 0.29603960396039536 * (x - 28.0)**3, x thuộc [28.0, 29.0]

4.1 + -0.7653465346534649 * (x - 29.0) + -0.26930693069306927 * (x - 29.0)**2 + -0.06534653465346556 * (x - 29.0)**3, x thuộc [29.0, 30.0]
```



#CHƯƠNG 5: XẤP XỈ BÌNH PHƯƠNG TỐI THIỂU

Xấp xỉ bình phương tối thiểu rời rạc

Bài 2

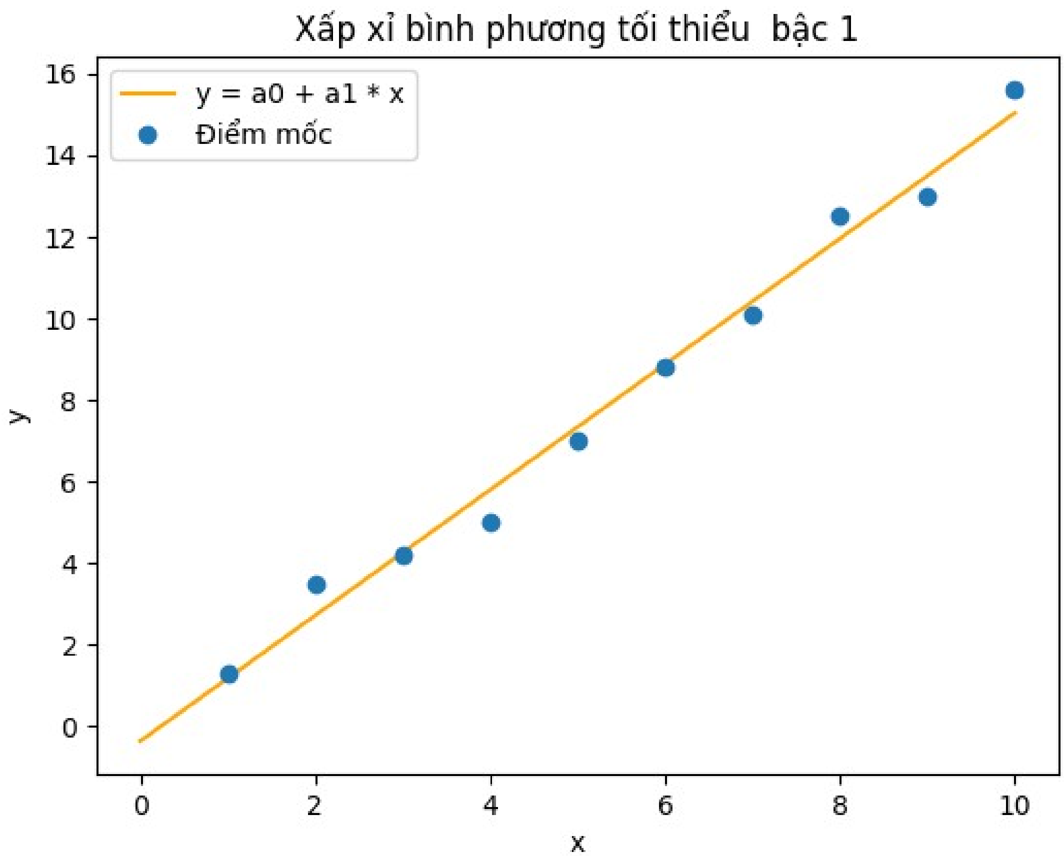
2. Compute the least squares polynomial of degree 2 for the data of Example 1, and compare the total error E for the two polynomials.

Table 8.1

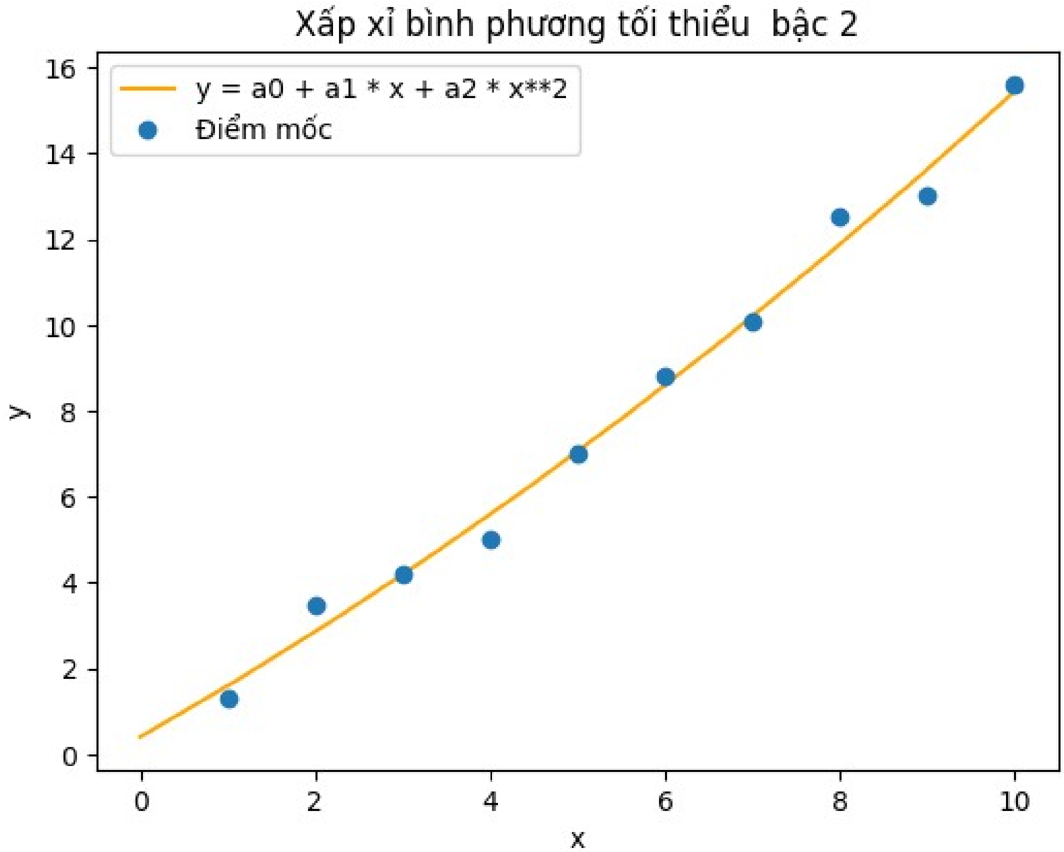
x_i	y_i	x_i	y_i
1	1.3	6	8.8
2	3.5	7	10.1
3	4.2	8	12.5
4	5.0	9	13.0
5	7.0	10	15.6

```python
import numpy as np
import matplotlib.pyplot as plt

# Dữ liệu mẫu
X = np.linspace(1, 10, 10)
Y = np.array([1.3, 3.5, 4.2, 5, 7, 8.8, 10.1, 12.5, 13, 15.6])

def linear_regression(X, Y):
    # Tính các tổng cần thiết cho xấp xỉ tuyến tính
    X_2 = np.sum(X**2)
    X_Y = np.sum(X * Y)
    X_ = np.sum(X)
    Y_ = np.sum(Y)

    # Tính hệ số a0 và a1 của đường thẳng
    a_0 = (X_2 * Y_ - X_Y * X_) / (len(X) * X_2 - X_**2)
    a_1 = (len(X) * X_Y - X_ * Y_) / (len(X) * X_2 - X_**2)
    a = np.array([a_0, a_1])
    return a

def parabol_regression(X, Y):
    # Tạo ma trận A và vector cột B cho hệ phương trình xấp xỉ bậc hai
    A = np.ones([3, 3])
    A[0, 0] = len(X)
    for i in range(0, 3):
```

```
        pow = i
        for j in range(0, 3):
            A[i, j] = np.sum(X**pow)
            pow += 1

    B = np.ones([3, 1])
    for i in range(0, 3):
        B[i, 0] = np.sum(X**i * Y)

    # Giải hệ phương trình để tìm các hệ số
    a = np.linalg.solve(A, B)
    return a

# Vẽ đường xấp xỉ tuyến tính
x = np.linspace(0, X.max(), 10)
a = linear_regression(X, Y)
y = a[0] + a[1] * x
plt.plot(x, y, color='orange', label='y = a0 + a1 * x')
plt.plot(X, Y, 'o', label='Điểm mốc')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xấp xỉ bình phương tối thiểu  bậc 1')
plt.show()
print('Tổng sai số của xấp xỉ bậc nhất: E = ', np.sum(abs(Y - (a[0] +
a[1] * X))))

# Vẽ đường xấp xỉ bậc hai
print()
a = parabol_regression(X, Y)
y = a[0] + a[1] * x + a[2] * x**2
plt.plot(x, y, color='orange', label='y = a0 + a1 * x + a2 * x**2')
plt.plot(X, Y, 'o', label='Điểm mốc')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xấp xỉ bình phương tối thiểu  bậc 2')
plt.show()
print('Tổng sai số của xấp xỉ bậc hai: E = ', np.sum(abs(Y - (a[0] +
a[1] * X + a[2] * X**2))))
```




```
Tổng sai số của xấp xỉ bậc nhất: E =  4.0763636363636335
```




```
Tổng sai số của xấp xỉ bậc hai: E =  3.3066666666666665
```

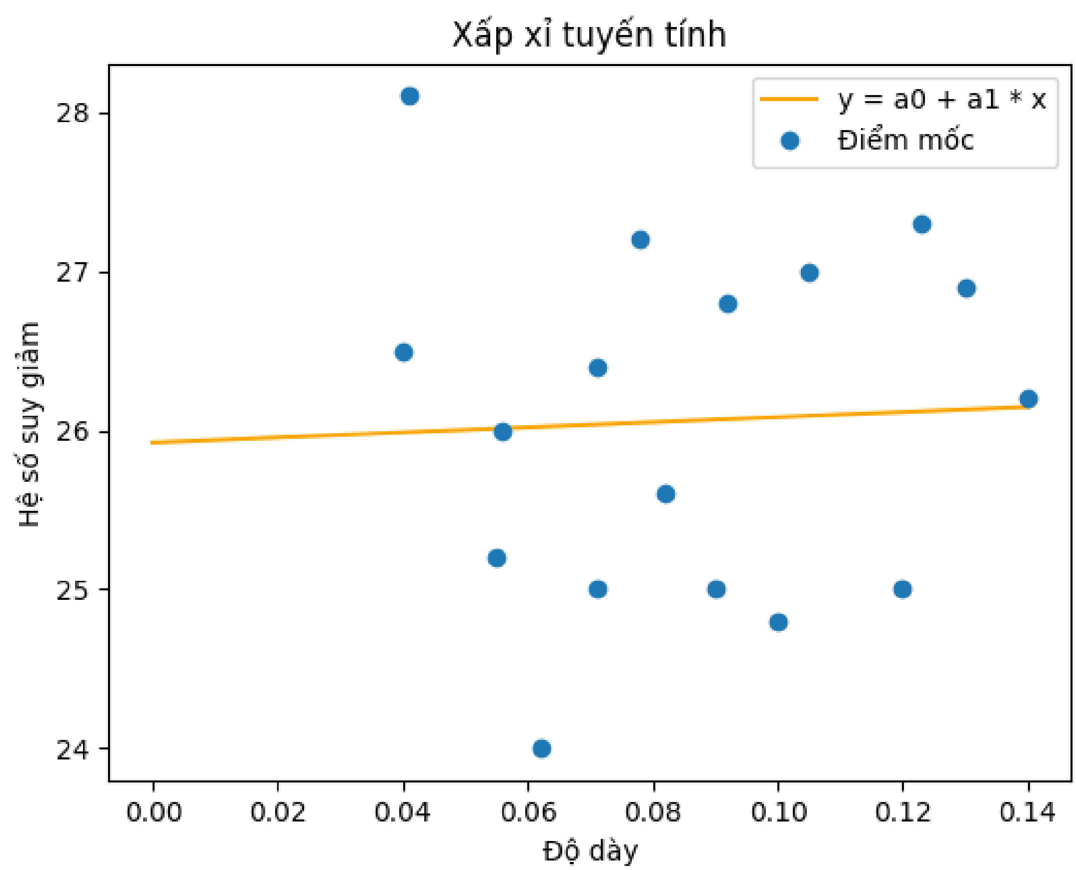
Bài 12

12. To determine a functional relationship between the attenuation coefficient and the thickness of a sample of taconite, V. P. Singh [Si] fits a collection of data by using a linear least squares polynomial. The following collection of data is taken from a graph in that paper. Find the linear least squares polynomial fitting these data.

Thickness (cm)	Attenuation coefficient (dB/cm)
0.040	26.5
0.041	28.1
0.055	25.2
0.056	26.0
0.062	24.0
0.071	25.0
0.071	26.4
0.078	27.2
0.082	25.6
0.090	25.0
0.092	26.8
0.100	24.8
0.105	27.0
0.120	25.0
0.123	27.3
0.130	26.9
0.140	26.2

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

# Dữ liệu mẫu về độ dày và hệ số suy giảm
Thickness = np.array([0.04, 0.041, 0.055, 0.056, 0.062, 0.071, 0.071,
                      0.078, 0.082, 0.09, 0.092, 0.1, 0.105, 0.12,
                      0.123, 0.13, 0.14])
Attenuation_coefficient = np.array([26.5, 28.1, 25.2, 26, 24, 25,
                                    26.4, 27.2, 25.6, 25, 26.8,
                                    24.8, 27, 25, 27.3, 26.9, 26.2])

# Hàm thực hiện xấp xỉ tuyến tính
def linear_regression(X, Y):
    # Tính các tổng cần thiết cho xấp xỉ tuyến tính
    X_2 = np.sum(X**2)
    X_Y = np.sum(X * Y)
    X_ = np.sum(X)
```

```
    Y_ = np.sum(Y)

    # Tính hệ số a0 và a1 của đường thẳng
    a_0 = (X_2 * Y_ - X_Y * X_) / (len(X) * X_2 - X_**2)
    a_1 = (len(X) * X_Y - X_ * Y_) / (len(X) * X_2 - X_**2)
    a = np.array([a_0, a_1])
    return a

# Tạo dải giá trị x để vẽ đường xấp xỉ
x = np.linspace(0, Thickness.max(), 10)

# Thực hiện xấp xỉ tuyến tính
a = linear_regression(Thickness, Attenuation_coefficient)
y = a[0] + a[1] * x

# Vẽ biểu đồ
plt.plot(x, y, color='orange', label='y = a0 + a1 * x')
plt.plot(Thickness, Attenuation_coefficient, 'o', label='Điểm mốc')
plt.legend()
plt.xlabel('Độ dày')
plt.ylabel('Hệ số suy giảm')
plt.title('Xấp xỉ tuyến tính')
plt.show()
```



Xấp xỉ bình phương tối thiểu liên tục và trực giao

Bài 2

2. Find the linear least squares polynomial approximation on the interval $[-1, 1]$ for the fol
functions.

a. $f(x) = x^2 - 2x + 3$ **b.** $f(x) = x^3$

c. $f(x) = \dfrac{1}{x+2}$ **d.** $f(x) = e^x$

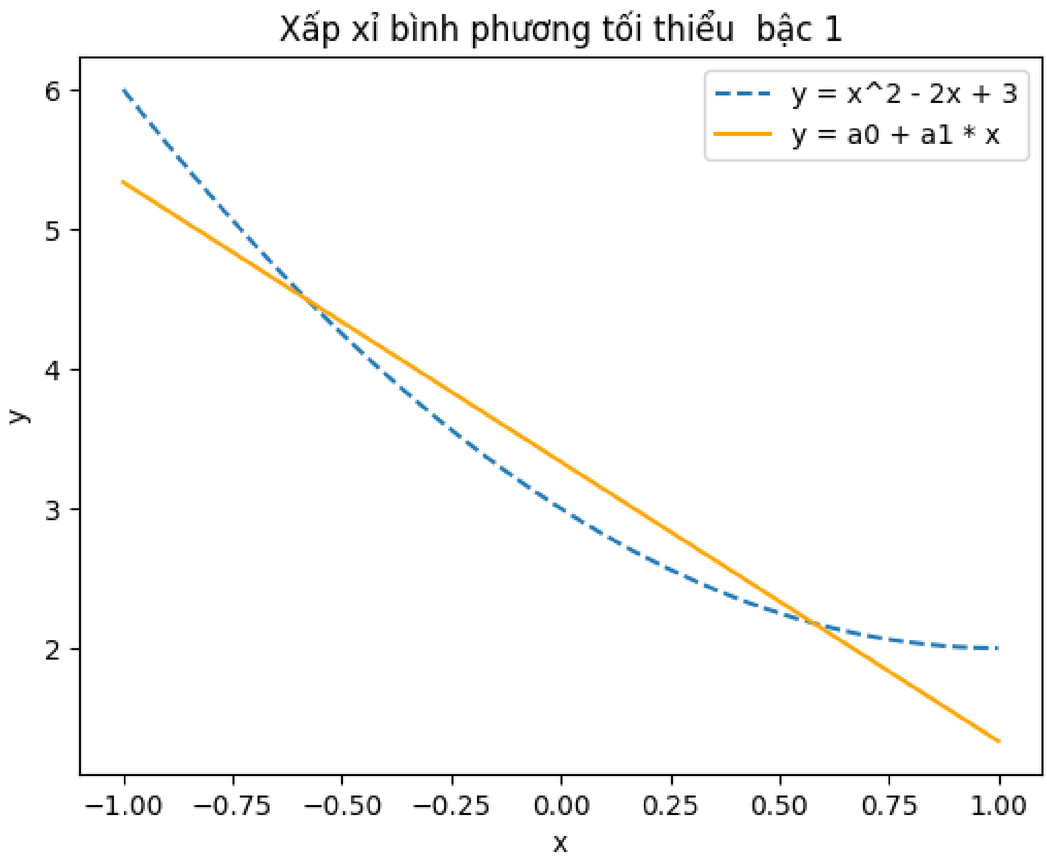
e. $f(x) = \frac{1}{2}\cos x + \frac{1}{3}\sin 2x$ **f.** $f(x) = \ln(x+2)$

```
#[- 1, 1] là khoảng giá trị của x
a = -1
b = 1

# Tạo một mảng X với 1000 điểm từ a đến b
X = np.linspace(a, b, 501)

# Hàm thực hiện xấp xỉ tuyến tính
def linear_regression(X, Y):
    X_2 = np.sum(X** 2)
    X_Y = np.sum(X * Y)
    X_ = np.sum(X)
    Y_ = np.sum(Y)

    a_0 = (X_2 * Y_ - X_Y * X_) / (len(X) * X_2 - X_** 2)
    a_1 = (len(X) * X_Y - X_ * Y_) / (len(X) * X_2 - X_** 2)
    a = np.array([a_0, a_1])
    return a

#Câu a
F = lambda x: x**2 - 2* x + 3
Y = F(X)

# Thực hiện xấp xỉ tuyến tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu a')
plt.plot(X, Y, '--', label='y = x^2 - 2x + 3')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xấp xỉ bình phương tối thiểu  bậc 1')
plt.show()

print()
```

```
#Câu b
F = lambda x: x**3
Y = F(X)

# Thực hiện xấp xỉ tuyến tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu b')
plt.plot(X, Y, '--', label='y = x^3')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xấp xỉ bình phương tối thiểu  bậc 1')
plt.show()

print()

#Câu c
F = lambda x: 1/(x + 2)
Y = F(X)

# Thực hiện xấp xỉ tuyến tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu c')
plt.plot(X, Y, '--', label='y = 1/ (x + 2)')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xấp xỉ bình phương tối thiểu  bậc 1')
plt.show()

print()

#Câu d
F = lambda x: np.exp(x)
Y = F(X)

# Thực hiện xấp xỉ tuyến tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu d')
plt.plot(X, Y, '--', label='y = e^x')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
```

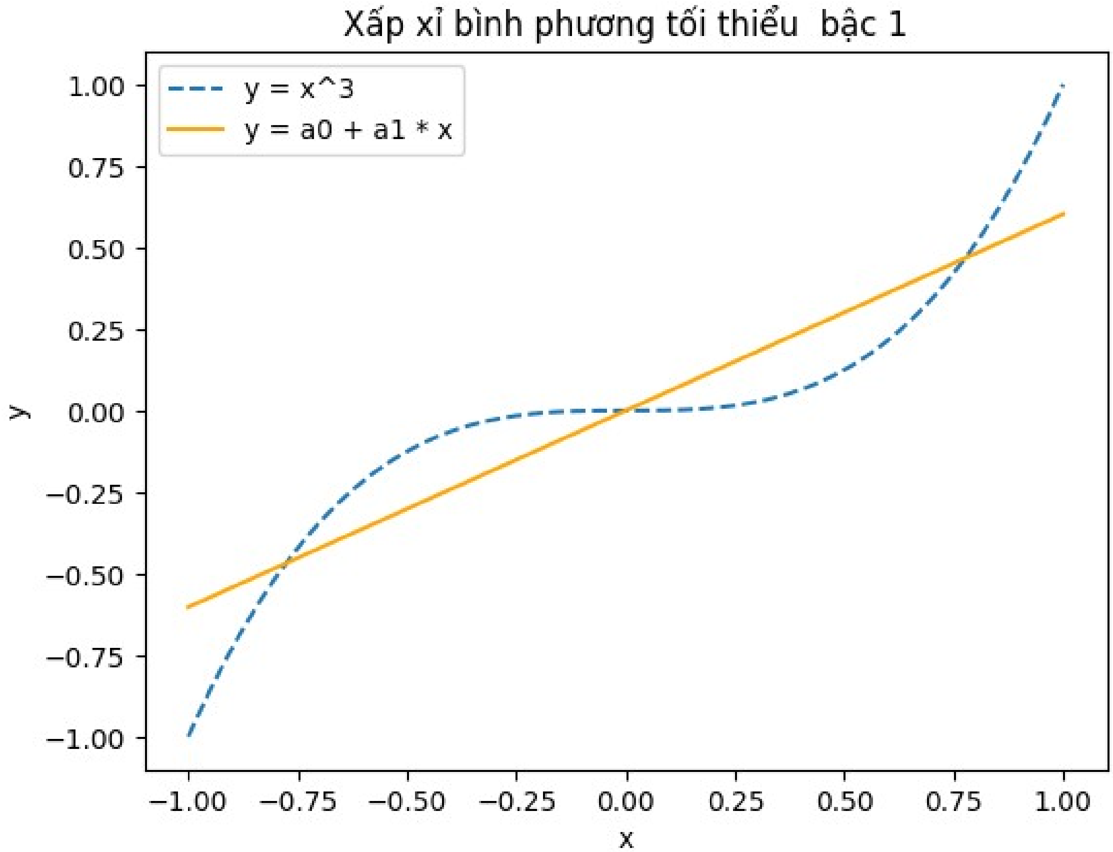
```
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xâṕ xỉ bình phương tôí thiêủ  bậc 1')
plt.show()

print()

#Câu e
F = lambda x: np.cos(x)/ 2 + np.sin(2 * x)/ 3
Y = F(X)

# Thực hiện xâ´p xỉ tuyê´n tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu e')
plt.plot(X, Y, '--', label='y = cosx/ 2 + sin2x / 3')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xâṕ xỉ bình phương tôí thiêủ  bậc 1')
plt.show()

print()

#Câu f
F = lambda x: np.log(x + 2)
Y = F(X)

# Thực hiện xâ´p xỉ tuyê´n tính cho dữ liệu Y
a = linear_regression(X, Y)
y = a[0] + a[1] * X

print('Câu f')
plt.plot(X, Y, '--', label='y = ln(x + 2)')
plt.plot(X, y, color = 'orange', label='y = a0 + a1 * x')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Xâṕ xỉ bình phương tôí thiêủ  bậc 1')
plt.show()

print()
```

Câu a



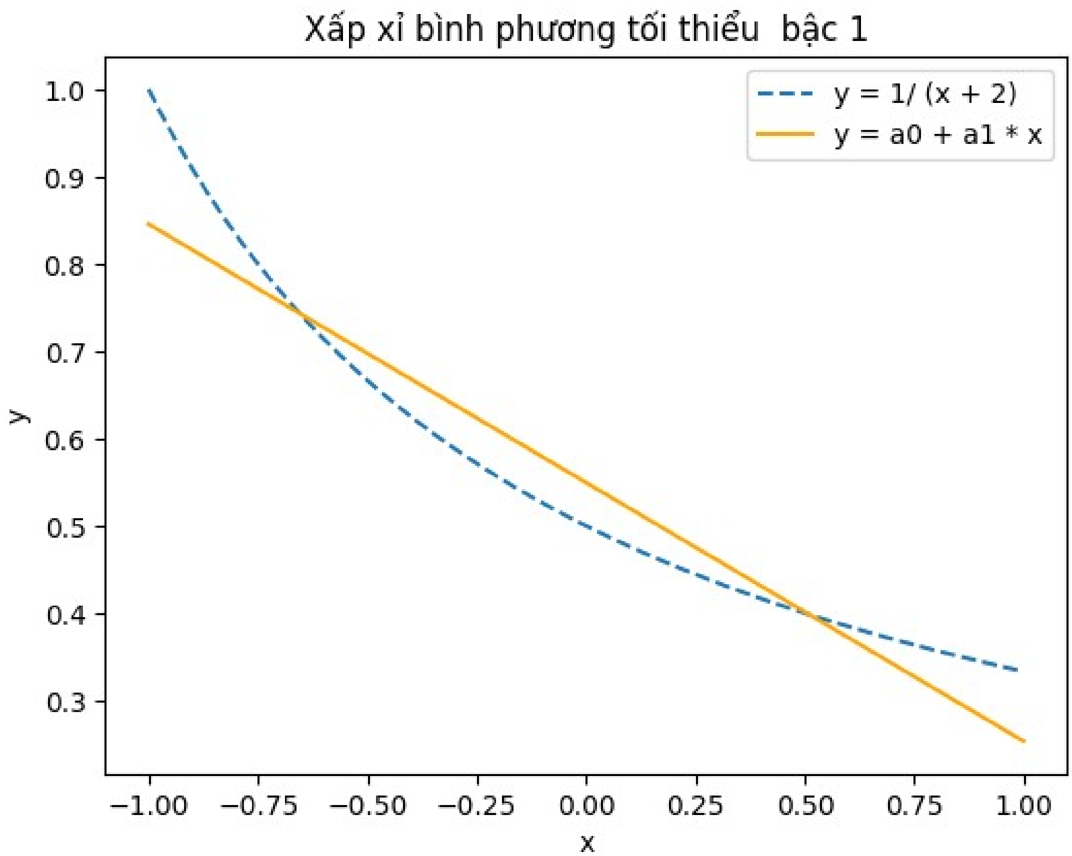
Xấp xỉ bình phương tối thiểu bậc 1
y = x^2 - 2x + 3
y = a0 + a1 * x
6
5
4
3
2
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x

Câu b

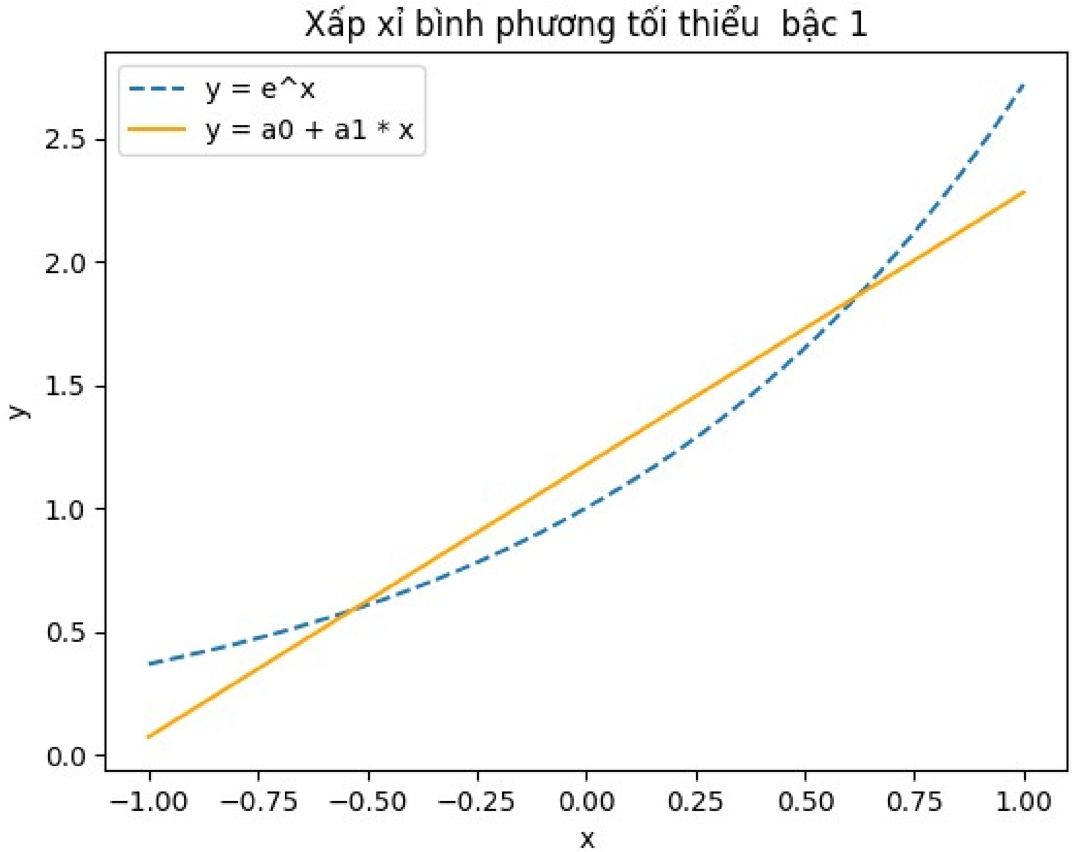


Xấp xỉ bình phương tối thiểu bậc 1
y = x^3
y = a0 + a1 * x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


```
Câu c
```

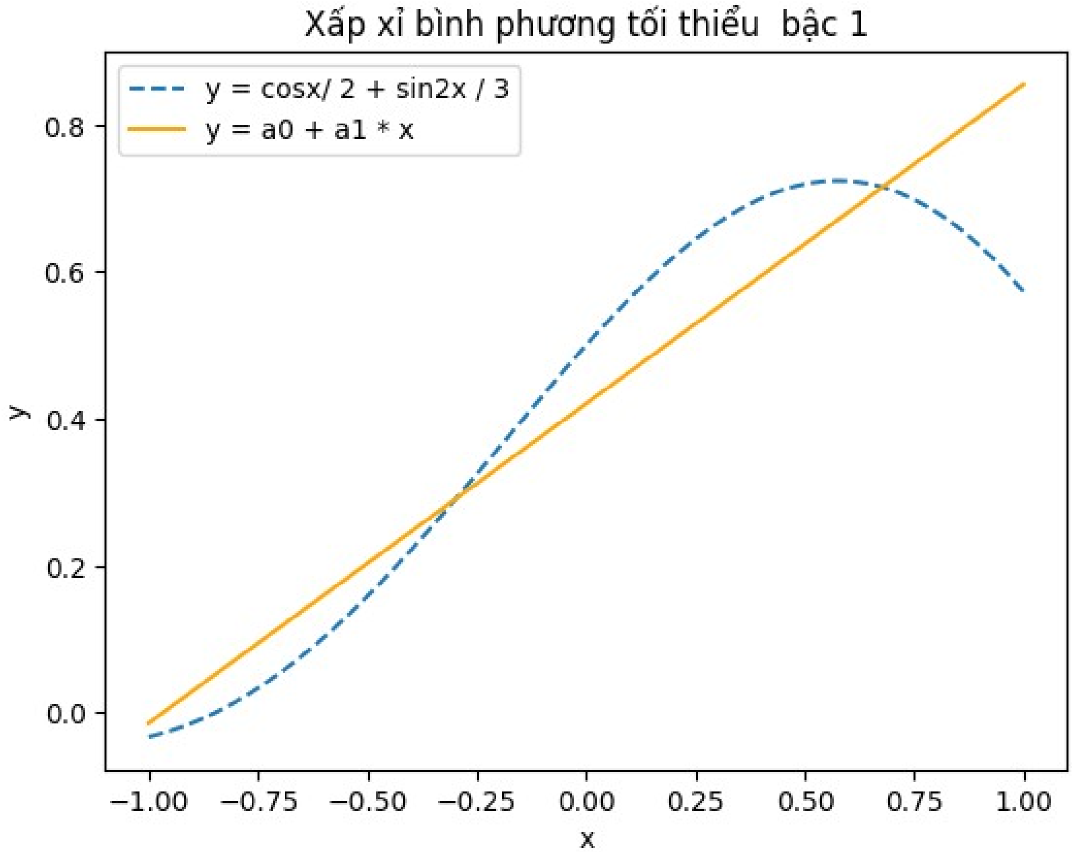

Xấp xỉ bình phương tối thiểu bậc 1
y = 1/ (x + 2)
y = a0 + a1 * x
y
x
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

```
Câu  d
```



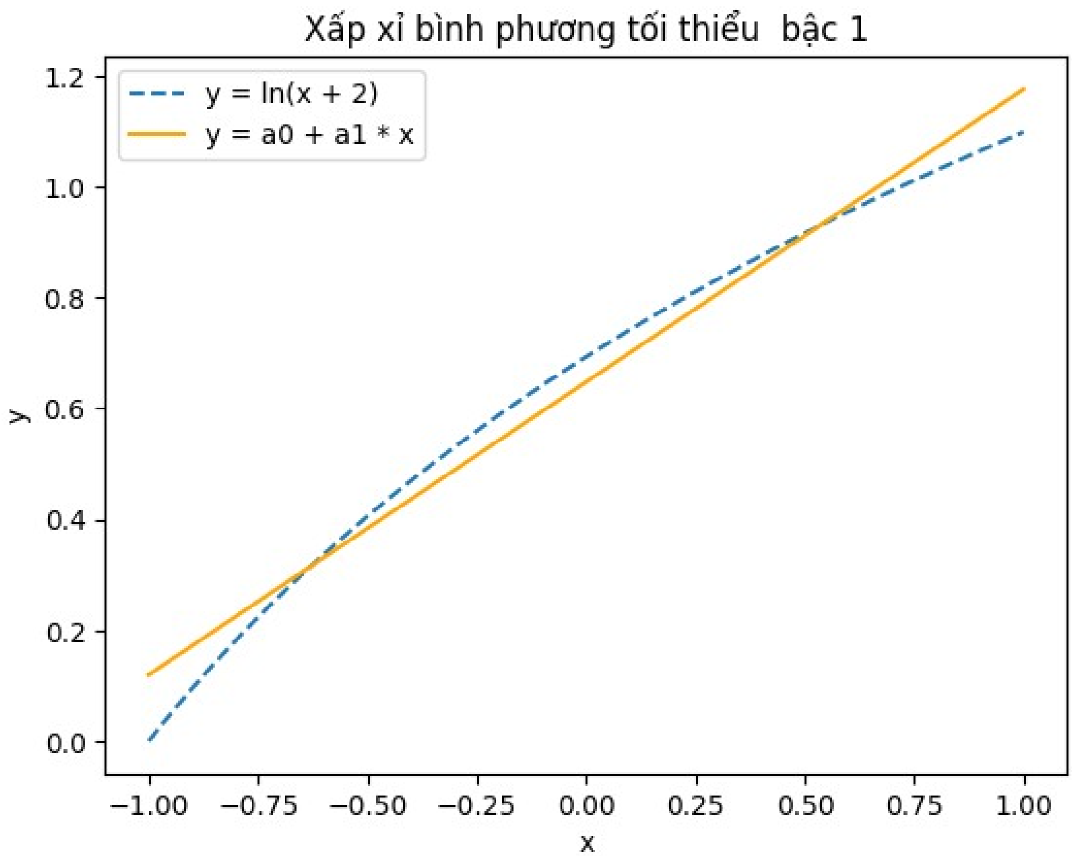
Xấp xỉ bình phương tối thiểu bậc 1
y = e^x
y = a0 + a1 * x
2.5
2.0
1.5
1.0
0.5
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


```
Câu e
```



Xấp xỉ bình phương tối thiểu bậc 1
y = cosx/ 2 + sin2x / 3
y = a0 + a1 * x
0.8
0.6
0.4
0.2
0.0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x


```
Câu f
```



#CHƯƠNG 6: TÍNH GẦN ĐÚNG ĐẠO HÀM VÀ TÍCH PHÂN

Tính gần đúng đạo hàm

Bài 2

2. Use the forward-difference formulas and backward-difference formulas to determine each missing entry in the following tables.

a.

x	$f(x)$	$f'(x)$
−0.3	1.9507	
−0.2	2.0421	
−0.1	2.0601	

b.

x	$f(x)$	$f'(x)$
1.0	1.0000	
1.2	1.2625	
1.4	1.6595	

```python
def forward_and_backward_difference(X, Y):
    """
    Hàm này tính đạo hàm bằng phương pháp sai phân tiến và lùi.

    Parameters:
        X (array-like): Mảng chứa các giá trị x.
        Y (array-like): Mảng chứa các giá trị y tương ứng với các giá
```

```
trị x.

    Returns:
        array: Mảng chứa các giá trị đạo hàm.
    """
    n = len(X)
    F = np.zeros([n, 1])
    for i in range(0, n):
        if i == 0:
            F[i, 0] = (Y[i + 1] - Y[i]) / (X[i + 1] - X[i])  # Phương
pháp sai phân tiến
        elif i == n - 1:
            F[i, 0] = (Y[i] - Y[i - 1]) / (X[i] - X[i - 1])  # Phương
pháp sai phân lùi
        else:
            F[i, 0] = (Y[i + 1] - Y[i]) / (X[i + 1] - X[i])  # Phương
pháp sai phân tiến
    return F

# Câu a
X = np.array([-0.3, -0.2, -0.1])
Y = np.array([1.9507, 2.0421, 2.0601])

dY = forward_and_backward_difference(X, Y)

print("Câu a")
print(tabulate(zip(X, Y, dY), headers=['X', 'F(x)', "F'(x)"],
tablefmt="grid"))

# Câu b
X = np.array([1, 1.2, 1.4])
Y = np.array([1, 1.2625, 1.6595])

dY = forward_and_backward_difference(X, Y)

print()
print("Câu b")
print(tabulate(zip(X, Y, dY), headers=['X', 'F(x)', "F'(x)"],
tablefmt="grid"))
```

```
Câu a
+------+--------+---------+
|    X |   F(x) |   F'(x) |
+======+========+=========+
| -0.3 | 1.9507 |   0.914 |
+------+--------+---------+
| -0.2 | 2.0421 |   0.18  |
+------+--------+---------+
| -0.1 | 2.0601 |   0.18  |
+------+--------+---------+
```

```
Câu b
+-----+--------+---------+
|   X |   F(x) |   F'(x) |
+=====+========+=========+
| 1   | 1      |  1.3125 |
+-----+--------+---------+
| 1.2 | 1.2625 |  1.985  |
+-----+--------+---------+
| 1.4 | 1.6595 |  1.985  |
+-----+--------+---------+
```

Bài 10 + 12

10. Use the formulas given in this section to determine, as accurately as possible, approximations for each missing entry in the following tables.

a.

x	$f(x)$	$f'(x)$
1.05	−1.709847	
1.10	−1.373823	
1.15	−1.119214	
1.20	−0.9160143	
1.25	−0.7470223	
1.30	−0.6015966	

b.

x	$f(x)$	$f'(x)$
−3.0	16.08554	
−2.8	12.64465	
−2.6	9.863738	
−2.4	7.623176	
−2.2	5.825013	
−2.0	4.389056	

12. The data in Exercise 10 were taken from the following functions. Compute the actual errors in Exercise 10, and find error bounds using the error formulas and Maple.

a. $f(x) = \tan 2x$ **b.** $f(x) = e^{-x} - 1 + x$

```python
def five_midpoint_rule(h, F, i):
    """
    Hàm này tính đạo hàm tại điểm i bằng phương pháp tích phân năm điểm trên điểm trung tâm.

    Parameters:
        h (float): Bước x.
        F (array-like): Mảng chứa các giá trị hàm F(x).
        i (int): Chỉ số của điểm cần tính đạo hàm.

    Returns:
        float: Giá trị đạo hàm tại điểm i.
    """
    return (F[i - 2] - 8 * F[i - 1] + 8 * F[i + 1] - F[i + 2]) / (12 * h)

def forward_five_end_point_rule(h, F, i):
    """
    Hàm này tính đạo hàm tại điểm i bằng phương pháp tích phân năm điểm ở điểm đầu.
```

```
    Parameters:
        h (float): Bước x.
        F (array-like): Mảng chứa các giá trị hàm F(x).
        i (int): Chỉ số của điểm cần tính đạo hàm.

    Returns:
        float: Giá trị đạo hàm tại điểm i.
    """
    return (-25 * F[i] + 48 * F[i + 1] - 36 * F[i + 2] + 16 * F[i + 3]
- 3 * F[i + 4]) / (12 * h)

def backward_five_end_point_rule(h, F, i):
    """
    Hàm này tính đạo hàm tại điểm i bằng phương pháp tích phân năm
điểm ở điểm cuối.

    Parameters:
        h (float): Bước x.
        F (array-like): Mảng chứa các giá trị hàm F(x).
        i (int): Chỉ số của điểm cần tính đạo hàm.

    Returns:
        float: Giá trị đạo hàm tại điểm i.
    """
    return (25 * F[i] - 48 * F[i - 1] + 36 * F[i - 2] - 16 * F[i - 3]
+ 3 * F[i - 4]) / (12 * h)

def five_points_rule(X, Y, h):
    """
    Hàm này tính toàn bộ đạo hàm của hàm số F(x) sử dụng các quy
tắc tích phân năm điểm.

    Parameters:
        X (array-like): Mảng chứa các giá trị x.
        Y (array-like): Mảng chứa các giá trị y tương ứng với các giá
trị x.
        h (float): Bước x.

    Returns:
        array: Mảng chứa các giá trị đạo hàm.
    """
    n = len(X)
    F = np.zeros([n, 1])
    for i in range(0, n):
        if i == 0 or i == 1:
            F[i, 0] = forward_five_end_point_rule(h, Y, i)
        elif i == n - 1 or i == n - 2:
            F[i, 0] = backward_five_end_point_rule(h, Y, i)
        else:
```

```
            F[i, 0] = five_midpoint_rule(h, Y, i)
    return F

# Câu a
X = np.array([1.05, 1.1, 1.15, 1.2, 1.25, 1.3])
Y = np.array([-1.709847, -1.373823, -1.119214,
              -0.9160143, -0.7470223, -0.6015966])
F = lambda x: np.tan(2 * x)
dF = lambda x: 2/ np.cos(2 * x)** 2
h = (X[-1] - X[0]) / (len(X) - 1)
dY = five_points_rule(X, Y, h)
dF = np.reshape(dF(X), (len(X), 1))
E = np.abs(dF - dY)
print("Câu a")
print(tabulate(zip(X, Y, dY, E), headers=['X', 'F(x)', "F'(x)",
"Error"], tablefmt="grid"))

# Câu b
X = np.array([-3, -2.8, -2.6, -2.4, -2.2, -2])
Y = np.array([16.08554, 12.64465, 9.863738, 7.623176, 5.825013,
4.389056])
F = lambda x: np.exp(-x) - 1 + x
dF = lambda x: -np.exp(-x) + 1
h = (X[-1] - X[0]) / (len(X) - 1)
dY = five_points_rule(X, Y, h)
dF = np.reshape(dF(X), (len(X), 1))
E = np.abs(dF - dY)
print()
print("Câu b")
print(tabulate(zip(X, Y, dY, E), headers=['X', 'F(x)', "F'(x)",
"Error"], tablefmt="grid"))
```

```
Câu a
+------+-----------+---------+------------+
|    X |      F(x) |   F'(x) |      Error |
+======+===========+=========+============+
| 1.05 | -1.70985  | 7.79869 | 0.0484619  |
+------+-----------+---------+------------+
| 1.1  | -1.37382  | 5.75375 | 0.0210282  |
+------+-----------+---------+------------+
| 1.15 | -1.11921  | 4.49941 | 0.00587019 |
+------+-----------+---------+------------+
| 1.2  | -0.916014 | 3.67551 | 0.00265236 |
+------+-----------+---------+------------+
| 1.25 | -0.747022 | 3.08842 | 0.0276648  |
+------+-----------+---------+------------+
| 1.3  | -0.601597 | 2.71099 | 0.0128443  |
+------+-----------+---------+------------+

Câu b
```

```
+------+----------+-----------+-------------+
|    X |     F(x) |     F'(x) |       Error |
+======+==========+===========+=============+
| -3   | 16.0855  | -19.0809  | 0.00466567  |
+------+----------+-----------+-------------+
| -2.8 | 12.6447  | -15.4409  | 0.0037626   |
+------+----------+-----------+-------------+
| -2.6 |  9.86374 | -12.463   | 0.000710952 |
+------+----------+-----------+-------------+
| -2.4 |  7.62318 | -10.0226  | 0.000590547 |
+------+----------+-----------+-------------+
| -2.2 |  5.82501 |  -8.02097 | 0.00404058  |
+------+----------+-----------+-------------+
| -2   |  4.38906 |  -6.38573 | 0.0033286   |
+------+----------+-----------+-------------+
```

Tính gần đúng tích phân

Bài 2

2. Approximate the following integrals using the Trapezoidal rule.

a. $\int_{-0.25}^{0.25} (\cos x)^2 \, dx$ **b.** $\int_{-0.5}^{0} x \ln(x+1) \, dx$

c. $\int_{0.75}^{1.3} \left((\sin x)^2 - 2x \sin x + 1\right) \, dx$ **d.** $\int_{e}^{e+1} \frac{1}{x \ln x} \, dx$

```python
# Hàm trapezoidal_rule tính xấp xỉ tích phân của hàm số F(x) trong khoảng [a, b] bằng phương pháp hình thang.
# Bước lặp h được tính là (b - a)/(2 - 1).
# Giá trị xấp xỉ được tính bằng công thức (F(a) + F(b)) / 2 * h.

def trapezoidal_rule(a, b, F):
  h = (b - a)/ (2 - 1)
  return (F(a) + F(b)) / 2 * h

# Hàm simpson_rule tính xấp xỉ tích phân của hàm số F(x) trong khoảng [a, b] bằng phương pháp Simpson 1/3.
# Bước lặp h được tính là (b - a)/(3 - 1).
# Giá trị xấp xỉ được tính bằng công thức (F(a) + 4 * F(a + h) + F(b)) * h / 3.

def simpson_rule(a, b, F):
  h = (b - a)/ (3 - 1)
  return (F(a) + 4 * F(a + h) + F(b)) * h / 3

# Hàm midpoint_rule tính xấp xỉ tích phân của hàm số F(x) trong khoảng [a, b] bằng phương pháp trung điểm.
# Bước lặp h được tính là (b - a)/(2 - 1).
```

```
# Điểm trung điểm được tính là (a + b)/ 2.
# Giá trị xấp xỉ được tính bằng công thức h * F(mid).

def midpoint_rule(a, b, F):
  h = (b - a)/ (2 - 1)
  mid = (a + b)/ 2
  return h * F(mid)

# Hàm composite_trapezoidal_rule tính xấp xỉ tích phân của hàm số
F(x) trong khoảng [a, b] bằng phương pháp hình thang với phân chia
thành n đoạn.
# Bước lặp h được tính là (b - a)/ n.
# Tổng các giá trị xấp xỉ trên các đoạn được tính bằng cách lặp qua
từng đoạn và áp dụng trapezoidal_rule.

def composite_trapezoidal_rule(a, b, F):
  n = 1000
  h = (b - a)/ n
  sum = 0
  while (a < b):
    sum += trapezoidal_rule(a, a + h, F)
    a += h
  return sum

# Hàm composite_simpson_rule tính xấp xỉ tích phân của hàm số F(x)
trong khoảng [a, b] bằng phương pháp Simpson 1/3 với phân chia thành
n đoạn.
# Bước lặp h được tính là (b - a)/ n.
# Tổng các giá trị xấp xỉ trên các đoạn được tính bằng cách lặp qua
từng đoạn và áp dụng simpson_rule.

def composite_simpson_rule(a, b, F):
  n = 1000
  h = (b - a)/ n
  sum = 0
  while (a < b):
    sum += simpson_rule(a, a + h, F)
    a += h
  return sum

# Hàm composite_midpoint_rule tính xấp xỉ tích phân của hàm số
F(x) trong khoảng [a, b] bằng phương pháp trung điểm với phân chia
thành n đoạn.
# Bước lặp h được tính là (b - a)/ n.
# Tổng các giá trị xấp xỉ trên các đoạn được tính bằng cách lặp qua
từng đoạn và áp dụng midpoint_rule.

def composite_midpoint_rule(a, b, F):
  n = 1000
  h = (b - a)/ n
```

```
  sum = 0
  while (a < b):
    sum += midpoint_rule(a, a + h, F)
    a += h
  return sum

# Câu a
a = -0.25
b = 0.25
F = lambda x: np.cos(x)** 2
res = composite_trapezoidal_rule(a, b, F)
print("Câu a")
print('Giá trị xấp xỉ với Composite Trapezoidal Rule = ', res)
print('Sai số', np.abs(res -0.4897127693))

# Câu b
a = -0.5
b = 0
F = lambda x: x * np.log(x + 1)
res = composite_trapezoidal_rule(a, b, F)
print()
print('Câu b')
print('Giá trị xấp xỉ với Composite Trapezoidal Rule = ', res)
print('Sai số', np.abs(res - 0.05256980729))

# Câu c
a = 0.75
b = 1.3
F = lambda x: np.sin(x)**2 - 2 * x * np.sin(x) + 1
res = composite_trapezoidal_rule(a, b, F)
print()
print('Câu c')
print('Giá trị xấp xỉ với Composite Trapezoidal Rule = ', res)
print('Sai số', np.abs(res - -0.02037679598))

# Câu d
a = np.e
b = np.e + 1
F = lambda x: 1/ (x * np.log(x))
res = composite_trapezoidal_rule(a, b, F)
print()
print('Câu d')
print('Giá trị xấp xỉ với Trapezoidal Rule = ', res)
print('Sai số', abs(res - 0.2725138802))

Câu a
Giá trị xấp xỉ với Composite Trapezoidal Rule =  0.4897127493260378
Sai số 1.9973962206432105e-08

Câu b
```

```
Giá trị xấp xỉ với Composite Trapezoidal Rule =  0.052569842563918716
Sai số 3.52739187164941e-08

Câu c
Giá trị xấp xỉ với Composite Trapezoidal Rule =  -0.02037681220793073
Sai số 1.6227930728363038e-08

Câu d
Giá trị xấp xỉ với Trapezoidal Rule =  0.2727186353904608
Sai số 0.00020475519046O8147
```

Bài 12

12. Repeat Exercise 4 using the Midpoint rule and the results of Exercise 10.

2. Approximate the following integrals using the Trapezoidal rule.

a. $\int_{-0.25}^{0.25} (\cos x)^2 \, dx$ **b.** $\int_{-0.5}^{0} x \ln(x+1) \, dx$

c. $\int_{0.75}^{1.3} \left((\sin x)^2 - 2x \sin x + 1\right) \, dx$ **d.** $\int_{e}^{e+1} \frac{1}{x \ln x} \, dx$

3. Find a bound for the error in Exercise 1 using the error formula, and compare this to the actua

4. Find a bound for the error in Exercise 2 using the error formula, and compare this to the actua

5. Repeat Exercise 1 using Simpson's rule.

6. Repeat Exercise 2 using Simpson's rule.

7. Repeat Exercise 3 using Simpson's rule and the results of Exercise 5.

8. Repeat Exercise 4 using Simpson's rule and the results of Exercise 6.

9. Repeat Exercise 1 using the Midpoint rule.

10. Repeat Exercise 2 using the Midpoint rule.

```python
#Cau a
a = -0.25
b = 0.25
F = lambda x: np.cos(x)** 2
res = midpoint_rule(a, b, F)
print("Câu a")
print('Giá trị xấp xỉ với Midpoint Rule = ', res)

#Cau b
a = -0.5
b = 0
F = lambda x: x * np.log(x + 1)
res = midpoint_rule(a, b, F)
print()
print('Cau b')
print('Giá trị xấp xỉ với Midpoint Rule = ', res)

#Cau c
```

```
a = 0.75
b = 1.3
F = lambda x: np.sin(x)**2 - 2 * x * np.sin(x) + 1
res = midpoint_rule(a, b, F)
print()
print('Cau c')
print('Giá trị xấp xỉ với Midpoint Rule = ', res)

#Cau d
a = np.e
b = np.e + 1
F = lambda x: 1/ (x * np.log(x))
res = midpoint_rule(a, b, F)
print()
print('Cau d')
print('Giá trị xấp xỉ với Midpoint Rule = ', res)

Câu a
Giá trị xấp xỉ với Midpoint Rule =  0.5

Cau b
Giá trị xấp xỉ với Midpoint Rule =  0.03596025905647261

Cau c
Giá trị xấp xỉ với Midpoint Rule =  -0.011895258503444106

Cau d
Giá trị xấp xỉ với Midpoint Rule =  0.26583859242827784
```

Bài 22

22. Given the function f at the following values,

x	1.8	2.0	2.2	2.4	2.6
$f(x)$	3.12014	4.42569	6.04241	8.03014	10.46675

approximate $\int_{1.8}^{2.6} f(x)\,dx$ using all the appropriate quadrature formulas of this section.

```
X = np.array([1.8, 2, 2.2, 2.4, 2.6])
Y = np.array([3.12014, 4.42569, 6.04241, 8.03014, 10.46675])
def trapezoidal(X, Y):
  h = X[1] - X[0]
  n = len(X)
  sum = 0
  for i in range (0, n - 1):
    sum += (Y[i] + Y[i + 1]) / 2 * h
  return sum
```

```python
def simpson(X, Y):
  h = X[1] - X[0]
  n = len(X)
  sum = 0
  for i in range (0,  n - 2, 2):
    sum += (Y[i] + 4 * Y[i + 1] + Y[i + 2]) * h / 3
  return sum

def midpoint(X, Y):
  h = (X[1] - X[0]) * 2
  sum = 0
  for i in range (1, 4, 2):
    sum += Y[i] * h
  return sum

print('Giá trị xấp xỉ với Trapezoidal Rule = ', trapezoidal(X, Y))
print()
print('Giá trị xấp xỉ với Simpson Rule = ', simpson(X, Y))
print()
print('Giá trị xấp xỉ với Midpoint Rule = ', midpoint(X, Y))
```

```
Giá trị xấp xỉ với Trapezoidal Rule =  5.058336999999999

Giá trị xấp xỉ với Simpson Rule =  5.033002

Giá trị xấp xỉ với Midpoint Rule =  4.982331999999999
```

#CHƯƠNG 7: GIẢI PTVP VÀ HPTVP

Giải PTVP bậc nhất

```
```

Giải PTVP bậc cao - Taylor bậc 2

Bài 2

2. Use Taylor's method of order two to approximate the solutions for each of the following initia problems.

a. $y' = e^{t-y}$, $0 \le t \le 1$, $y(0) = 1$, with $h = 0.5$

b. $y' = \dfrac{1+t}{1+y}$, $1 \le t \le 2$, $y(1) = 2$, with $h = 0.5$

c. $y' = -y + ty^{1/2}$, $2 \le t \le 3$, $y(2) = 2$, with $h = 0.25$

d. $y' = t^{-2}(\sin 2t - 2ty)$, $1 \le t \le 2$, $y(1) = 2$, with $h = 0.25$

```python
import sympy as sp

def Second_taylor_series(x0, y0, dy_dx, h, ilterations):
  x_ = [x0]
```

```
  y_ = [y0]

  # Khai báo các biến và hàm
  x = sp.symbols('x')
  y = sp.Function('y')(x)

  # Định nghĩa dx/dt (trong trường hợp này là dy/dx)
  # dy_dx = 1 - 2*y**2 - x
  dy_dx = dy_dx.simplify()
  # Tính đạo hàm bậc hai của y theo x
  d2y_dx2 = sp.diff(dy_dx, x)

  # Thay đạo hàm bậc nhất của y bằng dy/dx
  d2y_dx2 = d2y_dx2.subs(sp.Derivative(y, x), dy_dx)

  # Đơn giản hóa kết quả
  d2y_dx2 = d2y_dx2.simplify()
  for i in range(ilterations):
    res = y0 + h * dy_dx.subs({x: x0, y: y0}).evalf() + h** 2*
d2y_dx2.subs({x: x0, y: y0}).evalf()/ 2
    x_new = x0 + h
    x_.append(x_new)
    y_.append(res)
    x0 = x_new
    y0 = res
  return x_, y_

#Cau a
x0 = 0
y0 = 1
a = 0
b = 1
h = 0.5
ilterations = int((b - a)/ h)
x = sp.symbols('x')
y = sp.Function('y')(x)
dy = sp.exp(x - y)

X, Y = Second_taylor_series(x0, y0, dy, h, ilterations)
print('Câu a')
print(tabulate(zip(X, Y), headers = ['x', 'y'], tablefmt="grid"))

#Cau b
x0 = 1
y0 = 2
a = 1
b = 2
h = 0.5
ilterations = int((b - a)/ h)
x = sp.symbols('x')
```

```python
y = sp.Function('y')(x)
dy = (1 + x)/ (1 + y)

X, Y = Second_taylor_series(x0, y0, dy, h, ilterations)
print()
print("Câu b")
print(tabulate(zip(X, Y), headers = ['x', 'y'], tablefmt="grid"))

#Cau c
x0 = 2
y0 = 2
a = 2
b = 3
h = 0.25
ilterations = int((b - a)/ h)
x = sp.symbols('x')
y = sp.Function('y')(x)
dy = - y + x * y**(1/ 2)

X, Y = Second_taylor_series(x0, y0, dy, h, ilterations)
print()
print("Câu c")
print(tabulate(zip(X, Y), headers = ['x', 'y'], tablefmt="grid"))

#Cau d
x0 = 1
y0 = 2
a = 1
b = 2
h = 0.25
ilterations = int((b - a)/ h)
x = sp.symbols('x')
y = sp.Function('y')(x)
dy = x**(-2) * (sp.sin(2 * x) - 2 * x * y)

X, Y = Second_taylor_series(x0, y0, dy, h, ilterations)
print()
print("Câu d")
print(tabulate(zip(X, Y), headers = ['x', 'y'], tablefmt="grid"))
```

```
Câu a
+-----+---------+
|   x |       y |
+=====+=========+
| 0   | 1       |
+-----+---------+
| 0.5 | 1.21301 |
+-----+---------+
| 1   | 1.48933 |
+-----+---------+
```

```
Câu b
+-----+---------+
|   x |       y |
+=====+=========+
| 1   | 2       |
+-----+---------+
| 1.5 | 2.35648 |
+-----+---------+
| 2   | 2.74548 |
+-----+---------+

Câu c
+------+---------+
|    x |       y |
+======+=========+
| 2    | 2       |
+------+---------+
| 2.25 | 2.24372 |
+------+---------+
| 2.5  | 2.56341 |
+------+---------+
| 2.75 | 2.96339 |
+------+---------+
| 3    | 3.4487  |
+------+---------+

Câu d
+------+----------+
|    x |        y |
+======+==========+
| 1    | 2        |
+------+----------+
| 1.25 | 1.46265  |
+------+----------+
| 1.5  | 1.07852  |
+------+----------+
| 1.75 | 0.791842 |
+------+----------+
| 2    | 0.574516 |
+------+----------+
```

Bài 12

12. Use the Taylor method of order two with $h = 0.1$ to approximate the solution to

$$y' = 1 + t\sin(ty), \quad 0 \le t \le 2, \quad y(0) = 0.$$

```
a = 0
b = 2
h = 0.1
x0 = 0
y0 = 0
ilterations = int((b - a)/ h)
x = sp.symbols('x')
y = sp.Function('y')(x)
dy = 1 + x * sp.sin(x * y)

X, Y = Second_taylor_series(x0, y0, dy, h, ilterations)

print(tabulate(zip(X, Y), headers = ['x', 'y'], tablefmt="grid"))
```

```
+-----+----------+
|   x |        y |
+=====+==========+
| 0   | 0        |
+-----+----------+
| 0.1 | 0.1      |
+-----+----------+
| 0.2 | 0.20025  |
+-----+----------+
| 0.3 | 0.301653 |
+-----+----------+
| 0.4 | 0.405727 |
+-----+----------+
| 0.5 | 0.514639 |
+-----+----------+
| 0.6 | 0.631243 |
+-----+----------+
| 0.7 | 0.75908  |
+-----+----------+
| 0.8 | 0.902234 |
+-----+----------+
| 0.9 | 1.06478  |
+-----+----------+
| 1   | 1.2493   |
+-----+----------+
| 1.1 | 1.45398  |
+-----+----------+
| 1.2 | 1.66834  |
+-----+----------+
| 1.3 | 1.87142  |
+-----+----------+
| 1.4 | 2.03821  |
+-----+----------+
| 1.5 | 2.1526   |
+-----+----------+
| 1.6 | 2.21326  |
```

```
+-----+----------+
| 1.7 | 2.2283   |
+-----+----------+
| 1.8 | 2.20804  |
+-----+----------+
| 1.9 | 2.16141  |
+-----+----------+
| 2   | 2.09518  |
+-----+----------+
```

PP Euler

Bài 2

2. Use Euler's method to approximate the solutions for each of the following initial-value problems.

a. $y' = e^{t-y}, \quad 0 \le t \le 1, \quad y(0) = 1$, with $h = 0.5$

b. $y' = \dfrac{1+t}{1+y}, \quad 1 \le t \le 2, \quad y(1) = 2$, with $h = 0.5$

c. $y' = -y + ty^{1/2}, \quad 2 \le t \le 3, \quad y(2) = 2$, with $h = 0.25$

d. $y' = t^{-2}(\sin 2t - 2ty), \quad 1 \le t \le 2, \quad y(1) = 2$, with $h = 0.25$

```python
def first_order_taylor(x0, y0, h, Fxy, ilterations):
  x_ = [x0]
  y_ = [y0]
  for i in range(ilterations):
    y_new = y0 + h * Fxy(x0, y0)
    x_new = x0 + h
    x_.append(x_new)
    y_.append(y_new)
    x0 = x_new
    y0 = y_new
  return x_, y_

#Cau a
x0 = 0
y0 = 1
h = 0.5
a = 0
b = 1
n = int((b - a)/ h)

X = np.linspace(0, 1, n + 1)
dy = lambda x, y: np.exp(x - y)

stt, res = first_order_taylor(x0, y0, h, dy, n)
print("Câu a")
print(tabulate(zip(stt, res), headers = ['x', 'y'], tablefmt="grid"))
```

```
#Cau b
x0 = 1
y0 = 2
h = 0.5
a = 1
b = 2
n = int((b - a)/ h)
x = np.linspace(1, 2, n + 1)
dy = lambda x, y: (1 + x)/ (1 + y)

stt, res = first_order_taylor(x0, y0, h, dy, n)
print()
print("Câu b")
print(tabulate(zip(stt, res), headers = ['x', 'y'], tablefmt="grid"))

#Cau c
x0 = 2
y0 = 3
h = 0.25
a = 2
b = 3
n = int((b - a)/ h)
x = np.linspace(2, 3, n + 1)
dy = lambda x, y: - y + x * y**(1/ 2)

stt, res = first_order_taylor(x0, y0, h, dy, n)
print()
print("Câu c")
print(tabulate(zip(stt, res), headers = ['x', 'y'], tablefmt="grid"))

#Cau d
x0 = 1
y0 = 2
h = 0.25
a = 1
b = 2
n = int((b - a)/ h)
x = np.linspace(1, 2, n)
dy = lambda x, y: x**(-2) * (np.sin(2 * x) - 2 * x * y)

stt, res = first_order_taylor(x0, y0, h, dy, n)
print()
print("Câu d")
print(tabulate(zip(stt, res), headers = ['x', 'y'], tablefmt="grid"))
```

```
Câu a
+-----+---------+
|   x |       y |
+=====+=========+
```

```
| 0   | 1       |
+-----+---------+
| 0.5 | 1.18394 |
+-----+---------+
| 1   | 1.43625 |
+-----+---------+

Câu b
+-----+---------+
|   x |       y |
+=====+=========+
| 1   | 2       |
+-----+---------+
| 1.5 | 2.33333 |
+-----+---------+
| 2   | 2.70833 |
+-----+---------+

Câu c
+------+---------+
|    x |       y |
+======+=========+
| 2    | 3       |
+------+---------+
| 2.25 | 3.11603 |
+------+---------+
| 2.5  | 3.32996 |
+------+---------+
| 2.75 | 3.63798 |
+------+---------+
| 3    | 4.03979 |
+------+---------+

Câu d
+------+----------+
|    x |        y |
+======+==========+
| 1    | 2        |
+------+----------+
| 1.25 | 1.22732  |
+------+----------+
| 1.5  | 0.83215  |
+------+----------+
| 1.75 | 0.570447 |
+------+----------+
| 2    | 0.378827 |
+------+----------+
```

Bài 12

12. Consider the initial-value problem

$$y' = -10y, \quad 0 \le t \le 2, \quad y(0) = 1,$$

which has solution $y(t) = e^{-10t}$. What happens when Euler's method is applied to this problem with $h = 0.1$? Does this behavior violate Theorem 5.9?

```
def first_order_taylor(x0, y0, h, Fxy, ilterations):
  x_ = [x0]
  y_ = [y0]
  for i in range(ilterations):
    y_new = y0 + h * Fxy(x0, y0)
    x_new = x0 + h
    x_.append(x_new)
    y_.append(y_new)
    x0 = x_new
    y0 = y_new
  return x_, y_

dy = lambda x, y: -10 * y
x0 = 0
y0 = 1
h = 0.1
a = 0
b = 2
n = int((b - a)/ h)
X = np.linspace(a, b, n + 1)
Y = lambda x: np.e**(-10 * x)
x, y = first_order_taylor(x0, y0, h, dy, n)
print(tabulate(zip(x, y, Y(X)), headers = ['x', 'y', 'Exact values'],
tablefmt="grid"))
```

```
+-----+-----+----------------+
|   x |   y |   Exact values |
+=====+=====+================+
| 0   |   1 |    1           |
+-----+-----+----------------+
| 0.1 |   0 |    0.367879    |
+-----+-----+----------------+
| 0.2 |   0 |    0.135335    |
+-----+-----+----------------+
| 0.3 |   0 |    0.0497871   |
+-----+-----+----------------+
| 0.4 |   0 |    0.0183156   |
+-----+-----+----------------+
| 0.5 |   0 |    0.00673795  |
+-----+-----+----------------+
| 0.6 |   0 |    0.00247875  |
+-----+-----+----------------+
| 0.7 |   0 |    0.000911882 |
```

```
+-----+-----+----------------+
| 0.8 |   0 |    0.000335463 |
+-----+-----+----------------+
| 0.9 |   0 |    0.00012341  |
+-----+-----+----------------+
| 1   |   0 |    4.53999e-05 |
+-----+-----+----------------+
| 1.1 |   0 |    1.67017e-05 |
+-----+-----+----------------+
| 1.2 |   0 |    6.14421e-06 |
+-----+-----+----------------+
| 1.3 |   0 |    2.26033e-06 |
+-----+-----+----------------+
| 1.4 |   0 |    8.31529e-07 |
+-----+-----+----------------+
| 1.5 |   0 |    3.05902e-07 |
+-----+-----+----------------+
| 1.6 |   0 |    1.12535e-07 |
+-----+-----+----------------+
| 1.7 |   0 |    4.13994e-08 |
+-----+-----+----------------+
| 1.8 |   0 |    1.523e-08   |
+-----+-----+----------------+
| 1.9 |   0 |    5.6028e-09  |
+-----+-----+----------------+
| 2   |   0 |    2.06115e-09 |
+-----+-----+----------------+
```

PP Midpoint và hiệu chỉnh Heun

Euler cải tiến

Bài 2

2. Use the Modified Euler method to approximate the solutions to each of the following problems, and compare the results to the actual values.

a. $y' = e^{t-y}$, $0 \le t \le 1$, $y(0) = 1$, with $h = 0.5$; actual solution $y(t) = \ln(e^t +$

b. $y' = \dfrac{1+t}{1+y}$, $1 \le t \le 2$, $y(1) = 2$, with $h = 0.5$; actual solution $y(t) = \sqrt{t^2}$

c. $y' = -y + ty^{1/2}$, $2 \le t \le 3$, $y(2) = 2$, with $h = 0.25$; actual solu $\left(t - 2 + \sqrt{2}ee^{-t/2}\right)^2$.

d. $y' = t^{-2}(\sin 2t - 2ty)$, $1 \le t \le 2$, $y(1) = 2$, with $h = 0.25$; actual sol $\frac{1}{2}t^{-2}(4 + \cos 2 - \cos 2t)$.

```
# Hàm modified_euler tính giá trị xấp xỉ của phương trình vi phân bậc nhất y' = Fxy(x, y) sử dụng phương pháp Euler cải tiến
```

```
(Modified Euler method).
# Đầu vào:
# - x0: Giá trị ban đầu của biến độc lập x.
# - y0: Giá trị ban đầu của hàm số y tại x = x0.
# - h: Bước xấp xỉ.
# - Fxy: Hàm số Fxy(x, y) đại diện cho đạo hàm của hàm y theo biến x.
# - iterations: Số lần lặp để tính giá trị xấp xỉ.
# Đầu ra:
# - x_: Danh sách chứa các giá trị x tương ứng.
# - y_: Danh sách chứa các giá trị xấp xỉ của hàm số y tương ứng.
def modified_euler(x0, y0, h, Fxy, iterations):
  x_ = [x0]  # Khởi tạo danh sách chứa giá trị x với giá trị ban đầu x0.
  y_ = [y0]  # Khởi tạo danh sách chứa giá trị xấp xỉ y với giá trị ban đầu y0.
  for i in range(iterations):
    predictor = y0 + h * Fxy(x0, y0)  # Bước dự đoán: sử dụng Euler method để dự đoán giá trị y tại x + h.
    y_new = y0 + h * (Fxy(x0 + h, predictor) + Fxy(x0, y0)) / 2  # Tính giá trị xấp xỉ mới sử dụng trung bình có trọng số của đạo hàm tại x và x + h.
    x_new = x0 + h  # Tăng giá trị của x lên một bước xấp xỉ h.
    x_.append(x_new)  # Thêm giá trị mới của x vào danh sách xấp xỉ.
    y_.append(y_new)  # Thêm giá trị xấp xỉ mới của y vào danh sách xấp xỉ.
    x0 = x_new  # Cập nhật giá trị x0 cho lần lặp tiếp theo.
    y0 = y_new  # Cập nhật giá trị y0 cho lần lặp tiếp theo.
  return x_, y_  # Trả về danh sách các giá trị x và y tương ứng.


#Cau a
x0 = 0
y0 = 1
h = 0.5
a = 0
b = 1
n = int((b - a)/ h)
X = np.linspace(0, 1, n + 1)
dy = lambda x, y: np.exp(x - y)
real_y = lambda x: np.log(np.e** x + np.e - 1)
Y = real_y(X)

stt, res = modified_euler(x0, y0, h, dy, n)
E = np.abs(Y - res)
print("Câu a")
print(tabulate(zip(stt, res, Y, E), headers = ['x', 'y', 'Exact y', 'Error'], tablefmt="grid"))
```

```
#Cau b
x0 = 1
y0 = 2
h = 0.5
X = np.linspace(1, 2, 3)
dy = lambda x, y: (1 + x)/ (1 + y)
real_y = lambda x: np.sqrt(x**2 + 2* x + 6) - 1
Y = real_y(X)

stt, res = modified_euler(x0, y0, h, dy, 3 - 1)
E = np.abs(Y - res)
print()
print("Câu b")
print(tabulate(zip(stt, res, Y, E), headers = ['x', 'y', 'Exact y',
'Error'], tablefmt="grid"))

#Cau c
x0 = 2
y0 = 2
h = 0.25
X = np.linspace(2, 3, 5)
dy = lambda x, y: - y + x * y**(1/ 2)
real_y = lambda x: (x - 2 + np.sqrt(2) * np.e** (1 - x / 2))** 2
Y = real_y(X)

stt, res = modified_euler(x0, y0, h, dy, 5 - 1)
E = np.abs(Y - res)
print()
print("Câu c")
print(tabulate(zip(stt, res, Y, E), headers = ['x', 'y', 'Exact y',
'Error'], tablefmt="grid"))

#Cau d
x0 = 1
y0 = 2
h = 0.25
X = np.linspace(1, 2, 5)
dy = lambda x, y: x**(-2) * (np.sin(2 * x) - 2 * x * y)
real_y = lambda x: 1/ 2 * x**(-2)* (4 + np.cos(2) - np.cos(2 * x))
Y = real_y(X)

stt, res = modified_euler(x0, y0, h, dy, 5 - 1)
E = np.abs(Y - res)
print()
print("Câu d")
print(tabulate(zip(stt, res, Y, E), headers = ['x', 'y', 'Exact y',
'Error'], tablefmt="grid"))
```

```
Câu a
+-----+---------+----------+------------+
```

x	y	Exact y	Error
0	1	1	0
0.5	1.21813	1.21402	0.00410305
1	1.49755	1.48988	0.00767438

Câu b

x	y	Exact y	Error
1	2	2	0
1.5	2.35417	2.3541	6.47004e-05
2	2.74175	2.74166	8.7696e-05

Câu c

x	y	Exact y	Error
2	2	2	4.44089e-16
2.25	2.2455	2.24412	0.00137834
2.5	2.56716	2.56445	0.00270402
2.75	2.96919	2.96519	0.00400068
3	3.45657	3.45129	0.00528177

Câu d

x	y	Exact y	Error
1	2	2	0
1.25	1.41608	1.4032	0.0128761
1.5	1.03101	1.01641	0.0146009
1.75	0.752267	0.73801	0.0142569
2	0.543245	0.529687	0.0135579

PP Runge Kutta bậc 4 giải htvp

Bài 2

2. Use the Runge-Kutta method for systems to approximate the solutions of the following systems of first-order differential equations, and compare the results to the actual solutions.

a. $u_1' = u_1 - u_2 + 2, \quad u_1(0) = -1;$
$u_2' = -u_1 + u_2 + 4t, \quad u_2(0) = 0; \quad 0 \le t \le 1; \quad h = 0.1;$
actual solutions $u_1(t) = -\frac{1}{2}e^{2t} + t^2 + 2t - \frac{1}{2}$ and $u_2(t) = \frac{1}{2}e^{2t} + t^2 - \frac{1}{2}$.

b. $u_1' = \frac{1}{9}u_1 - \frac{2}{3}u_2 - \frac{1}{9}t^2 + \frac{2}{3}, \quad u_1(0) = -3;$
$u_2' = u_2 + 3t - 4, \quad u_2(0) = 5; \quad 0 \le t \le 2; \quad h = 0.2;$
actual solutions $u_1(t) = -3e^t + t^2$ and $u_2(t) = 4e^t - 3t + 1$.

c. $u_1' = u_1 + 2u_2 - 2u_3 + e^{-t}, \quad u_1(0) = 3;$
$u_2' = u_2 + u_3 - 2e^{-t}, \quad u_2(0) = -1;$
$u_3' = u_1 + 2u_2 + e^{-t}, \quad u_3(0) = 1; \quad 0 \le t \le 1; \quad h = 0.1;$
actual solutions $u_1(t) = -3e^{-t} - 3\sin t + 6\cos t$, $u_2(t) = \frac{3}{2}e^{-t} + \frac{3}{10}\sin t - \frac{21}{10}\cos t - \frac{2}{5}e^{2t}$,
and $u_3(t) = -e^{-t} + \frac{12}{5}\cos t + \frac{9}{5}\sin t - \frac{2}{5}e^{2t}$.

d. $u_1' = 3u_1 + 2u_2 - u_3 - 1 - 3t - 2\sin t, \quad u_1(0) = 5;$
$u_2' = u_1 - 2u_2 + 3u_3 + 6 - t + 2\sin t + \cos t, \quad u_2(0) = -9;$
$u_3' = 2u_1 + 4u_3 + 8 - 2t, \quad u_3(0) = -5; \quad 0 \le t \le 2; \quad h = 0.2;$
actual solutions $u_1(t) = 2e^{3t} + 3e^{-2t} + 1$, $u_2(t) = -8e^{-2t} + e^{4t} - 2e^{3t} + \sin t$, and $u_3(t) = 2e^{4t} - 4e^{3t} - e^{-2t} - 2$.

```python
def Runge_kutta_4(x0, y0, y1, a, b, h, F1, F2, ilterations):
  x = x0
  z1 = y0
  z2 = y1
  K1 = np.zeros([1, 2])
  K2 = np.zeros([1, 2])
  K3 = np.zeros([1, 2])
  K4 = np.zeros([1, 2])
  y_ = [y0]
  dy_ = [y1]
  x_  = [x0]
  for i in range(ilterations):
    K1[0, 0] = h * F1(x, z1, z2)
    K1[0, 1] = h * F2(x, z1, z2)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
    x = x + h
    y_.append(z1)
```

```
    dy_.append(z2)
    x_.append(x)
  return x_, y_, dy_

# Đầu vào:
# - x0: Giá trị ban đầu của biến độc lập x.
# - y0: Giá trị ban đầu của hàm số y1 tại x = x0.
# - y1: Giá trị ban đầu của hàm số y2 tại x = x0.
# - y2: Giá trị ban đầu của hàm số y3 tại x = x0.
# - a: Giá trị ban đầu của khoảng cách x.
# - b: Giá trị kết thúc của khoảng cách x.
# - h: Bước xấp xỉ.
# - F1: Hàm số F1(x, y1, y2, y3) đại diện cho đạo hàm của hàm y1
theo biến x.
# - F2: Hàm số F2(x, y1, y2, y3) đại diện cho đạo hàm của hàm y2
theo biến x.
# - F3: Hàm số F3(x, y1, y2, y3) đại diện cho đạo hàm của hàm y3
theo biến x.
# - iterations: Số lần lặp để tính giá trị xấp xỉ.
# Đầu ra:
# - x_: Danh sách chứa các giá trị x tương ứng.
# - r1: Danh sách chứa các giá trị xấp xỉ của hàm số y1 tương ứng.
# - r2: Danh sách chứa các giá trị xấp xỉ của hàm số y2 tương ứng.
# - r3: Danh sách chứa các giá trị xấp xỉ của hàm số y3 tương ứng.
def Runge_kutta_4_3_an(x0, y0, y1, y2, a, b, h, F1, F2, F3,
iterations):
  x = x0
  z1 = y0
  z2 = y1
  z3 = y2
  K1 = np.zeros([1, 3])
  K2 = np.zeros([1, 3])
  K3 = np.zeros([1, 3])
  K4 = np.zeros([1, 3])
  r1 = [y0]
  r2 = [y1]
  r3 = [y2]
  x_  = [x0]
  for i in range(iterations):
    K1[0, 0] = h * F1(x, z1, z2, z3)
    K1[0, 1] = h * F2(x, z1, z2, z3)
    K1[0, 2] = h * F3(x, z1, z2, z3)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2, z3
+ K1[0, 2]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2, z3
+ K1[0, 2]/ 2)
    K2[0, 2] = h * F3(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2, z3
+ K1[0, 2]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2, z3
```

```
+ K2[0, 2]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2, z3
+ K2[0, 2]/ 2)
    K3[0, 2] = h * F3(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2, z3
+ K2[0, 2]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1], z3 + K3[0,
2])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1], z3 + K3[0,
2])
    K4[0, 2] = h * F3(x + h, z1 + K3[0, 0], z2 + K3[0, 1], z3 + K3[0,
2])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
    z3 = z3 + (K1[0, 2] + 2 * K2[0, 2] + 2 * K3[0, 2] + K4[0, 2]) / 6
    x = x + h
    r1.append(z1)
    r2.append(z2)
    r3.append(z3)
    x_.append(x)
  return x_, r1, r2, r3


#Cau a
dF1 = lambda t, u1, u2: u1 - u2 + 2
dF2 = lambda t, u1, u2: -u1 + u2 + 4 * t
x0 = 0
u1_x0 = -1
u2_x0 = 0
h = 0.1
a = 0
b = 1
n = int((b - a)/ h)
x = np.linspace(a, b, n + 1)
U1 = lambda x: -1/2 * np.e**(2*x) + x** 2 + 2 * x - 1/ 2
U2 = lambda x: 1/2 * np.e**(2*x) + x** 2 - 1/ 2

t, u1_t, u2_t = Runge_kutta_4(x0, u1_x0, u2_x0, a, b, h, dF1, dF2, n)
E1 = np.abs(U1(x) - u1_t)
E2 = np.abs(U2(x) - u2_t)
print('Câu a')
print(tabulate(zip(t, u1_t, U1(x), E1, u2_t, U2(x), E2), headers =
['t', 'u1(t)', 'u1(exact)', 'E1', 'u2(t)', 'u2(exact)', 'E2'],
tablefmt="grid"))

print()
#Cau b
dF1 = lambda t, u1, u2: 1/9*u1-2/3*u2-1/9*t** 2+2/3
dF2 = lambda t, u1, u2: u2+3*t - 4
x0 = 0
u1_x0 = -3
```

```
u2_x0 = 5
h = 0.2
a = 0
b = 2
n = int((b - a)/ h)
x = np.linspace(a, b, n + 1)
U1 = lambda x: -3*np.e**(x) + x**2
U2 = lambda x: 4*np.e**x - 3*x + 1

t, u1_t, u2_t = Runge_kutta_4(x0, u1_x0, u2_x0, a, b, h, dF1, dF2, n)
E1 = np.abs(U1(x) - u1_t)
E2 = np.abs(U2(x) - u2_t)
print('Câu b')
print(tabulate(zip(t, u1_t, U1(x), E1, u2_t, U2(x), E2), headers =
['t', 'u1(t)', 'u1(exact)', 'E1', 'u2(t)', 'u2(exact)', 'E2'],
tablefmt="grid"))

print()
#Cau c
dF1 = lambda t, u1, u2, u3: u1 + 2*u2 - 2*u3+ np.e**(-t)
dF2 = lambda t, u1, u2, u3: u2+u3-2*np.e**(-t)
dF3 = lambda t, u1, u2, u3: u1 + 2 * u2 + np.e**(-t)
x0 = 0
u1_x0 = 3
u2_x0 = -1
u3_x0 = 1
h = 0.1
a = 0
b = 1
n = int((b - a)/ h)
x = np.linspace(a, b, n + 1)
U1 = lambda x: -3*np.e**(-x) - 3*np.sin(x) + 6 *np.cos(x)
U2 = lambda x: 3/ 2*np.e**(-x) + 3/ 10 * np.sin(x) - 21/ 10*np.cos(x)
- 2/5*np.e**(2*x)
U3 = lambda x: -np.e**(-x) + 12/5*np.cos(x)+9/5*np.sin(x)-2/5*np.e**(2
* x)

t, u1_t, u2_t, u3_t = Runge_kutta_4_3_an(x0, u1_x0, u2_x0, u3_x0, a,
b, h, dF1, dF2, dF3, n)
E1 = np.abs(U1(x) - u1_t)
E2 = np.abs(U2(x) - u2_t)
E3 = np.abs(U3(x) - u3_t)
print('Câu c')
print(tabulate(zip(t, u1_t, U1(x), E1, u2_t, U2(x), E2, u3_t, U3(x),
E3), headers = ['t', 'u1(t)', 'u1(exact)', 'E1', 'u2(t)', 'u2(exact)',
'E2', 'u3(t)', 'u3(exact)', 'E3'], tablefmt="grid"))

print()
#Cau d
dF1 = lambda t, u1, u2, u3: 3*u1 + 2*u2-u3-1-3*t-2*np.sin(t)
```

```
dF2 = lambda t, u1, u2, u3: u1 - 2 * u2 + 3 * u3 + 6 - t + 2 *
np.sin(t)+np.cos(t)
dF3 = lambda t, u1, u2, u3: 2*u1+4*u3+8-2*t
x0 = 0
u1_x0 = 5
u2_x0 = -9
u3_x0 = -5
h = 0.2
a = 0
b = 2
n = int((b - a)/ h)
x = np.linspace(a, b, n + 1)
U1 = lambda x: 2*np.e**(3*x)+3*np.e**(-2*x) + 1
U2 = lambda x: -8*np.e**(-2*x)+np.e**(4*x)-2 *np.e**(3*x)+np.sin(x)
U3 = lambda x: 2*np.e**(4*x)-4*np.e**(3*x)-np.e**(-2*x)-2

t, u1_t, u2_t, u3_t = Runge_kutta_4_3_an(x0, u1_x0, u2_x0, u3_x0, a,
b, h, dF1, dF2, dF3, n)
E1 = np.abs(U1(x) - u1_t)
E2 = np.abs(U2(x) - u2_t)
E3 = np.abs(U3(x) - u3_t)
print('Câu d')
print(tabulate(zip(t, u1_t, U1(x), E1, u2_t, U2(x), E2, u3_t, U3(x),
E3), headers = ['t', 'u1(t)', 'u1(exact)', 'E1', 'u2(t)', 'u2(exact)',
'E2', 'u3(t)', 'u3(exact)', 'E3'], tablefmt="grid"))
```

```
Câu a
+-----+-----------+------------+------------+---------
+------------+------------+
|   t |     u1(t) |   u1(exact) |          E1 |    u2(t) |   u2(exact)
|          E2 |
+=====+===========+============+============+=========+===========
=+============+
| 0   | -1        |   -1        | 0           | 0        |    0
| 0           |
+-----+-----------+------------+------------+---------
+------------+------------+
| 0.1 | -0.9007   |   -0.900701 | 1.37908e-06 | 0.1207   |    0.120701
| 1.37908e-06 |
+-----+-----------+------------+------------+---------
+------------+------------+
| 0.2 | -0.805909 |   -0.805912 | 3.36882e-06 | 0.285909 |    0.285912
| 3.36882e-06 |
+-----+-----------+------------+------------+---------
+------------+------------+
| 0.3 | -0.721053 |   -0.721059 | 6.17202e-06 | 0.501053 |    0.501059
| 6.17202e-06 |
+-----+-----------+------------+------------+---------
+------------+------------+
| 0.4 | -0.65276  |   -0.65277  | 1.00514e-05 | 0.77276  |    0.77277
```

```
| 1.00514e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.5 | -0.609126 |   -0.609141 | 1.53459e-05 | 1.10913  |    1.10914
| 1.53459e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.6 | -0.600036 |   -0.600058 | 2.24922e-05 | 1.52004  |    1.52006
| 2.24922e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.7 | -0.637568 |   -0.6376   | 3.20507e-05 | 2.01757  |    2.0176
| 3.20507e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.8 | -0.736471 |   -0.736516 | 4.47392e-05 | 2.61647  |    2.61652
| 4.47392e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.9 | -0.914762 |   -0.914824 | 6.14751e-05 | 3.33476  |    3.33482
| 6.14751e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 1   | -1.19444  |   -1.19453  | 8.34286e-05 | 4.19444  |    4.19453
| 8.34286e-05 |
+-----+----------+------------+------------+---------
+------------+------------+

Câu b
+-----+----------+------------+------------+---------
+------------+------------+
|   t |     u1(t) |   u1(exact) |          E1 |    u2(t) |   u2(exact)
|            E2 |
+=====+==========+============+============+=========+===========
=+============+
| 0   |  -3       |    -3       | 0           |  5       |     5
| 0           |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.2 |  -3.6242  |    -3.62421 | 8.26534e-06 |  5.2856  |     5.28561
| 1.10326e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.4 |  -4.31545 |    -4.31547 | 2.01944e-05 |  5.76727 |     5.7673
| 2.69506e-05 |
+-----+----------+------------+------------+---------
+------------+------------+
| 0.6 |  -5.10632 |    -5.10636 | 3.70041e-05 |  6.48843 |     6.48848
| 4.93762e-05 |
```

```
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 0.8 |  -6.03656 |    -6.03662 | 6.02703e-05 |  7.50208 |     7.50216
| 8.04109e-05 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 1   |  -7.15475 |    -7.15485 | 9.20277e-05 |  8.873   |     8.87313
| 0.000122767 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 1.2 |  -8.52022 |    -8.52035 | 0.000134895 | 10.6803  |    10.6805
| 0.000179938 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 1.4 | -10.2054  |   -10.2056  | 0.000192236 | 13.0205  |    13.0208
| 0.000256406 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 1.6 | -12.2988  |   -12.2991  | 0.000268356 | 16.0118  |    16.0121
| 0.000357914 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 1.8 | -14.9086  |   -14.9089  | 0.00036876  | 19.7981  |    19.7986
| 0.000491801 |
+-----+-----------+-------------+-------------+----------
+------------+------------+
| 2   | -18.1667  |   -18.1672  | 0.000500471 | 24.5556  |    24.5562
| 0.000667429 |
+-----+-----------+-------------+-------------+----------
+------------+------------+

Câu c
+-----+-----------+-------------+-------------+----------
+------------+------------+------------+-------------+-------------+
|   t |     u1(t) |   u1(exact) |          E1 |    u2(t) |   u2(exact)
|          E2 |      u3(t) |   u3(exact) |          E3 |
+=====+===========+=============+=============+==========+============
=+=============+============+============+=============+
| 0   |  3        |    3        | 0           | -1       |    -1
| 0           |  1         |   1         | 1.11022e-16 |
+-----+-----------+-------------+-------------+----------
+------------+------------+------------+-------------+-------------+
| 0.1 |  2.95601  |    2.95601  | 3.3812e-07  | -1.19086 |    -1.19086
| 6.86542e-07 |  1.17431   |   1.17431   | 4.48526e-07 |
+-----+-----------+-------------+-------------+----------
+------------+------------+------------+-------------+-------------+
| 0.2 |  2.8282   |    2.8282   | 7.96431e-07 | -1.36717 |    -1.36717
| 1.76584e-06 |  1.29431   |   1.2943    | 1.3981e-06  |
+-----+-----------+-------------+-------------+----------
```

```
+------------+------------+-----------+------------+------------+
| 0.3 |  2.62301  |    2.623    | 1.36237e-06 | -1.53517 |    -1.53517
| 3.38789e-06 |  1.35508   |   1.35508   | 2.99908e-06 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.4 |  2.34715  |    2.34715  | 2.02159e-06 | -1.70213 |    -1.70214
| 5.74835e-06 |  1.35097   |   1.35096   | 5.44592e-06 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.5 |  2.00763  |    2.00763  | 2.75815e-06 | -1.8766  |    -1.87661
| 9.10109e-06 |  1.27533   |   1.27532   | 8.98968e-06 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.6 |  1.61165  |    1.61165  | 3.55471e-06 | -2.06863 |    -2.06864
| 1.37741e-05 |  1.12032   |   1.1203    | 1.39539e-05 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.7 |  1.16665  |    1.16664  | 4.39281e-06 | -2.29009 |    -2.29011
| 2.01896e-05 |  0.876569  |   0.876548  | 2.07548e-05 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.8 |  0.68019  |    0.680185 | 5.25309e-06 | -2.55507 |    -2.5551
| 2.88896e-05 |  0.532825  |   0.532795  | 2.99269e-05 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 0.9 |  0.159976 |    0.15997  | 6.11557e-06 | -2.88035 |    -2.88039
| 4.05681e-05 |  0.0754659 |   0.0754237 | 4.21552e-05 |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+
| 1   | -0.38623  |   -0.386237 | 6.95996e-06 | -3.28594 |    -3.286
| 5.61117e-05 | -0.51207   |  -0.512129  | 5.8316e-05  |
+-----+----------+------------+------------+---------
+------------+------------+-----------+------------+------------+

Câu d
+-----+----------+------------+----------+-----------
+------------+------------+-----------+------------+------------+
|   t |     u1(t) |   u1(exact) |        E1 |      u2(t) |   u2(exact)
|           E2 |      u3(t) |   u3(exact) |          E3 |
+=====+==========+============+==========+===========+===========
=+============+===========+===========+============+
| 0   |   5       |     6       | 1         |   -9       |    -9
|  0          |   -5       |    -5       |  0          |
+-----+----------+------------+----------+-----------
+------------+------------+-----------+------------+------------+
| 0.2 |   5.854   |     6.6552  | 0.801198  |   -6.58494 |    -6.58259
|  0.00234919 |   -5.5112  |    -5.50771 |  0.00348883 |
+-----+----------+------------+----------+-----------
+------------+------------+-----------+------------+------------+
```

```
| 0.4 |   8.38331 |     8.98822 | 0.604912  |   -4.90202 |    -4.89241
|  0.0096013  |   -5.8413  |    -5.82373 |  0.0175677  |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 0.6 |  13.5889  |    14.0029  | 0.413971  |   -2.95419 |    -2.92103
|  0.0331595  |   -4.51809 |    -4.45343 |  0.0646549  |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 0.8 |  23.4176  |    23.652   | 0.234429  |    1.48419 |     1.58836
|  0.104171   |    2.56363 |     2.77046 |  0.206829   |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 1   |  41.4983  |    41.5771  | 0.0788238 |   13.8803  |    14.1859
|  0.30552    |   26.1091  |    26.7188  |  0.60974    |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 1.2 |  74.4959  |    74.4686  | 0.0273265 |   47.6679  |    48.5202
|  0.852321   |   92.8336  |    94.5372  |  1.70356    |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 1.4 | 134.588   |   134.555   | 0.0326936 |  135.261   |   137.553
|  2.29173    |  267.464   |   272.047   |  4.58257    |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 1.6 | 243.978   |   244.143   | 0.165025  |  353.504   |   359.498
|  5.99331    |  703.622   |   715.608   | 11.9859     |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 1.8 | 443.127   |   443.895   | 0.768209  |  882.032   |   897.373
| 15.3415     | 1760.53    |  1791.21    | 30.6824     |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
| 2   | 805.738   |   807.913   | 2.17471   | 2136.25    |  2174.86
| 38.6121     | 4268.96    |  4346.18    | 77.2237     |
+-----+-----------+------------+-----------+-----------
+-------------+------------+------------+------------+-------------+
```

Giải ptvp nhiều bước

Bài 2

2. Use each of the Adams-Bashforth methods to approximate the solutions to the following initial-value problems. In each case use starting values obtained from the Runge-Kutta method of order four. Compare the results to the actual values.

a. $y' = \dfrac{2-2ty}{t^2+1}, \quad 0 \le t \le 1, \quad y(0) = 1,$ with $h = 0.1$ actual solution $y(t) = \dfrac{2t+1}{t^2+2}$.

b. $y' = \dfrac{y^2}{1+t}, \quad 1 \le t \le 2, \quad y(1) = -(\ln 2)^{-1},$ with $h = 0.1$ actual solution $y(t) = \dfrac{-1}{\ln(t+1)}$.

c. $y' = (y^2+y)/t, \quad 1 \le t \le 3, \quad y(1) = -2,$ with $h = 0.2$ actual solution $y(t) = \dfrac{2t}{1-t}$.

d. $y' = -ty + 4t/y, \quad 0 \le t \le 1, \quad y(0) = 1,$ with $h = 0.1$ actual solution $y(t) = \sqrt{4-3e^{-t^2}}$.

```python
def Runge_kutta_4(x0, y0, a, b, h, Fxy):
    """
    Phương pháp Runge-Kutta bậc 4 để giải các phương trình vi phân thông thường bậc nhất.

    Tham số:
        x0 (float): Giá trị ban đầu của biến độc lập.
        y0 (float): Giá trị ban đầu của biến phụ thuộc.
        a (float): Bắt đầu của đoạn.
        b (float): Kết thúc của đoạn.
        h (float): Bước nhảy.
        Fxy (function): Một hàm đại diện cho đạo hàm dy/dx.

    Returns:
        list, list: Danh sách các giá trị x và y.
    """
    n = int((b - a) / h)
    x_ = [x0]  # Khởi tạo danh sách để lưu trữ các giá trị x
    y_ = [y0]  # Khởi tạo danh sách để lưu trữ các giá trị y

    for i in range(n):
        k1 = h * Fxy(x0, y0)
        k2 = h * Fxy(x0 + h/2, y0 + k1/2)
        k3 = h * Fxy(x0 + h/2, y0 + k2/2)
        k4 = h * Fxy(x0 + h, y0 + k3)

        # Tính toán giá trị y mới
        y_new = y0 + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        # Cập nhật các giá trị x và y
        x_new = x0 + h
        x_.append(x_new)
        y_.append(y_new)

        # Cập nhật x và y cho vòng lặp tiếp theo
        x0 = x_new
        y0 = y_new
```

```
    return x_, y_

def adams_fourth_order_predictor_corrector(res, stt, Fxy, h, n):
    """
    Phương pháp dự đoán-sửa lỗi bậc 4 của Adams để giải các
phương trình vi phân thông thường bậc nhất.

    Tham số:
        res (list): Danh sách các giải gần đúng.
        stt (list): Danh sách các điểm tương ứng trên đoạn.
        Fxy (function): Một hàm đại diện cho đạo hàm dy/dx.
        h (float): Bước nhảy.
        n (int): Số lần lặp.

    Returns:
        list: Danh sách các giải gần đúng đã được cập nhật.
    """
    # Lặp qua để tính toán các giải pháp
    for i in range(4, n):
        res[i] = res[i - 1] + h * (55 * Fxy(stt[i - 1], res[i - 1])
                                        - 59 * Fxy(stt[i - 2], res[i -
2])
                                        + 37 * Fxy(stt[i - 3], res[i -
3])
                                        - 4 * Fxy(stt[i - 4], res[i -
4])) / 24

    return res

#Cau a
a = 0
b = 1
x0 = 0
y0 = 1
h = 0.1
dy = lambda x, y: (2 - 2 * x * y)/ (x** 2 + 1)
Y = lambda x: (2 * x + 1)/ (x** 2 + 1)
stt, res = Runge_kutta_4(x0, y0, a, b, h, dy)
correct = adams_fourth_order_predictor_corrector(res, stt, dy, h,
len(res))
stt = np.array(stt)
correct = np.array(correct)
E = np.abs(Y(stt) - correct)
print("Câu a")
print(tabulate(zip(stt, correct, Y(stt), E), headers = ['x', 'Adams 4
bước', 'Exact Values', 'Error'], tablefmt="grid"))

#Cau b
a = 1
b = 2
```

```
x0 = 1
y0 = -np.log(2)**(-1)
h = 0.1
dy = lambda x, y: y**2/ (1 + x)
Y = lambda x: -1/ np.log(x + 1)
stt, res = Runge_kutta_4(x0, y0, a, b, h, dy)
correct = adams_fourth_order_predictor_corrector(res, stt, dy, h,
len(res))
stt = np.array(stt)
correct = np.array(correct)
E = np.abs(Y(stt) - correct)
print()
print("Câu b")
print(tabulate(zip(stt, correct, Y(stt), E), headers = ['x', 'Adams 4
bước', 'Exact Values', 'Error'], tablefmt="grid"))

#Cau c
a = 1
b = 3
x0 = 1
y0 = -2
h = 0.2
dy = lambda x, y: (y** 2 + y)/ x
Y = lambda x: 2 * x/ (1 - 2* x)
stt, res = Runge_kutta_4(x0, y0, a, b, h, dy)
correct = adams_fourth_order_predictor_corrector(res, stt, dy, h,
len(res))
stt = np.array(stt)
correct = np.array(correct)
E = np.abs(Y(stt) - correct)
print()
print("Câu c")
print(tabulate(zip(stt, correct, Y(stt), E), headers = ['x', 'Adams 4
bước', 'Exact Values', 'Error'], tablefmt="grid"))

#Cau d
a = 0
b = 1
x0 = 0
y0 = 1
h = 0.1
dy = lambda x, y: -x * y + 4 * x / y
Y = lambda x: np.sqrt(4 - 3 * np.e** (-x** 2))
stt, res = Runge_kutta_4(x0, y0, a, b, h, dy)
correct = adams_fourth_order_predictor_corrector(res, stt, dy, h,
len(res))
stt = np.array(stt)
correct = np.array(correct)
E = np.abs(Y(stt) - correct)
```

```python
print()
print("Câu d")
print(tabulate(zip(stt, correct, Y(stt), E), headers = ['x', 'Adams 4 bước', 'Exact Values', 'Error'], tablefmt="grid"))
```

```
Câu a
+-----+----------------+----------------+-------------+
|   x |   Adams 4 bước |   Exact Values |       Error |
+=====+================+================+=============+
| 0   |        1       |        1       | 0           |
+-----+----------------+----------------+-------------+
| 0.1 |        1.18812 |        1.18812 | 4.72059e-08 |
+-----+----------------+----------------+-------------+
| 0.2 |        1.34615 |        1.34615 | 2.37596e-07 |
+-----+----------------+----------------+-------------+
| 0.3 |        1.46789 |        1.46789 | 5.67627e-07 |
+-----+----------------+----------------+-------------+
| 0.4 |        1.59341 |        1.55172 | 0.0416854   |
+-----+----------------+----------------+-------------+
| 0.5 |        1.67183 |        1.6     | 0.0718308   |
+-----+----------------+----------------+-------------+
| 0.6 |        1.71318 |        1.61765 | 0.0955284   |
+-----+----------------+----------------+-------------+
| 0.7 |        1.71851 |        1.61074 | 0.107769    |
+-----+----------------+----------------+-------------+
| 0.8 |        1.69622 |        1.58537 | 0.110853    |
+-----+----------------+----------------+-------------+
| 0.9 |        1.65272 |        1.54696 | 0.105762    |
+-----+----------------+----------------+-------------+
| 1   |        1.59499 |        1.5     | 0.0949949   |
+-----+----------------+----------------+-------------+

Câu b
+-----+----------------+----------------+-------------+
|   x |   Adams 4 bước |   Exact Values |       Error |
+=====+================+================+=============+
| 1   |      -1.4427   |      -1.4427   | 0           |
+-----+----------------+----------------+-------------+
| 1.1 |      -1.34782  |      -1.34782  | 3.15799e-08 |
+-----+----------------+----------------+-------------+
| 1.2 |      -1.2683   |      -1.2683   | 4.85715e-08 |
+-----+----------------+----------------+-------------+
| 1.3 |      -1.20061  |      -1.20061  | 5.72913e-08 |
+-----+----------------+----------------+-------------+
| 1.4 |      -1.12072  |      -1.14225  | 0.0215302   |
+-----+----------------+----------------+-------------+
| 1.5 |      -1.05655  |      -1.09136  | 0.0348076   |
+-----+----------------+----------------+-------------+
| 1.6 |      -0.998441 |      -1.04656  | 0.0481188   |
+-----+----------------+----------------+-------------+
```

```
| 1.7 |       -0.950108 |      -1.00679  | 0.0566856   |
+-----+-----------------+----------------+-------------+
| 1.8 |       -0.907631 |      -0.971233 | 0.063602    |
+-----+-----------------+----------------+-------------+
| 1.9 |       -0.87073  |      -0.939222 | 0.0684925   |
+-----+-----------------+----------------+-------------+
| 2   |       -0.837991 |      -0.910239 | 0.0722484   |
+-----+-----------------+----------------+-------------+

Câu c
+-----+-----------------+----------------+-------------+
|   x |    Adams 4 bước |   Exact Values |       Error |
+=====+=================+================+=============+
| 1   |        -2       |        -2      | 0           |
+-----+-----------------+----------------+-------------+
| 1.2 |        -1.71425 |       -1.71429 | 4.05338e-05 |
+-----+-----------------+----------------+-------------+
| 1.4 |        -1.55552 |       -1.55556 | 3.26707e-05 |
+-----+-----------------+----------------+-------------+
| 1.6 |        -1.45452 |       -1.45455 | 2.57053e-05 |
+-----+-----------------+----------------+-------------+
| 1.8 |        -1.32068 |       -1.38462 | 0.0639357   |
+-----+-----------------+----------------+-------------+
| 2   |        -1.25969 |       -1.33333 | 0.0736433   |
+-----+-----------------+----------------+-------------+
| 2.2 |        -1.19358 |       -1.29412 | 0.100539    |
+-----+-----------------+----------------+-------------+
| 2.4 |        -1.16709 |       -1.26316 | 0.0960691   |
+-----+-----------------+----------------+-------------+
| 2.6 |        -1.13889 |       -1.2381  | 0.0992004   |
+-----+-----------------+----------------+-------------+
| 2.8 |        -1.12403 |       -1.21739 | 0.0933626   |
+-----+-----------------+----------------+-------------+
| 3   |        -1.10957 |       -1.2     | 0.0904305   |
+-----+-----------------+----------------+-------------+

Câu d
+-----+-----------------+----------------+-------------+
|   x |    Adams 4 bước |   Exact Values |       Error |
+=====+=================+================+=============+
| 0   |         1       |         1      | 0           |
+-----+-----------------+----------------+-------------+
| 0.1 |         1.01482 |        1.01482 | 3.67145e-07 |
+-----+-----------------+----------------+-------------+
| 0.2 |         1.05718 |        1.05718 | 1.19064e-06 |
+-----+-----------------+----------------+-------------+
| 0.3 |         1.1217  |        1.1217  | 1.87145e-06 |
+-----+-----------------+----------------+-------------+
| 0.4 |         1.20082 |        1.20149 | 0.000669079 |
```

```
+-----+----------------+----------------+--------------+
| 0.5 |        1.29501 |        1.28981 | 0.00520402   |
+-----+----------------+----------------+--------------+
| 0.6 |        1.39503 |        1.38093 | 0.0141014    |
+-----+----------------+----------------+--------------+
| 0.7 |        1.49613 |        1.47042 | 0.0257158    |
+-----+----------------+----------------+--------------+
| 0.8 |        1.59192 |        1.55503 | 0.0368856    |
+-----+----------------+----------------+--------------+
| 0.9 |        1.67935 |        1.63261 | 0.0467347    |
+-----+----------------+----------------+--------------+
| 1   |        1.7555  |        1.70187 | 0.05363      |
+-----+----------------+----------------+--------------+
```

GIẢI PTVP CẤP 2 = PP RUNGE KUTTA BẬC 4

```
def F(x, y, dy):
  return (x** 3 * np.log(x)- 2 * y + 2 * x * dy)/ x** 2
x0 = 1
y0 = 1
y1 = 0
a = 1
b = 2
h = 0.1
def Runge_kutta_4(x0, y0, y1, a, b, h, F, ilterations):
  def F1(x, y, dy):
    return dy
  F2 = F
  x = x0
  z1 = y0
  z2 = y1
  K1 = np.zeros([1, 2])
  K2 = np.zeros([1, 2])
  K3 = np.zeros([1, 2])
  K4 = np.zeros([1, 2])
  y_ = [y0]
  dy_ = [y1]
  x_  = [x0]
  for i in range(ilterations):
    K1[0, 0] = h * F1(x, z1, z2)
    K1[0, 1] = h * F2(x, z1, z2)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
```

```
        x = x + h
        y_.append(z1)
        dy_.append(z2)
        x_.append(x)
    return x_, y_, dy_

ilterations = int((b - a)/ h)
x, y, dy = Runge_kutta_4(x0, y0, y1, a, b, h, F, ilterations)
print(tabulate(zip(x, y, dy), headers = ['x', 'y', 'dy'],
tablefmt="grid"))
```

```
+-----+----------+-----------+
|   x |        y |        dy |
+=====+==========+===========+
| 1   | 1        |  0        |
+-----+----------+-----------+
| 1.1 | 0.990178 | -0.194513 |
+-----+----------+-----------+
| 1.2 | 0.961524 | -0.376187 |
+-----+----------+-----------+
| 1.3 | 0.915455 | -0.542409 |
+-----+----------+-----------+
| 1.4 | 0.853637 | -0.690774 |
+-----+----------+-----------+
| 1.5 | 0.777969 | -0.819058 |
+-----+----------+-----------+
| 1.6 | 0.690563 | -0.92519  |
+-----+----------+-----------+
| 1.7 | 0.593734 | -1.00723  |
+-----+----------+-----------+
| 1.8 | 0.489981 | -1.06336  |
+-----+----------+-----------+
| 1.9 | 0.381982 | -1.09187  |
+-----+----------+-----------+
| 2   | 0.272582 | -1.09112  |
+-----+----------+-----------+
```

PP Linear Shooting

Bài 2

2. The boundary-value problem

$$y'' = y' + 2y + \cos x, \quad 0 \le x \le \tfrac{\pi}{2}, \quad y(0) = -0.3, \quad y\left(\tfrac{\pi}{2}\right) = -0.1$$

has the solution $y(x) = -\frac{1}{10}(\sin x + 3\cos x)$. Use the Linear Shooting method to approximate the solution, and compare the results to the actual solution.

a. With $h = \frac{\pi}{4}$;

b. With $h = \frac{\pi}{8}$.

```python
def Runge_kutta_4(x0, y0, y1, a, b, h, F, ilterations):
  def F1(x, y, dy):
    return dy
  F2 = F
  x = x0
  z1 = y0
  z2 = y1
  K1 = np.zeros([1, 2])
  K2 = np.zeros([1, 2])
  K3 = np.zeros([1, 2])
  K4 = np.zeros([1, 2])
  y_ = [y0]
  dy_ = [y1]
  x_  = [x0]
  for i in range(ilterations):
    K1[0, 0] = h * F1(x, z1, z2)
    K1[0, 1] = h * F2(x, z1, z2)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
    x = x + h
    y_.append(z1)
    dy_.append(z2)
    x_.append(x)
  return x_, y_, dy_

def linear_shooting_method(x0, alpha, beta, a, b, h, F, ilterations):
  #Bai toan 1
  F1 = F
  y0 = alpha
  y1 = 0
  x, u, i = Runge_kutta_4(x0, y0, y1, a, b, h, F1, ilterations)

  #Bai toan 2
  def F2(x, y, dy):
    return dy + 2 * y
  y0 = 0
  y1 = 1
  x, v, i = Runge_kutta_4(x0, y0, y1, a, b, h, F2, ilterations)

  #Ket qua cuoi cung
  u = np.array(u)
  v = np.array(v)
  y = u + v * (beta - u[-1])/ v[-1]
  return x, u, v, y
```

```python
def F(x, y, dy):
  return dy + 2 * y + np.cos(x)

a = 0
b = np.pi/ 2
x1 = 0
x2 = np.pi/ 2
alpha = -0.3
beta = -0.1
h = np.pi/ 4
ilterations = int((b - a)/ h)
X = np.linspace(a, b, ilterations + 1)
Y = lambda x: -0.1 * (np.sin(x) + 3 * np.cos(x))
x, u, v, y = linear_shooting_method(x1, alpha, beta, a, b, h, F,
ilterations)
E = np.abs(Y(X) - y)

print('Câu a, h = pi/ 4')
print(tabulate(zip(X, u, v, y, Y(X), E), headers = ['x', 'Xấp xỉ
y(x)', "Xấp xỉ y'(x)", 'y(x) điều chỉnh', 'Exact Values', 'Error'],
tablefmt="grid"))

print()
h = np.pi/ 8
ilterations = int((b - a)/ h)
X = np.linspace(a, b, ilterations + 1)
Y = lambda x: -0.1 * (np.sin(x) + 3 * np.cos(x))
x, u, v, y = linear_shooting_method(x1, alpha, beta, a, b, h, F,
ilterations)
E = np.abs(Y(X) - y)
print('Câu b, h = pi/ 8')
print(tabulate(zip(X, u, v, y, Y(X), E), headers = ['x', 'Xấp xỉ
y(x)',"Xấp xỉ y'(x)", 'y(x) điều chỉnh', 'Exact Values', 'Error'],
tablefmt="grid"))
```

```
Câu a, h = pi/ 4
+----------+---------------+----------------+-------------------
+----------------+-------------+
|        x |   Xấp xỉ y(x) |   Xấp xỉ y'(x) |   y(x) điều chỉnh |
Exact Values |       Error |
+==========+===============+================+===================+=====
===========+=============+
| 0        |     -0.3      |        0       |         -0.3      |
-0.3      | 5.55112e-17 |
+----------+---------------+----------------+-------------------
+----------------+-------------+
| 0.785398 |     -0.144031 |        1.41533 |         -0.282452 |
-0.282843 | 0.000390493 |
+----------+---------------+----------------+-------------------
```

```
+---------------+------------+
| 1.5708   |     0.614565 |        7.30632 |          -0.1       |
-0.1      | 5.55112e-17 |
+----------+--------------+---------------+------------------
+---------------+------------+

Câu b, h = pi/ 8
+----------+--------------+---------------+------------------
+---------------+------------+
|        x |   Xấp xỉ y(x) |   Xấp xỉ y'(x) |   y(x) điều chỉnh |
Exact Values |        Error |
+==========+==============+===============+===================+======
===========+============+
| 0        |     -0.3       |       0        |          -0.3       |
-0.3      | 5.55112e-17 |
+----------+--------------+---------------+------------------
+---------------+------------+
| 0.392699 |     -0.265016 |       0.505039 |          -0.315415 |
-0.315432 | 1.72417e-05 |
+----------+--------------+---------------+------------------
+---------------+------------+
| 0.785398 |     -0.138395 |       1.44731  |          -0.282825 |
-0.282843 | 1.76386e-05 |
+----------+--------------+---------------+------------------
+---------------+------------+
| 1.1781   |      0.132163 |       3.40053  |          -0.207184 |
-0.207193 | 8.61238e-06 |
+----------+--------------+---------------+------------------
+---------------+------------+
| 1.5708   |      0.658835 |       7.60413  |          -0.1       |
-0.1      | 5.55112e-17 |
+----------+--------------+---------------+------------------
+---------------+------------+
```

PP Shooting cho nonlinear problems

Bài 2

2. Use the Nonlinear Shooting Algorithm with $h = 0.25$ to approximate the solution to the boundary-value problem

$$y'' = 2y^3, \quad -1 \le x \le 0, \quad y(-1) = \frac{1}{2}, \quad y(0) = \frac{1}{3}.$$

Compare your results to the actual solution $y(x) = 1/(x+3)$.

```
def Runge_kutta_4(x0, y0, y1, a, b, h, F, ilterations):
  def F1(x, y, dy):
    return dy
  F2 = F
```

```
  x = x0
  z1 = y0
  z2 = y1
  K1 = np.zeros([1, 2])
  K2 = np.zeros([1, 2])
  K3 = np.zeros([1, 2])
  K4 = np.zeros([1, 2])
  y_ = [y0]
  dy_ = [y1]
  x_  = [x0]
  for i in range(ilterations):
    K1[0, 0] = h * F1(x, z1, z2)
    K1[0, 1] = h * F2(x, z1, z2)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
    x = x + h
    y_.append(z1)
    dy_.append(z2)
    x_.append(x)
  return x_, y_, dy_

def linear_shooting_method(x0, alpha, beta, a, b, h, F, ilterations):
  #Bai toan 1
  F1 = F
  y0 = alpha
  y1 = 0
  x, u, i = Runge_kutta_4(x0, y0, y1, a, b, h, F1, ilterations)

  #Bai toan 2
  def F2(x, y, dy):
    return 2* y** 3
  y0 = 0
  y1 = 1
  x, v, i = Runge_kutta_4(x0, y0, y1, a, b, h, F2, ilterations)

  #Ket qua cuoi cung
  u = np.array(u)
  v = np.array(v)
  y = u + v * (beta - u[-1])/ v[-1]
  return x, u, v, y

def F(x, y, dy):
  return 2* y** 3
```

```
a = -1
b = 0
x1 = -1
x2 = 0
alpha = 1/ 2
beta = 1/ 3
h = 0.25
ilterations = int((b - a)/ h)
X = np.linspace(a, b, ilterations + 1)
Y = lambda x: 1/ (x + 3)
x, u, v, y = linear_shooting_method(x1, alpha, beta, a, b, h, F,
ilterations)
E = np.abs(Y(X) - y)

print('Câu a, h = 0.25')
print(tabulate(zip(X, u, v, y, Y(X), E), headers = ['x', 'Xấp xỉ
y(x)', "Xấp xỉ y'(x)", 'y(x) điều chỉnh', 'Exact Values', 'Error'],
tablefmt="grid"))
```

```
Câu a, h = 0.25
+-------+---------------+----------------+-------------------
+----------------+------------+
|     x |   Xấp xỉ y(x) |   Xấp xỉ y'(x) |   y(x) điều chỉnh |   Exact
Values |      Error |
+=======+===============+================+===================+========
========+============+
| -1    |      0.5      |       0        |          0.5      |
0.5      | 0          |
+-------+---------------+----------------+-------------------
+----------------+------------+
| -0.75 |      0.507874 |       0.250081 |          0.437948 |
0.444444 | 0.00649607 |
+-------+---------------+----------------+-------------------
+----------------+------------+
| -0.5  |      0.532262 |       0.503103 |          0.391588 |
0.4      | 0.0084115  |
+-------+---------------+----------------+-------------------
+----------------+------------+
| -0.25 |      0.575702 |       0.774271 |          0.359207 |
0.363636 | 0.00442969 |
+-------+---------------+----------------+-------------------
+----------------+------------+
|  0    |      0.643346 |       1.10872  |          0.333333 |
0.333333 | 0          |
+-------+---------------+----------------+-------------------
+----------------+------------+
```

Bài 2

2. Use the Nonlinear Shooting Algorithm with $h = 0.25$ to approximate the solution to the boundary-value problem

$$y'' = 2y^3, \quad -1 \le x \le 0, \quad y(-1) = \frac{1}{2}, \quad y(0) = \frac{1}{3}.$$

Compare your results to the actual solution $y(x) = 1/(x+3)$.

```
def Runge_kutta_4(x0, y0, y1, a, b, h, F, ilterations):
  def F1(x, y, dy):
    return dy
  F2 = F
  x = x0
  z1 = y0
  z2 = y1
  K1 = np.zeros([1, 2])
  K2 = np.zeros([1, 2])
  K3 = np.zeros([1, 2])
  K4 = np.zeros([1, 2])
  y_ = [y0]
  dy_ = [y1]
  x_  = [x0]
  for i in range(ilterations):
    K1[0, 0] = h * F1(x, z1, z2)
    K1[0, 1] = h * F2(x, z1, z2)
    K2[0, 0] = h * F1(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K2[0, 1] = h * F2(x + h/ 2, z1 + K1[0, 0]/ 2, z2 + K1[0, 1]/ 2)
    K3[0, 0] = h * F1(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K3[0, 1] = h * F2(x + h/ 2, z1 + K2[0, 0]/ 2, z2 + K2[0, 1]/ 2)
    K4[0, 0] = h * F1(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    K4[0, 1] = h * F2(x + h, z1 + K3[0, 0], z2 + K3[0, 1])
    z1 = z1 + (K1[0, 0] + 2 * K2[0, 0] + 2 * K3[0, 0] + K4[0, 0]) / 6
    z2 = z2 + (K1[0, 1] + 2 * K2[0, 1] + 2 * K3[0, 1] + K4[0, 1]) / 6
    x = x + h
    y_.append(z1)
    dy_.append(z2)
    x_.append(x)
  return x_, y_

def shooting_nonlinear_problems(x1, x2, alpha, beta, h, ilterations,
F, eps):
  #Chon dieu kien ban dau
  x0 = x1
  y0 = alpha
  # Du doan 2 gia tri dau
  gamma_1 = -5
  gamma_2 = 5
  x, y = Runge_kutta_4(x0, y0, gamma_1, a, b, h, F, ilterations)
  phi_1 = y[-1]
  x, y = Runge_kutta_4(x0, y0, gamma_2, a, b, h, F, ilterations)
```

```
  phi_2 = y[-1]
  #Xap xi gamma
  while(1):
    gamma_3 = gamma_2 - (phi_2 - beta) * (gamma_2 - gamma_1)/ (phi_2 -
phi_1)
    x, y = Runge_kutta_4(x0, y0, gamma_3, a, b, h, F, ilterations)
    phi_3 = y[-1]
    if (np.abs(phi_3 - beta) < eps):
      return Runge_kutta_4(x0, y0, gamma_3, a, b, h, F, ilterations)
    gamma_1 = gamma_2
    gamma_2 = gamma_3
    phi_1 = phi_2
    phi_2 = phi_3


def F(x, y, dy):
  return 2 * y** 3

a = -1
b = 0
x1 = -1
x2 = 0
alpha = 1/2
beta = 1/3
h = 0.25
eps = 1e-4
ilterations = int((b - a)/ h)
X = np.linspace(a, b, ilterations + 1)
Y = lambda x: 1/ (x + 3)
x, res = shooting_nonlinear_problems(x1, x2, alpha, beta, h,
ilterations, F, eps)
E = np.abs(Y(X) - res)
print(tabulate(zip(X, res, Y(X), E), headers = ['x', 'y', 'Exact
Values', 'Error'], tablefmt="grid"))
```

```
+-------+----------+----------------+-------------+
|     x |        y |   Exact Values |       Error |
+=======+==========+================+=============+
| -1    | 0.5      |       0.5      | 0           |
+-------+----------+----------------+-------------+
| -0.75 | 0.444446 |       0.444444 | 1.84084e-06 |
+-------+----------+----------------+-------------+
| -0.5  | 0.400002 |       0.4      | 1.83279e-06 |
+-------+----------+----------------+-------------+
| -0.25 | 0.363637 |       0.363636 | 1.00618e-06 |
+-------+----------+----------------+-------------+
|  0    | 0.333333 |       0.333333 | 2.38553e-07 |
+-------+----------+----------------+-------------+
```