1) a) The method can be re-written like

$$0\cdot y_K - 1 y_{K+1} + 1\cdot y_{K+2} = h\left(b_0 f_K + b_1 f_{K+1} + b_2 f_{K+2}\right)$$

So that with $m=2$, $\begin{bmatrix} a_0 = 0 \\ a_1 = -1 \\ a_2 = 1 \end{bmatrix}$. As the method is third order, we must satisfy the theorem for $p=3$, meaning $j \in \{1,2,3\}$, leading to the following system of equations

$$\begin{cases} a_1 + 2a_2 = b_0 + b_1 + b_2, & j=1 \\ a_1 + 4a_2 = 2b_1 + 4b_2, & j=2 \\ a_1 + 8a_2 = 3b_1 + 12b_2, & j=3 \end{cases}$$

$$\downarrow$$

$$\begin{cases} 1 = b_0 + b_1 + b_2 & \textcircled{1} \\ 3 = 2b_1 + 4b_2 & \textcircled{2} \\ 7 = 3b_1 + 12b_2 & \textcircled{3} \end{cases}$$

$\textcircled{3} - \textcircled{2}$

$\rightarrow \quad 4 = b_1 + 8b_2 \rightarrow b_1 = 4 - 8b_2 \rightarrow b_1 = 4 - \frac{40}{12} = \frac{8}{12}$

$\textcircled{3} \rightarrow \quad 3 = 8 - 12b_2 \rightarrow b_2 = 5/12$

$\textcircled{1} \rightarrow \quad b_0 + 8/12 + 5/12 = 1 \rightarrow b_0 = -1/12$

So, $\boxed{b_0 = -1/12, \quad b_1 = 2/3, \quad b_2 = 5/12}$

(b) Using Newton's form, our polynomial takes the form

$$p(s) = a + b(s - t_{K+1}) + c(s - t_{K+1})(s - t_{K+2})$$

Where we can find a, b, c with the divided difference table

$$\begin{array}{llll}
t_{K+1} & f_{K+1} & & \\
t_{K+2} & f_{K+2} & \dfrac{f_{K+2} - f_{K+1}}{t_{K+2} - t_{K+1}} = \dfrac{f_{K+2} - f_{K+1}}{h} & \\
t_K & f_K & \dfrac{f_K - f_{K+2}}{t_K - t_{K+2}} = \dfrac{f_{K+2} - f_K}{2h} & \dfrac{\dfrac{f_{K+2} - f_K}{2h} - \dfrac{f_{K+2} - f_{K+1}}{h}}{-h}
\end{array}$$

So that $a = f_{K+1}$, $b = \dfrac{f_{K+2} - f_{K+1}}{h}$, $c = \dfrac{f_{K+2} - 2f_{K+1} + f_K}{2h^2}$

We then have

$$y_{K+2} = y_{K+1} + \int_{t_{K+1}}^{t_{K+2}} a + b(s - t_{K+1}) + c(s - t_{K+1})(s - t_{K+2}) \, ds$$

$$= y_{K+1} + a(t_{K+2} - t_{K+1}) + \frac{b}{2}(s - t_{K+1})^2 \Big|_{t_{K+1}}^{t_{K+2}}$$

$$+ c\left(\frac{s^3}{3} - \frac{s^2}{2}(t_{K+1} + t_{K+2}) + t_{K+1} t_{K+2} s\right) \Big|_{t_{K+1}}^{t_{K+2}}$$

$$= y_{K+1} + ah + \frac{b}{2}\left[(\overbrace{t_{K+2} - t_{K+1}}^{h})^2 \right] + c \left[\frac{1}{3}(t_{K+2}^3 - t_{K+1}^3) \right.$$

$$\left. - \frac{1}{2}(t_{K+1} + t_{K+2})(t_{K+2}^2 - t_{K+1}^2) + t_{K+1}t_{K+2}(\overbrace{t_{K+2} - t_{K+1}}^{h}) \right]$$

$$= y_{K+1} + ah + \frac{b}{2}h^2 + c\left[\frac{1}{3}(\overbrace{t_{K+2} - t_{K+1}}^{h})(t_{K+2}^2 + t_{K+2}t_{K+1} + t_{K+1}^2) \right.$$

$$\left. - \frac{1}{2}(t_{K+1} + t_{K+2})^2 h + t_{K+1}t_{K+2}h \right]$$

$$= y_{K+1} + ah + \frac{b}{2}h^2 + ch\left[\frac{1}{3}t_{K+2}^2 + \frac{1}{3}t_{K+2}t_{K+1} + \frac{1}{3}t_{K+1}^2 \right.$$
$$\left. - \frac{1}{2}t_{K+2}^2 - \cancel{t_{K+1}t_{K+2}} - \frac{1}{2}t_{K+1}^2 + \cancel{t_{K+1}t_{K+2}} \right]$$

$$= y_{K+1} + ah + \frac{b}{2}h^2 + ch\left[-\frac{1}{6}t_{K+2}^2 + \frac{1}{3}t_{K+2}t_{K+1} - \frac{1}{6}t_{K+1}^2 \right]$$

$$= y_{K+1} + ah + \frac{b}{2}h^2 - \frac{ch}{6}\left[(\overbrace{t_{K+2} - t_{K+1}}^{h^2})^2 \right]$$

$$= y_{K+1} + ah + \frac{b}{2}h^2 - \frac{c}{6}h^3$$

Now substituting for $a, b, c,$ we have

$$= y_{K+1} + hf_{K+1} + \frac{f_{K+2} - f_{K+1}}{2\cancel{h}}h^{\cancel{2}} - \frac{f_{K+2} - 2f_{K+1} + f_K}{12h^{\cancel{2}}}h^{\cancel{3}}$$

$$= y_{K+1} + h\left[\frac{12f_{K+1} + 6f_{K+2} - 6f_{K+1} - f_{K+2} + 2f_{K+1} - f_K}{12} \right]$$

$$= y_{k+1} + h\left[-\frac{1}{12} f_k + \frac{2}{3} f_{k+1} + \frac{5}{12} f_{k+2}\right]$$

So just as in (a), $\boxed{b_0 = -1/2 \;,\; b_1 = 2/3 \;,\; b_2 = 5/12}$

2)$^{(a)}$ By re-indexing, we can write the method as

$$y_{K+2} - y_K = h\left[f_{K+2} - 3f_{K+1} + 4f_K\right]$$

So that $[a_0 = -1,\ a_1 = 0,\ a_2 = 1]$ and $[b_0 = 4,\ b_1 = -3,\ b_2 = 1]$
The characteristic polynomial is

$$\lambda^2 - 1 := 0$$

$$\leftrightarrow \quad \lambda = \pm 1$$

So as each λ is distinct and $|\lambda| \le 1$, the method is <u>zero stable</u>. Now we check the order of convergence,

$$a_0 + a_1 + a_2 = -1 + 0 + 1 = 0 \ \checkmark$$

$$0\cdot\overset{a_0}{(-1)} + 1\overset{a_1}{(0)} + 2\overset{a_2}{(1)} = 2 = \overset{4}{b_0} + \overset{-3}{b_1} + \overset{1}{b_2} = 2 \qquad p=1$$

$$0^2\cdot(-1) + 1^2\cdot 0 + 2^2\cdot 1 = 4 \ne 2(-3+2) = -2 \qquad p=2$$

So by the Dahlquist Equivalence Theorem (DET), this method is <u>convergent</u> with <u>LTE $O(h)$</u>.

(b) Again re-indexing, we have

$$y_{k+2} - 2y_{k+1} + y_k = h(f_{k+2} - f_{k+1})$$

So that $[a_0 = 1, a_1 = -2, a_2 = 1]$ and $[b_0 = 0, b_1 = -1, b_2 = 1]$
The characteristic polynomial is

$$\lambda^2 - 2\lambda + 1 := 0$$

$$\leftrightarrow (\lambda - 1)^2 = 0 \leftrightarrow \lambda = 1, \text{ w/ mult} = 2$$

So both roots have magnitude 1, and fails the root condition, so that this method is <u>not convergent</u>, by the (DET). We now find the LTE, we have

$$1 + (-2) + 1 = 0 \checkmark$$

$$0 \cdot 1 + 1 \cdot (-2) + 2 \cdot 1 = 0 = 0 + (-1) + 1 \checkmark \quad p = 1$$

$$0^2 \cdot 1 + 1^2 \cdot (-2) + 2^2 \cdot 1 = 2 = 2(-1 + 2) = 2 \checkmark \quad p = 2$$

$$0^3 \cdot 1 + 1^3 \cdot (-2) + 2^3 \cdot 1 = 6 \neq 3(-1 + 4) = 9 \times \quad p = 3$$

So the <u>LTE is $O(h^2)$</u>.

(c) Again re-indexing, we have

$$y_{k+2} - y_{k+1} - y_k = h\left[f_{k+2} - f_{k+1}\right]$$

So that $[a_0 = -1, a_1 = -1, a_2 = 1]$ and $[b_0 = 0, b_1 = -1, b_2 = 1]$.

However, notice that $a_0 + a_1 + a_2 = -1 \neq 0$. So this method cannot have LTE $O(h^p)$ for any $p \geq 1$, and immediately fails the (DET) so that this method is <u>not convergent</u>.

3) a) I used the forward Euler method because I figured it would be good to try. To back up my choice, I compared the numerical derivative of the computed solution with the real dy/dt and $d\alpha/dt$ with the L2 norm. The approximation seems very accurate except at the "cusps" of the oscillations as indicated by the figures on the following pages. I needed to have h be small in order to converge to this solution, if it was too large, the solutions blow up.

b) The Jacobian is $\begin{bmatrix} 1+h(x_{k+1}^2-1) & -h \\ \varepsilon h & 1 \end{bmatrix}$.

which is found by the system

$$0 = x_{k+1} - x_k - h\left(-\frac{x_{k+1}^3}{3} + x_{k+1} + \alpha_{k+1}\right)$$

$$0 = \alpha_{k+1} - \alpha_k + h\varepsilon x_{k+1} .$$

I was able to use larger step values and still get a good solution.

4) we send the method through the test equation

$y' = \lambda y$,

$$\rightarrow \quad y_{k+1} = y_k + h\lambda \frac{(y_k + y_{k+1})}{2}$$

$$\rightarrow \quad y_{k+1} - \frac{h\lambda}{2} y_{k+1} = y_k + \frac{h\lambda}{2} y_k$$

$$\rightarrow \quad y_{k+1}\left(1 - \frac{h\lambda}{2}\right) = y_k\left(1 + \frac{h\lambda}{2}\right)$$

$$\rightarrow \quad y_{k+1} = \frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}} y_k$$

So if $z = h\lambda$, the region of absolute stability is defined by

$$\left| \frac{1 + \frac{z}{2}}{1 - \frac{z}{2}} \right| \leq 1$$

$$\leftrightarrow \quad \frac{\left|1 + \frac{1}{2}x + \frac{1}{2}iy\right|}{\left|1 - \frac{1}{2}x - \frac{1}{2}iy\right|} \leq 1$$

$$\frac{\sqrt{(1 + 1/2x)^2 + (1/2y)^2}}{\sqrt{(1 - 1/2x)^2 + (-1/2y)^2}} \leq 1$$

$$\leftrightarrow \quad (1+\tfrac{1}{2}x)^2 + \cancel{\tfrac{1}{4}y^2} \leq (1-\tfrac{1}{2}x)^2 + \cancel{\tfrac{1}{4}y^2}$$

$$\leftrightarrow \quad |1+\tfrac{1}{2}x| \leq |1-\tfrac{1}{2}x|$$

which is true when $x \leq 0$. As $x = \mathrm{Re}(z)$, the region of absolute stability is the entire left half plane, hence the method is A-stable.

5) We have

$$|y_{k+1} - w^{j+1}| = \left| \frac{h\lambda}{2} (y_{k+1} - w^j) \right|$$

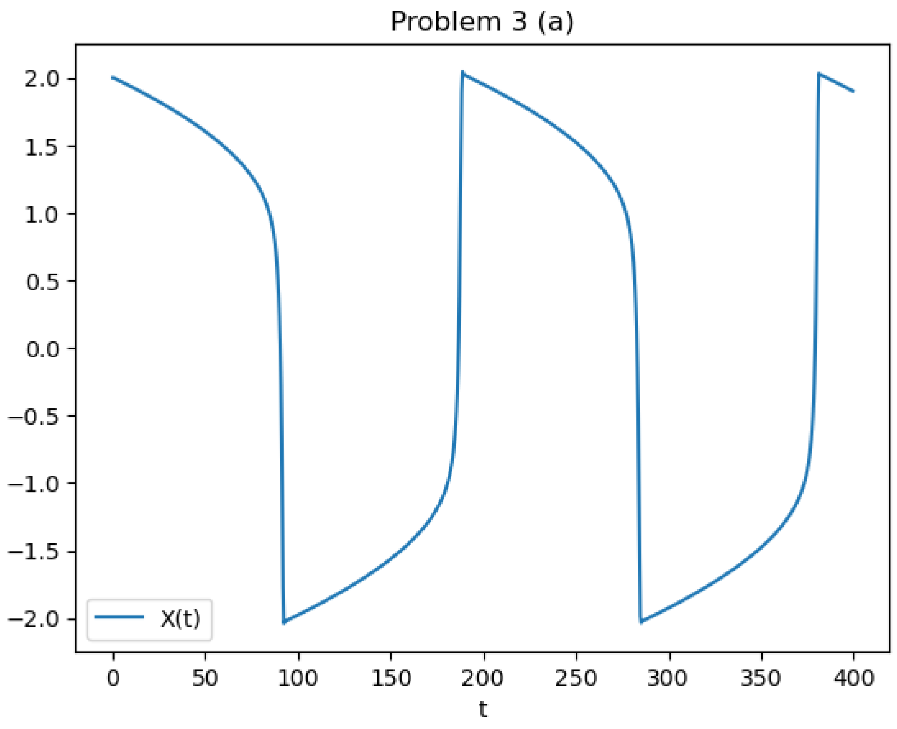
$$= \left|\frac{h\lambda}{2}\right| \, |y_{k+1} - w^j|$$

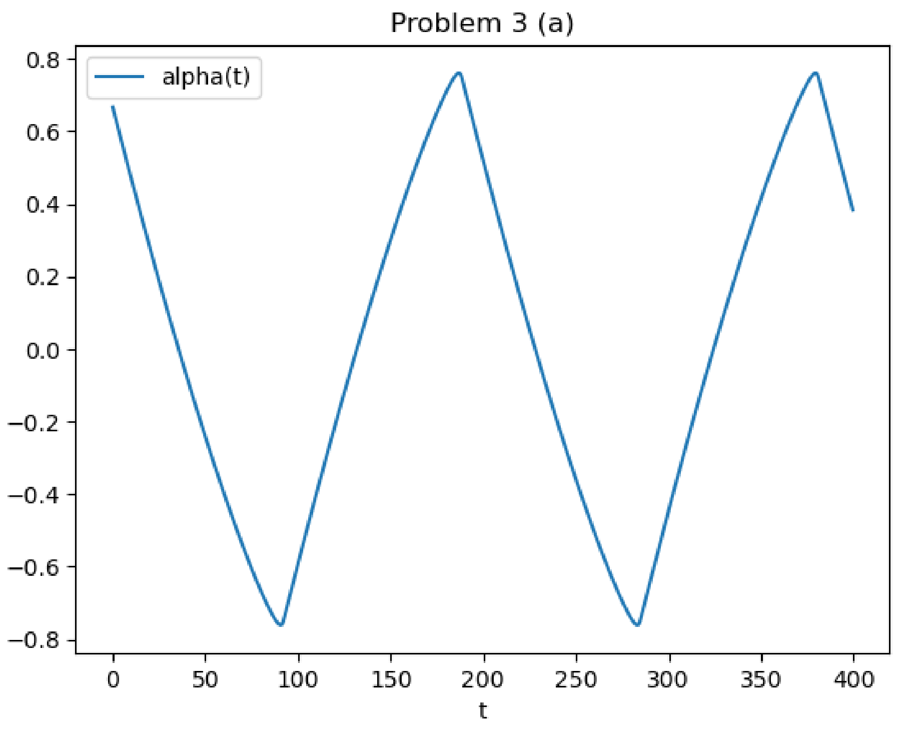
This implies convergence to y_{k+1} immediately if this a contraction, ie if

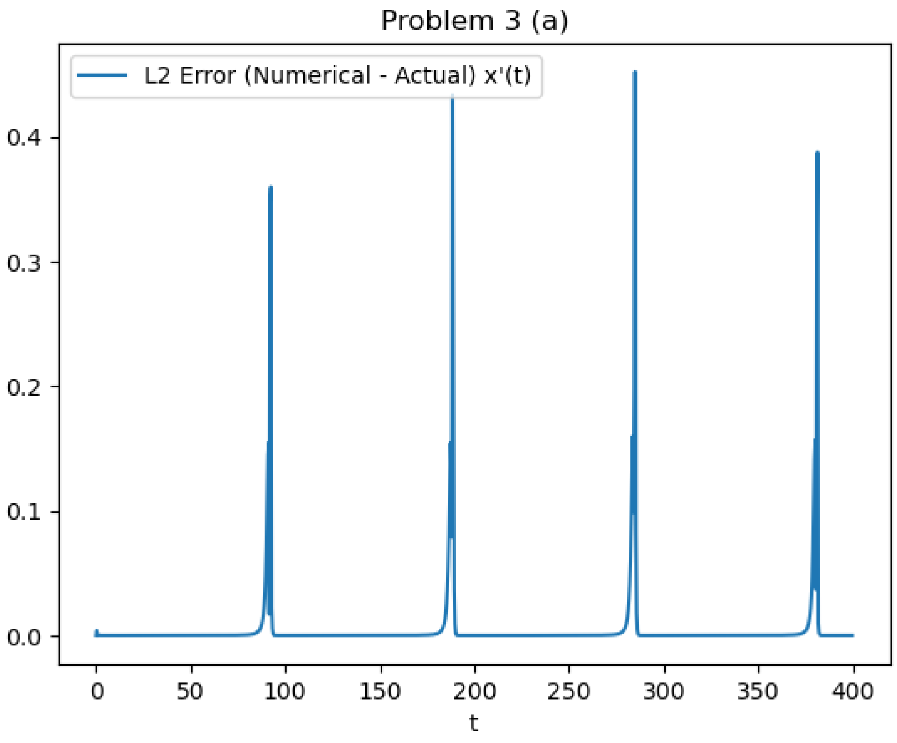
$\left|\frac{h\lambda}{2}\right| < 1 \quad \leftrightarrow \quad \boxed{|h\lambda| < 2.}$

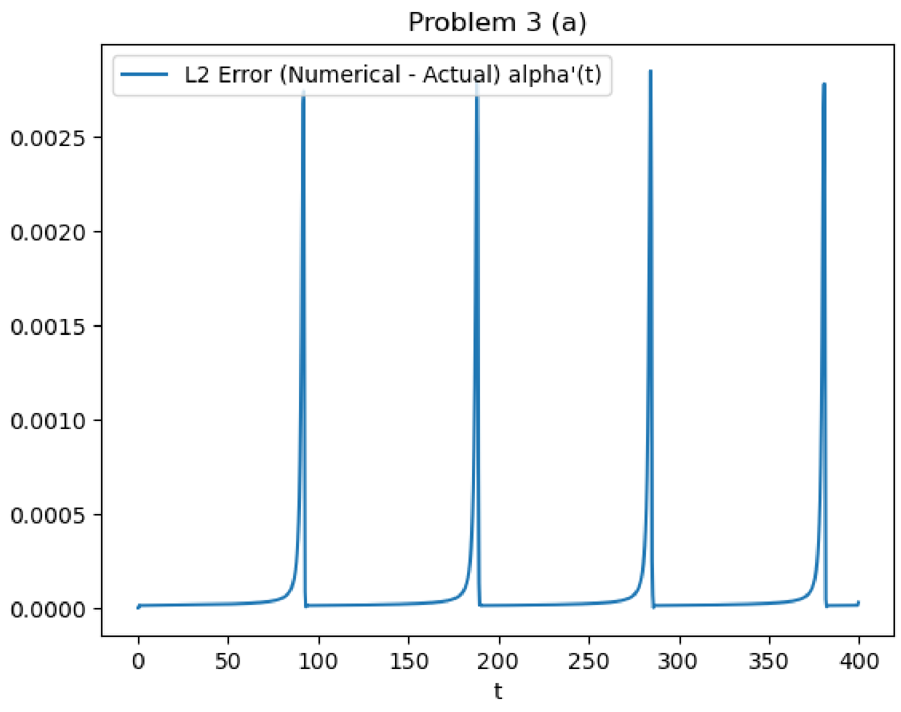
```
import numpy as np
import matplotlib.pyplot as plt

# Problem 3 (a).
# Use Euler's method (forward).
# Params.
eps = 1/100
x0 = 2
a0 = 2/3

# To t = 400.
n = 1000
h = 400 / n

# Make storage for solutions.
x = np.zeros(shape=n)
x[0] = x0
a = np.zeros(shape=n)
a[0] = a0

# Do forward Euler.
for i in range(1, n):
    f_x_i = -(x[i-1] ** 3)/3 + x[i-1] + a[i-1]
    f_a_i = -eps * x[i-1]
    x[i] = x[i-1] + h * f_x_i
    a[i] = a[i-1] + h * f_a_i

# Compute numerical derivative / error.
t = np.arange(n) * h
numer_deriv_x = np.gradient(x, t)
actual_deriv_x = -(x ** 3)/3 + x + a
error_x = np.sqrt((numer_deriv_x - actual_deriv_x) ** 2)

numer_deriv_a = np.gradient(a, t)
actual_deriv_a = -eps * x
error_a = np.sqrt((numer_deriv_a - actual_deriv_a) ** 2)

# Plot solutions.
fig, ax = plt.subplots()
plt.plot(t, x, label="X(t)")
plt.title("Problem 3 (a)")
plt.xlabel("t")
plt.legend()
plt.show()

fig, ax = plt.subplots()
```

```
plt.plot(t, a, label="alpha(t)")
plt.title("Problem 3 (a)")
plt.xlabel("t")
plt.legend()
plt.show()

fig, ax = plt.subplots()
plt.plot(t, error_x, label="L2 Error (Numerical - Actual) x'(t)")
plt.title("Problem 3 (a)")
plt.xlabel("t")
plt.legend()
plt.show()

fig, ax = plt.subplots()
plt.plot(t, error_a, label="L2 Error (Numerical - Actual) alpha'(t)")
plt.title("Problem 3 (a)")
plt.xlabel("t")
plt.legend()
plt.show()
```



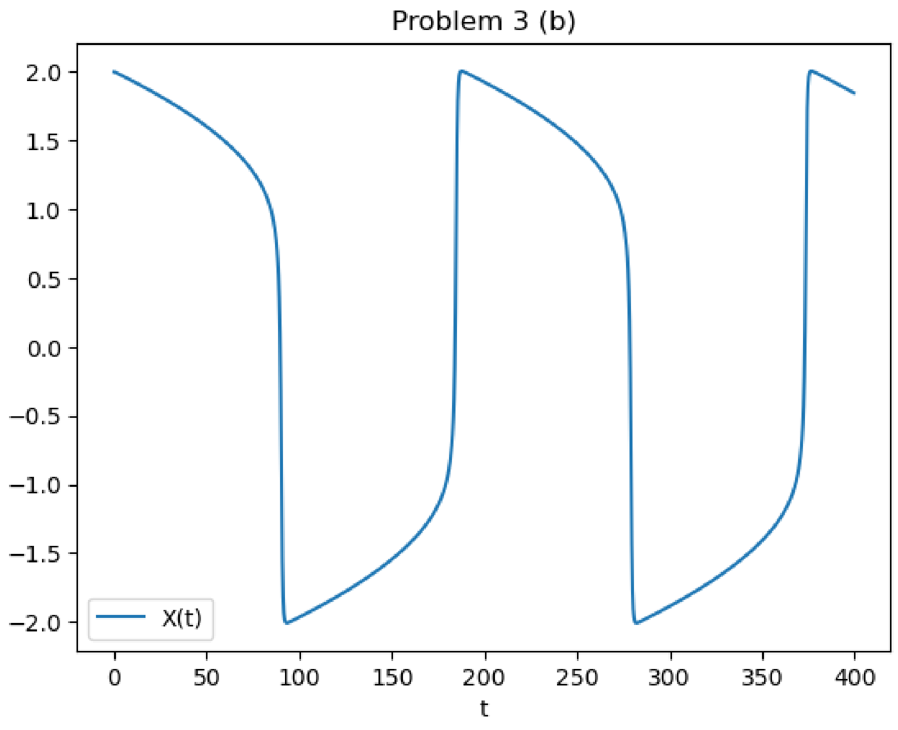

Problem 3 (a)
alpha(t)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
50
100
150
200
250
300
350
400
t

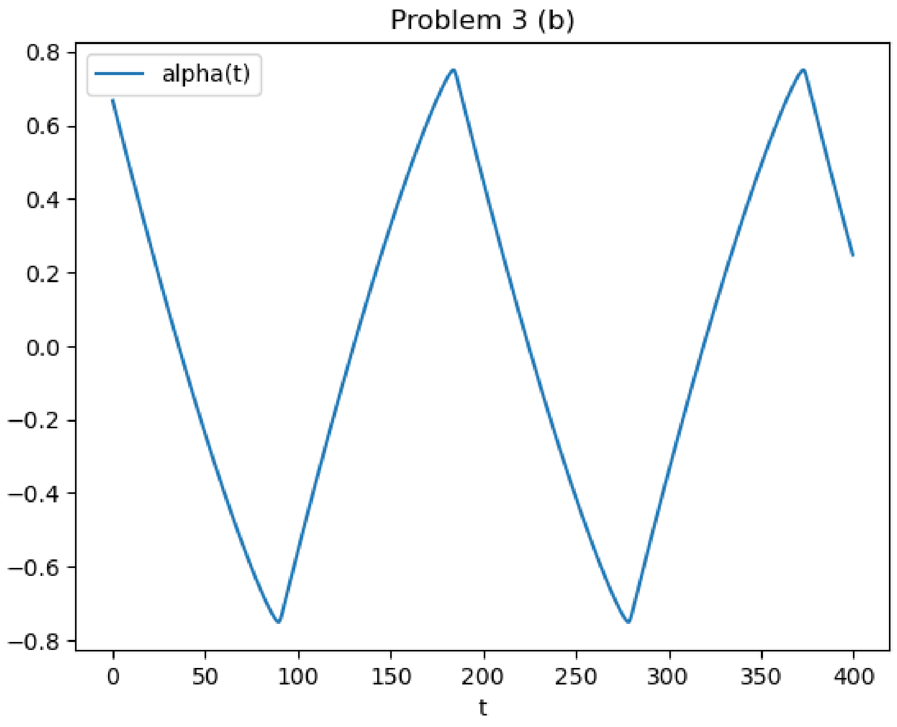

Problem 3 (a)
L2 Error (Numerical - Actual) x'(t)
0.0
0.1
0.2
0.3
0.4
0
50
100
150
200
250
300
350
400
t

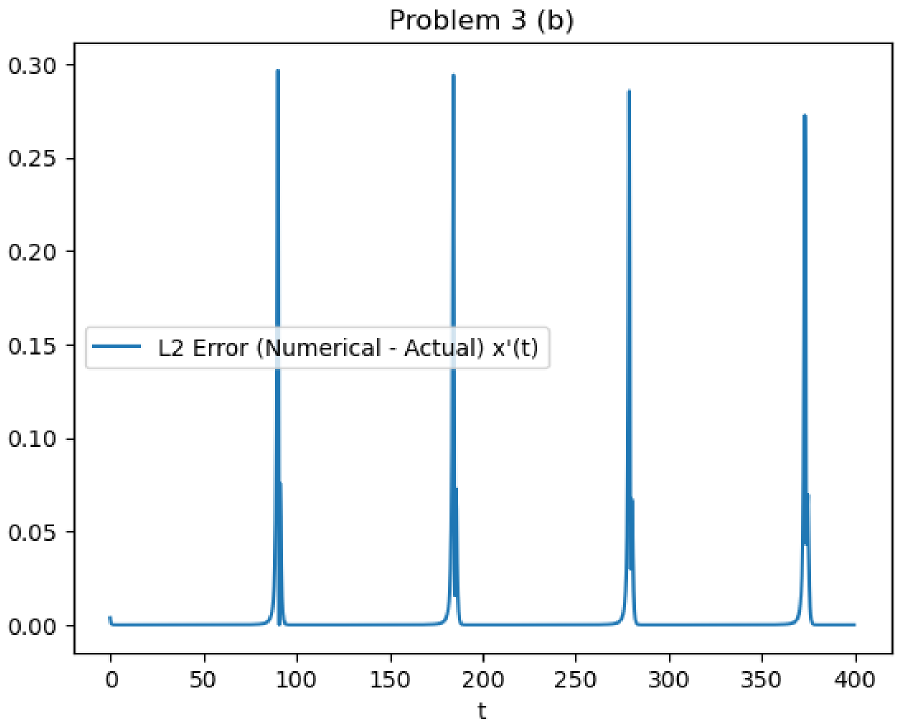


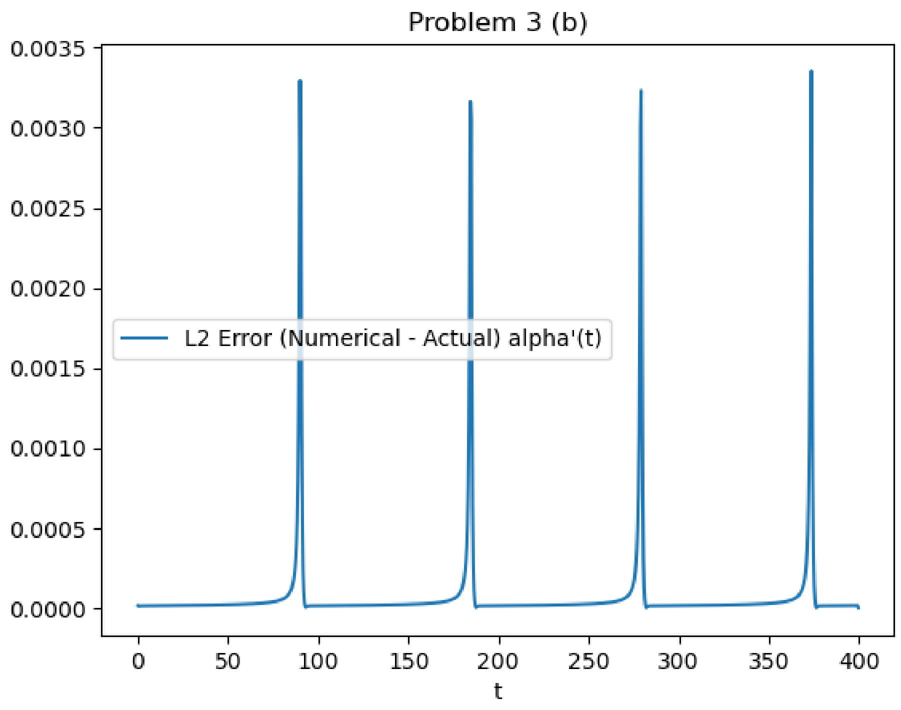
```
# Problem 3 (b).
# Use Euler's method (backward).
# Params.
eps = 1/100
x0 = 2
a0 = 2/3

# To t = 400.
n = 1000
h = 400 / n

# Make storage for solutions.
x = np.zeros(shape=n)
x[0] = x0
a = np.zeros(shape=n)
a[0] = a0

# Stack x and a.
w = np.row_stack([x, a])

for i in range(1, n):
    # Previous.
    w_prev = w[:, i-1]
    x_prev = w_prev[0]
    a_prev = w_prev[1]

    if i == 1:
        # Initial guess for the new values (starting with the previous values).
        x_new = x_prev
        a_new = a_prev

    # Get G.
    g0 = x_new - x_prev - h * (-(1/3) * x_prev ** 3 + x_prev + a_prev)
    g1 = a_new - a_prev + h * eps * x_prev
    G = np.array([g0, g1])

    # Get J.
    J = np.array([[1 + h*(x_prev ** 2 - 1),  -h], [eps * h, 1]])

    # Update the guess for w_new using Newton's method
    delta_w = np.linalg.solve(J, -G)
    x_new += delta_w[0]
    a_new += delta_w[1]

    # # Update.
    w[:, i] = np.array([x_new, a_new])

# Separate x and a.
t = np.arange(n) * h
x = w[0, :]
a = w[1, :]

# Compute numerical derivative / error.
numer_deriv_x = np.gradient(x, t)
actual_deriv_x = -(x ** 3)/3 + x + a
error_x = np.sqrt((numer_deriv_x - actual_deriv_x) ** 2)

numer_deriv_a = np.gradient(a, t)
actual_deriv_a = -eps * x
error_a = np.sqrt((numer_deriv_a - actual_deriv_a) ** 2)

# Plot.
fig, ax = plt.subplots()
plt.plot(t, x, label="X(t)")
plt.title("Problem 3 (b)")
plt.xlabel("t")
plt.legend()
plt.show()
```

```
fig, ax = plt.subplots()
plt.plot(t, a, label="alpha(t)")
plt.title("Problem 3 (b)")
plt.xlabel("t")
plt.legend()
plt.show()

fig, ax = plt.subplots()
plt.plot(t, error_x, label="L2 Error (Numerical - Actual) x'(t)")
plt.title("Problem 3 (b)")
plt.xlabel("t")
plt.legend()
plt.show()

fig, ax = plt.subplots()
plt.plot(t, error_a, label="L2 Error (Numerical - Actual) alpha'(t)")
plt.title("Problem 3 (b)")
plt.xlabel("t")
plt.legend()
plt.show()
```




Problem 3 (b)
alpha(t)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
50
100
150
200
250
300
350
400
t


Problem 3 (b)
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200
250
300
350
400
t
L2 Error (Numerical - Actual) x'(t)


Problem 3 (b)
L2 Error (Numerical - Actual) alpha'(t)
t
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
50
100
150
200
250
300
350
400