

Homework 5 of Computational Mathematics*

Chang, Yung-Hsuan

111652004

Department of Applied Mathematics

May 23, 2024

*For further information (such as codes), please refer to https://github.com/eiken59/CM_HW/tree/main/HW5.

Problem 1. Show that each of the following initial-value problems has a unique solution and find the solution. Can Theorem 5.4 be applied in each case?

a. $y' = t^{-2}(\sin 2t - 2ty), \quad 1 \leq t \leq 2, \quad y(1) = 2;$ and

b. $y' = -y + t\sqrt{y}, \quad 2 \leq t \leq 3, \quad y(2) = 2.$

Solution.

a. The domain of $f(t, y) = \frac{\sin 2t - 2ty}{t^2}$ is $D = [1, 2] \times \mathbb{R}$. It is clear that f is continuous on D . Fix a $t \in [1, 2]$. Then, for $y_1, y_2 \in \mathbb{R}$,

$$\begin{aligned} |f(t, y_1) - f(t, y_2)| &= \frac{2t|y_1 - y_2|}{t^2} \\ &\leq 2|y_1 - y_2|, \end{aligned}$$

which implies f satisfies a Lipschitz condition in the variable y on D with a Lipschitz constant 2.

By Theorem 5.4, the initial-value problem has a unique solution. Using algebra, we have

$$\begin{aligned} t^2 y' + 2ty &= \sin 2t \\ t^2 y &= -\frac{1}{2} \cos 2t + C \\ \xrightarrow{y(1)=2} y &= \frac{-\cos 2t + 4 + \cos 2}{2t^2}. \end{aligned}$$

b.

Problem 2. For each choice of $f(t, y)$ given in parts (a)-(d):

- i. Does f satisfy a Lipschitz condition on $D = \{(t, y) \mid 0 \leq t \leq 1, -\infty < y < \infty\}$?
- ii. Can Theorem 5.6 be used to show that the initial-value problem

$$y' = f(t, y), \quad 0 \leq t \leq 1, \quad y(0) = 1$$

is well-posed?

- a. $f(t, y) = e^{t-y}$; and
- b. $f(t, y) = \frac{1+y}{1+t}$.

Solution.

- a. Fix a $t \in [0, 1]$. Then, for $y_1, y_2 \in \mathbb{R}$ with $y_1 > y_2$,

$$\begin{aligned} |e^{t-y_1} - e^{t-y_2}| &\geq |e^{-y_1} - e^{-y_2}| \\ &\geq e^{-y_2} \end{aligned}$$

is unbounded, which implies that f does not satisfy a Lipschitz condition on D in the variable y .

We cannot use Theorem 5.6 here since f does not satisfy a Lipschitz condition.

- b. Fix a $t \in [0, 1]$. Then, for $y_1, y_2 \in \mathbb{R}$,

$$\left| \frac{1+y_1}{1+t} - \frac{1+y_2}{1+t} \right| \leq |y_1 - y_2|,$$

which implies f satisfies a Lipschitz condition in the variable y on D with Lipschitz constant 1.

Since f is continuous on D , by Theorem 5.6, the initial-value problem is well-posed. \square

Problem 3. Use Euler's method to approximate the solutions for each of the following initial-value problems.

a. $y' = \frac{2 - 2ty}{t^2 + 1}$, $0 \leq t \leq 1$, $y(0) = 1$ with $h = 0.1$; and

b. $y' = \frac{y^2}{1 + t}$, $1 \leq t \leq 2$, $y(1) = -\frac{1}{\ln 2}$ with $h = 0.1$.

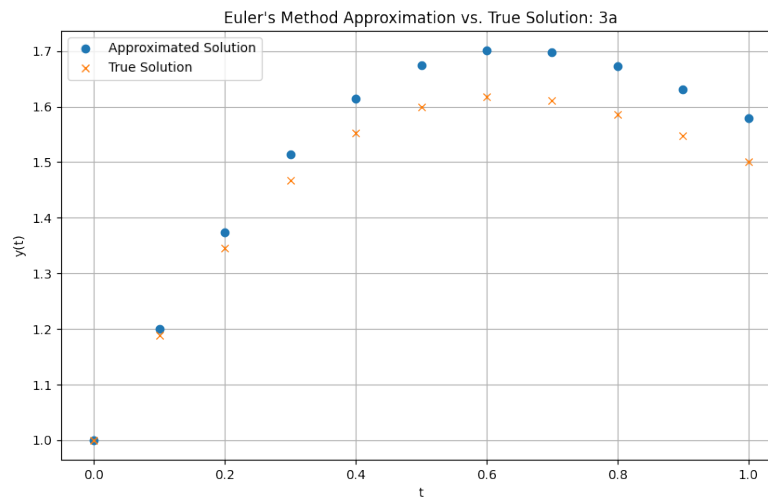
Show that the actual solutions are indeed $y(t) = \frac{2t + 1}{t^2 + 1}$ and $y(t) = \frac{-1}{\ln(t + 1)}$, respectively. Plot the errors between your numerical solutions and the exact solutions. Draw your conclusion regarding to the order of error with respect to the time step dt .

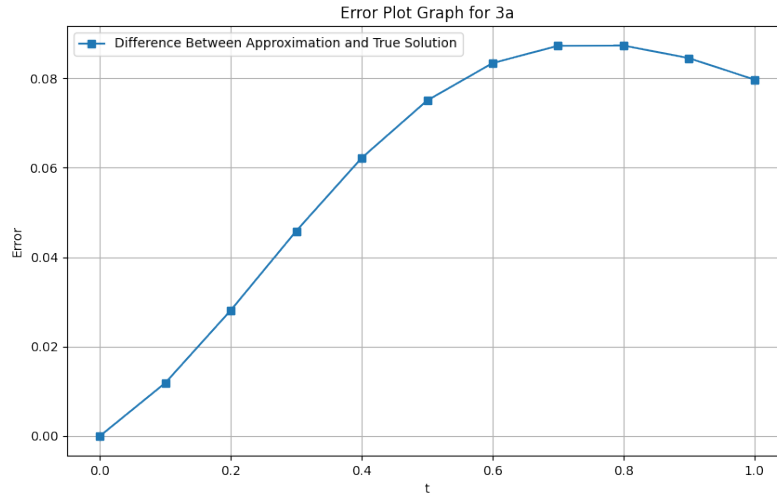
Solution.

- a. We directly differentiate the function and see whether it is a solution. Let $y(t) = \frac{2t + 1}{t^2 + 1}$. It is clear that $y(0) = 1$, and we have

$$\begin{aligned} y'(t) &= \frac{2(t^2 + 1) - (2t + 1)(2t)}{(t^2 + 1)^2} \\ &= \frac{2}{t^2 + 1} - \frac{2t}{t^2 + 1} \frac{2t + 1}{t^2 + 1} \\ &= \frac{2 - 2ty}{t^2 + 1}. \end{aligned}$$

The graph of the approximated solution and the actual solution can be seen below. The graph of error can also be seen below.

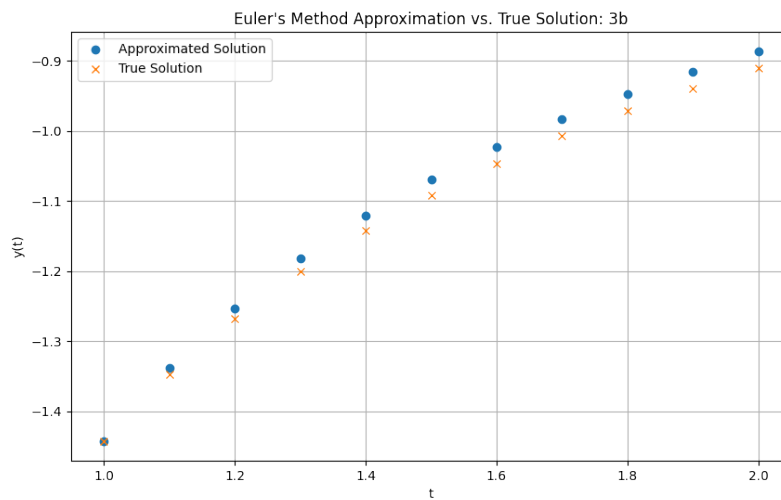


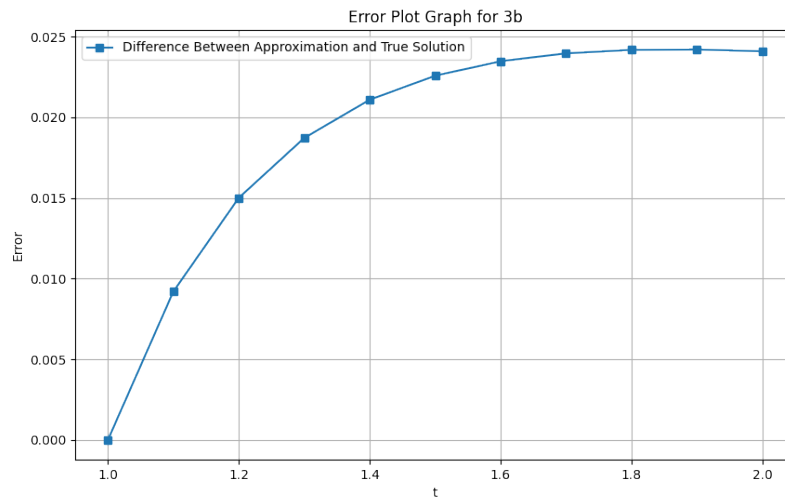


- b. We directly differentiate the function and see whether it is a solution. Let $y(t) = \frac{-1}{\ln(t+1)}$. It is clear that $y(1) = -\frac{1}{\ln 2}$, and we have

$$\begin{aligned} y'(t) &= \frac{1/(t+1)}{(\ln(t+1))^2} \\ &= \frac{y^2}{1+t}. \end{aligned}$$

The graph of the approximated solution and the actual solution can be seen below. The graph of error can also be seen below.





□

```

1. import math
2. import matplotlib.pyplot as plt
3.
4. def plot_arrays(x_axis, y_1s, y_2s, title):
5.     plt.figure(figsize=(10, 6))
6.     plt.plot(x_axis, y_1s, 'o', label='Approximated Solution')
7.     plt.plot(x_axis, y_2s, 'x', label='True Solution')
8.
9.     plt.xlabel('t')
10.    plt.ylabel('y(t)')
11.    plt.title(title)
12.    plt.legend()
13.    plt.grid(True)
14.    plt.savefig(f"P{title[-2:]} .png", transparent=True)
15.
16. def plot_errors(x_axis, y_1s, y_2s, title):
17.     plt.figure(figsize=(10, 6))
18.     y_diff = [abs(y1 - y2) for y1, y2 in zip(y_1s, y_2s)]
19.     plt.plot(x_axis, y_diff, 's-', label='Difference Between Approximation and True Solution')
20.
21.     for i in range(len(x_axis)):
22.         plt.annotate(f'{abs(y_1s[i] - y_2s[i]):.2e}', (x_axis[i], min(y_1s[i], y_2s[i])),
23.                     textcoords="offset points", xytext=(0, -15), ha='center')
24.
25.     plt.xlabel('t')
26.     plt.ylabel('Error')
27.     plt.title(title)
28.     plt.legend()
29.     plt.grid(True)
30.     plt.savefig(f"P{title[-2:]}e.png", transparent=True)
31.
32. def euler_method(f, true_y, alpha, a, b, h):
33.     N = int((b - a) / h)
34.     y_0 = alpha
35.     approx_soln_list = [y_0]
36.     real_soln_list = [true_y(a)]
37.     t_values = [a]
38.
39.     for i in range(1, N + 1):
40.         t_i = a + i * h
41.         t_values.append(t_i)
42.         y_0 += h * f(t_values[-2], y_0)
43.         approx_soln_list.append(y_0)
44.         real_soln_list.append(true_y(t_i))
45.
46.     return t_values, approx_soln_list, real_soln_list
47.
48. def f_a(t, y):
49.     return (2 - 2 * t * y) / (t * t + 1)
50.
51. def true_y_a(t):
52.     return (2 * t + 1) / (t * t + 1)
53.
54. def f_b(t, y):
55.     return y * y / (1 + t)
56.
57. def true_y_b(t):
58.     return -1 / math.log(t + 1)
59.
60.
61. t_values_a, approx_soln_a, real_soln_a = euler_method(f_a, true_y_a, 1, 0, 1, 0.1)
62.
63. plot_arrays(t_values_a, approx_soln_a, real_soln_a, "Euler's Method Approximation vs. True Solution: 3a")
64. plot_errors(t_values_a, approx_soln_a, real_soln_a, "Error Plot Graph for 3a")
65.
66.
67. t_values_b, approx_soln_b, real_soln_b = euler_method(f_b, true_y_b, -1 / math.log(2), 1, 2, 0.1)
68.
69. plot_arrays(t_values_b, approx_soln_b, real_soln_b, "Euler's Method Approximation vs. True Solution: 3b")
70. plot_errors(t_values_b, approx_soln_b, real_soln_b, "Error Plot Graph for 3b")

```

The code for Problem 3

Problem 4. Given the initial-value problem

$$y' = -y + t + 1, \quad 0 \leq t \leq 5, \quad y(0) = 1$$

with exact solution $y(t) = e^{-t} + t$.

- Approximate $y(5)$ using Euler's method with $h = 0.2$, $h = 0.1$, and $h = 0.05$.
- Determine the optimal value of h to use in computing $y(5)$, assuming $\delta = 10^{-6}$ and that Eq. (5.14) is valid.

Solution.

- Using Euler's method with

$$\omega_{i+1} = \omega_i + h(-y + hi + 1), \quad i = 1, 2, \dots, \frac{5-0}{h}.$$

By Python, we have

$$y(5) \approx 5.003777893186297 \quad \text{when } h = 0.2;$$

$$y(5) \approx 5.005153775207321 \quad \text{when } h = 0.1;$$

$$y(5) \approx 5.005920529220334 \quad \text{when } h = 0.05;$$

- By assuming (5.14) is true and $\delta = 10^{-6}$, the minimal value of $E(h)$ occurs when

$$h = \sqrt{\frac{2 \cdot 10^{-6}}{\max_{t \in [0,5]} |e^{-t}|}} = \sqrt{2 \cdot 10^{-6}} \approx 1.4142135623731 \times 10^{-3}.$$

□

```

1. def euler_method(f, h, a, b):
2.     y_0 = 1
3.     N = int((b - a) / h)
4.     for i in range(N):
5.         y_0 += h * f(a + h * i, y_0)
6.     return y_0
7.
8. def f_4(t, y):
9.     return -y + t + 1
10.
11. for h in [0.2, 0.1, 0.05]:
12.     print(f"With h = {h}, the approximation is {euler_method(f_4, h, 0, 5)}.")

```

The code for Problem 4a

Problem 5. Use Taylor's method of order two to approximate the solutions for each of the following initial-value problems.

a. $y' = \frac{1+t}{1+y}$, $1 \leq t \leq 2$, $y(1) = 2$ with $h = 0.5$; and

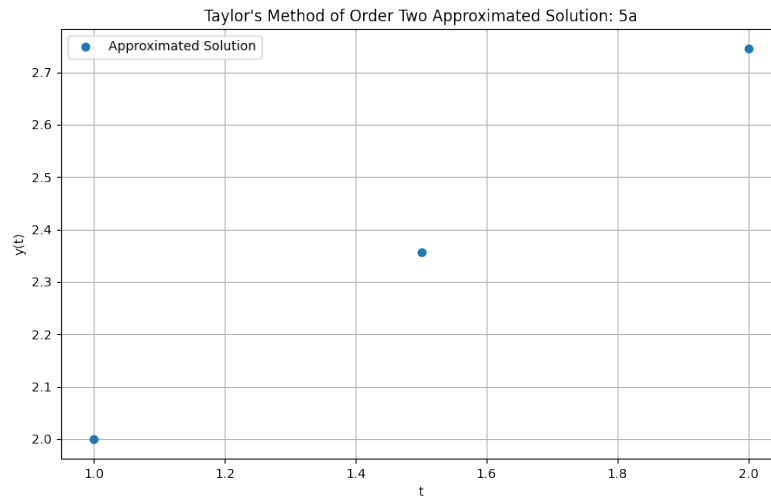
b. $y' = -y + t\sqrt{y}$, $2 \leq t \leq 3$, $y(2) = 2$ with $h = 0.25$.

Solution.

a. By calculus,

$$\frac{d}{dt} \frac{1+t}{1+y} = \frac{1}{1+y} - \frac{(1+t)^2}{(1+y)^3}.$$

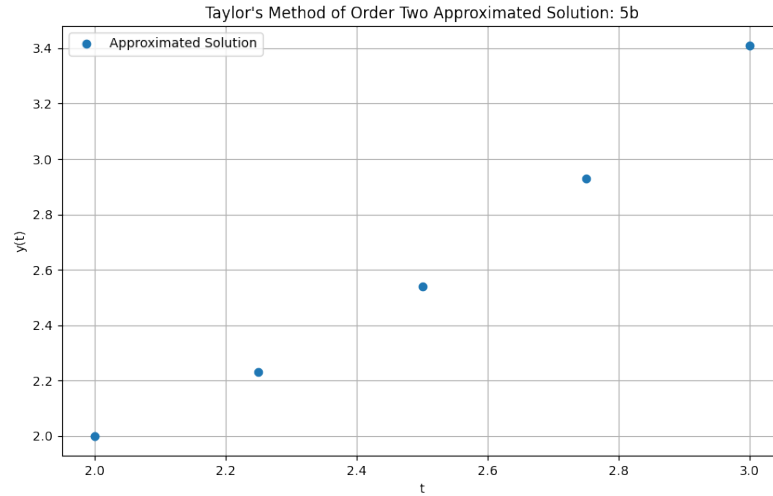
By Python, we have the following result:



b. By calculus,

$$\frac{d}{dt} (-y + t\sqrt{y}) = y + \sqrt{y} - \frac{3}{2}t\sqrt{y} + \frac{t^2}{2}.$$

By Python, we have the following result:



```

1. import math
2. import matplotlib.pyplot as plt
3.
4. def plot_array(x_axis, y_axis, title):
5.     plt.figure(figsize=(10, 6))
6.     plt.plot(x_axis, y_axis, 'o', label='Approximated Solution')
7.     plt.xlabel('t')
8.     plt.ylabel('y(t)')
9.     plt.title(title)
10.    plt.legend()
11.    plt.grid(True)
12.    plt.savefig(f"P{title[-2:]} .png", transparent=True)
13.
14. def taylor_method(n, f_family, alpha, a, b, h):
15.     N = int((b - a) / h)
16.     y_0 = alpha
17.     approx_soln_list = [y_0]
18.     t_values = [a]
19.
20.     for i in range(1, N + 1):
21.         T = 0
22.         for ii in range(n):
23.             T += h ** ii * f_family[ii](t_values[-1], approx_soln_list[-1]) / math.factorial(ii + 1)
24.             approx_soln_list.append(approx_soln_list[-1] + h * T)
25.             t_values.append(a + h * i)
26.
27.     return t_values, approx_soln_list
28.
29. def f_a(t, y):
30.     return (1 + t) / (1 + y)
31.
32. def f_b(t, y):
33.     return -y + t * y ** 0.5
34.
35. def Df_a(t, y):
36.     return 1 / (1 + y) - (1 + t) ** 2 / (1 + y) ** 3
37.
38. def Df_b(t, y):
39.     return y + y * 0.5 - 3 * t * y ** 0.5 / 2 + t ** 2 / 2
40.
41. f_a_family = [f_a, Df_a]
42. f_b_family = [f_b, Df_b]
43.
44.
45. t_values_a, approx_soln_a = taylor_method(2, f_a_family, 2, 1, 2, 0.5)
46.
47. plot_array(t_values_a, approx_soln_a, "Taylor's Method of Order Two Approximated Solution: 5a")
48.
49. t_values_b, approx_soln_b = taylor_method(2, f_b_family, 2, 2, 3, 0.25)
50.
51. plot_array(t_values_b, approx_soln_b, "Taylor's Method of Order Two Approximated Solution: 5b")

```

The code for Problem 5

Problem 6. Given the initial-value problem

$$y' = \frac{2}{t}y + t^2e^t, \quad 1 \leq t \leq 2, \quad y(1) = 0$$

with exact solution $y(t) = t^2(e^t - e)$.

- a. Use Taylor's method of order two with $h = 0.1$ to approximate the solution, and compare it with the actual values of y .
- b. Use the answers generated in part (a) and linear interpolation to approximate y at the following values, and compare them to the actual values of y .
 - i. $y(1.04)$;
 - ii. $y(1.55)$; and
 - iii. $y(1.97)$.
- c. Use Taylor's method of order four with $h = 0.1$ to approximate the solution, and compare it with the actual values of y .
- d. Use the answers generated in part (c) and piecewise cubic Hermite interpolation to approximate y at the following values, and compare them to the actual values of y .
 - i. $y(1.04)$;
 - ii. $y(1.55)$; and
 - iii. $y(1.97)$.

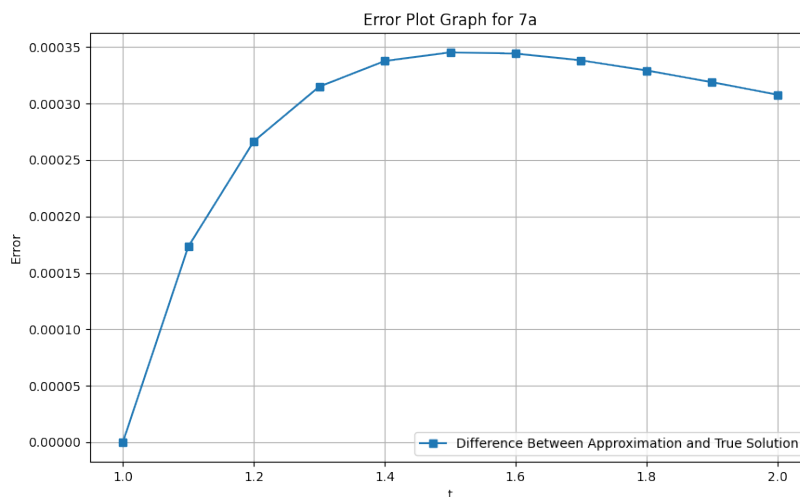
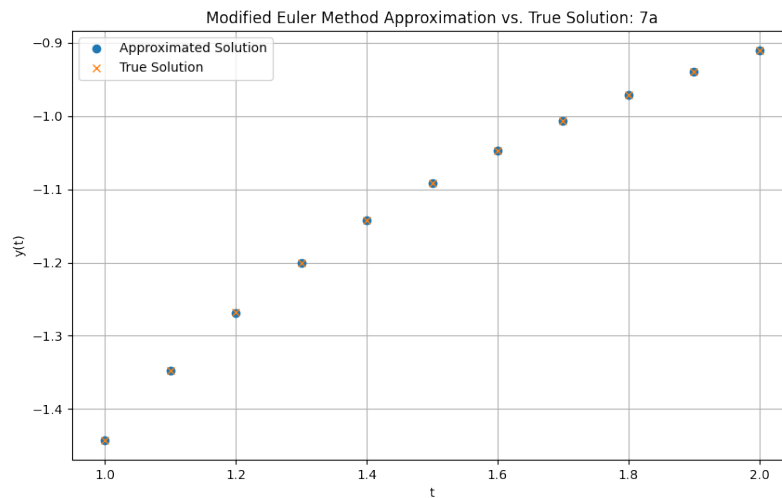
Solution.

Problem 7. Use the modified Euler method to approximate the solutions to each of the following initial-value problems, and compare the results to the actual values.

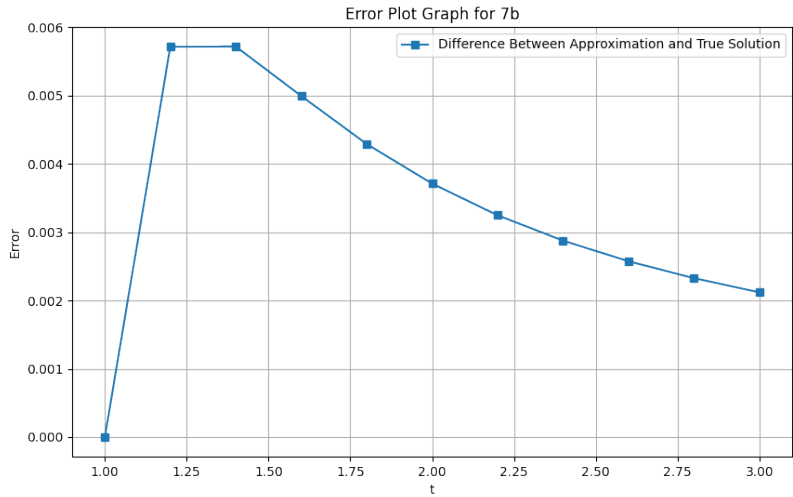
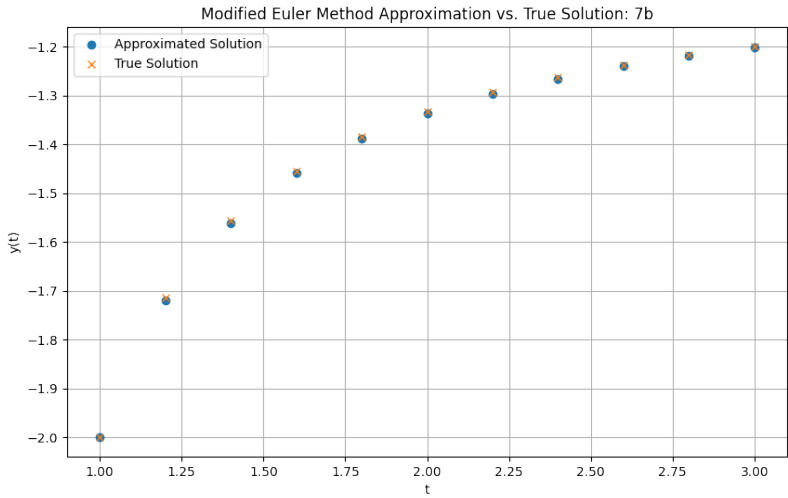
- a. $y' = \frac{y^2}{1+t}$, $1 \leq t \leq 2$, $y(1) = -\frac{1}{\ln 2}$ with $h = 0.1$; actual solution $y = \frac{-1}{\ln(t+1)}$; and
- b. $y' = \frac{y^2 + y}{t}$, $1 \leq t \leq 3$, $y(1) = -2$ with $h = 0.2$; actual solution $y(t) = \frac{2t}{1-2t}$.

Solution. By Python, we have the following results:

- a. The first graph is with approximated solution and the actual solution; the second graph is the error plot graph. In the error plot graph, I used lines to connect each pair of adjacent dots to increase visual clarity.



b. The first graph is with approximated solution and the actual solution; the second graph is the error plot graph. In the error plot graph, I used lines to connect each pair of adjacent dots to increase visual clarity.



□

```

1. import math
2. import matplotlib.pyplot as plt
3.
4. def plot_arrays(x_axis, y_1s, y_2s, title):
5.     plt.figure(figsize=(10, 6))
6.     plt.plot(x_axis, y_1s, 'o', label='Approximated Solution')
7.     plt.plot(x_axis, y_2s, 'x', label='True Solution')
8.     plt.xlabel('t')
9.     plt.ylabel('y(t)')
10.    plt.title(title)
11.    plt.legend()
12.    plt.grid(True)
13.    plt.savefig(f"P{title[-2:]}e.png", transparent=True)
14.
15. def plot_errors(x_axis, y_1s, y_2s, title):
16.     plt.figure(figsize=(10, 6))
17.     y_diff = [abs(y1 - y2) for y1, y2 in zip(y_1s, y_2s)]
18.     plt.plot(x_axis, y_diff, 's-', label='Difference Between Approximation and True Solution')
19.
20.     plt.xlabel('t')
21.     plt.ylabel('Error')
22.     plt.title(title)
23.     plt.legend()
24.     plt.grid(True)
25.     plt.savefig(f"P{title[-2:]}e.png", transparent=True)
26.
27. def modified_euler_method(f, true_y, alpha, a, b, h):
28.     N = int((b - a) / h)
29.     y_0 = alpha
30.     approx_soln_list = [y_0]
31.     real_soln_list = [true_y(a)]
32.     t_values = [a]
33.
34.     for i in range(1, N + 1):
35.         t_i = a + i * h
36.         t_values.append(t_i)
37.         y_0 += h * (f(t_values[-2], y_0) + f(t_i, y_0 + h * f(t_values[-2], y_0))) / 2
38.         approx_soln_list.append(y_0)
39.         real_soln_list.append(true_y(t_i))
40.
41.     return t_values, approx_soln_list, real_soln_list
42.
43. def f_a(t, y):
44.     return y * y / (1 + t)
45.
46. def true_y_a(t):
47.     return -1 / math.log(t + 1)
48.
49. def f_b(t, y):
50.     return (y * y + y) / t
51.
52. def true_y_b(t):
53.     return 2 * t / (1 - 2 * t)
54.
55.
56. t_values_a, approx_soln_a, real_soln_a = modified_euler_method(f_a, true_y_a, -1 / math.log(2), 1, 2, 0.1)
57.
58. plot_arrays(t_values_a, approx_soln_a, real_soln_a, "Modified Euler Method Approximation vs. True Solution: 7a")
59. plot_errors(t_values_a, approx_soln_a, real_soln_a, "Error Plot Graph for 7a")
60.
61. t_values_b, approx_soln_b, real_soln_b = modified_euler_method(f_b, true_y_b, -2, 1, 3, 0.2)
62.
63. plot_arrays(t_values_b, approx_soln_b, real_soln_b, "Modified Euler Method Approximation vs. True Solution: 7b")
64. plot_errors(t_values_b, approx_soln_b, real_soln_b, "Error Plot Graph for 7b")

```

The code for Problem 7

Problem 8. Show that the difference method

$$\omega_0 = \alpha$$

$$\omega_{i+1} = \omega_i + a_1 f(t_i, \omega_i) + a_2 f(t_i + \alpha_2, \omega_i + \delta_2 f(t_i, \omega_i)),$$

for each $i = 0, 1, 2, \dots, N - 1$, cannot have local truncation error $\mathcal{O}(h^3)$ for any choice of constants a_1 , a_2 , α_2 , and δ_2 .

Solution.

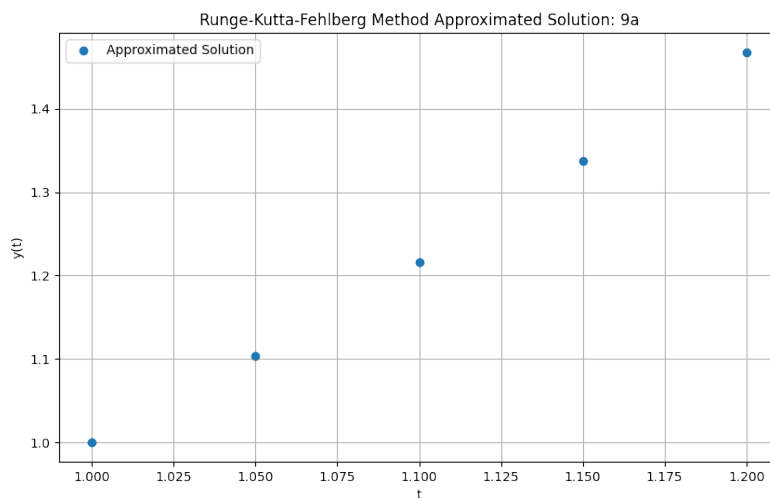
Problem 9. Use the Runge-Kutta-Fehlberg algorithm with tolerance $TOL = 10^{-4}$ to approximate the solution to the following initial-value problems.

a. $y' = \left(\frac{y}{t}\right)^2 + \frac{y}{t}, \quad 1 \leq t \leq 1.2, \quad y(1) = 1$ with $hmax = 0.05$ and $hmin = 0.02$; and

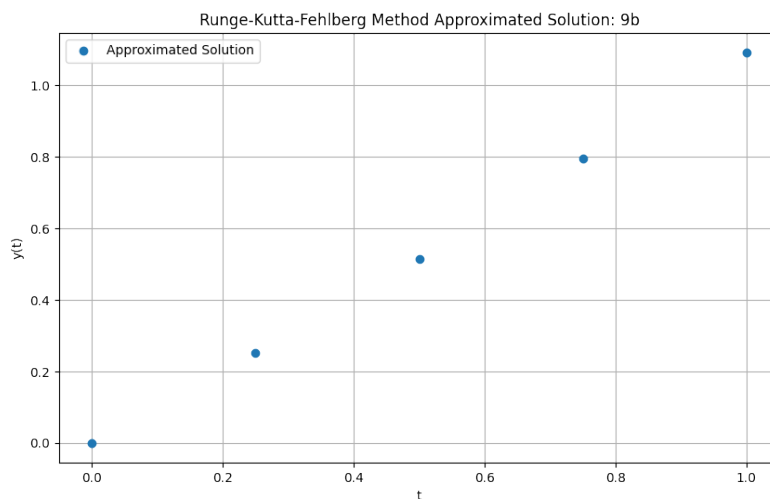
b. $y' = \sin t + e^{-t}, \quad 0 \leq t \leq 1, \quad y(0) = 0$ with $hmax = 0.25$ and $hmin = 0.02$.

Solution. The code is provided after the two graphs.

a.



b.



□


```

1. import math
2. import matplotlib.pyplot as plt
3.
4. def plot_array(x_axis, y_axis, title):
5.     plt.figure(figsize=(10, 6))
6.     plt.plot(x_axis, y_axis, 'o', label='Approximated Solution')
7.     plt.xlabel('t')
8.     plt.ylabel('y(t)')
9.     plt.title(title)
10.    plt.legend()
11.    plt.grid(True)
12.    plt.savefig(f"P{title[-2:]} .png", transparent=True)
13.
14. def runge_kutta_fehlberg_method(f, alpha, TOL, a, b, hmax, hmin):
15.     h = hmax
16.     y_0 = alpha
17.     approx_soln_list = [y_0]
18.     t_values = [a]
19.
20.     FLAG = True
21.
22.     while(FLAG):
23.         t = t_values[-1]
24.         k_1 = h * f(t, y_0)
25.         k_2 = h * f(t + h / 4, y_0 + k_1 / 4)
26.         k_3 = h * f(t + 3 * h / 8, y_0 + 3 * k_1 / 32 + 9 * k_2 / 32)
27.         k_4 = h * f(t + 12 * h / 13, y_0 + 1932 * k_1 / 2197 - 7200 * k_2 / 2197 + 7296 * k_3 / 2197)
28.         k_5 = h * f(t + h, y_0 + 439 * k_1 / 216 - 8 * k_2 + 3680 * k_3 / 513 - 845 * k_4 / 4104)
29.         k_6 = h * f(t + h / 2, y_0 - 8 * k_1 / 27 + 2 * k_2 - 3544 * k_3 / 2565 + 1859 * k_4 / 4104 - 11 * k_5 / 40)
30.
31.         R = abs(k_1 / 360 - 128 * k_3 / 4275 - 2197 * k_4 / 75240 + k_5 / 50 + 2 * k_6 / 55) / h
32.
33.         if R <= TOL:
34.             t += h
35.             t_values.append(t)
36.             y_0 += 25 * k_1 / 216 + 1408 * k_3 / 2565 + 2197 * k_4 / 4104 - k_5 / 5.
37.             approx_soln_list.append(y_0)
38.
39.             delta = 0.84 * pow(TOL / R, 0.25)
40.             if delta <= 0.1:
41.                 h = 0.1 * h
42.             elif delta >= 4:
43.                 h = 4 * h
44.             else:
45.                 h = delta * h
46.
47.             if h > hmax:
48.                 h = hmax
49.
50.             if t >= b:
51.                 FLAG = False
52.             elif t + h > b:
53.                 h = b - t
54.             elif h < hmin:
55.                 FLAG = False
56.                 print("Minimum h exceeded.")
57.
58.     return t_values, approx_soln_list
59.
60. def f_a(t, y):
61.     return y * y / (t * t) + y / t
62.
63. def f_b(t, y):
64.     return math.sin(t) + math.exp(-t)
65.
66. t_values_a, approx_soln_a = runge_kutta_fehlberg_method(f_a, 1, 10 ** -4, 1, 1.2, 0.05, 0.02)
67.
68. print(approx_soln_a)
69. plot_array(t_values_a, approx_soln_a, "Runge-Kutta-Fehlberg Method Approximated Solution: 9a")
70.
71. t_values_b, approx_soln_b = runge_kutta_fehlberg_method(f_b, 0, 10 ** -4, 0, 1, 0.25, 0.02)
72.
73. print(approx_soln_b)
74. plot_array(t_values_b, approx_soln_b, "Runge-Kutta-Fehlberg Method Approximated Solution: 9b")

```

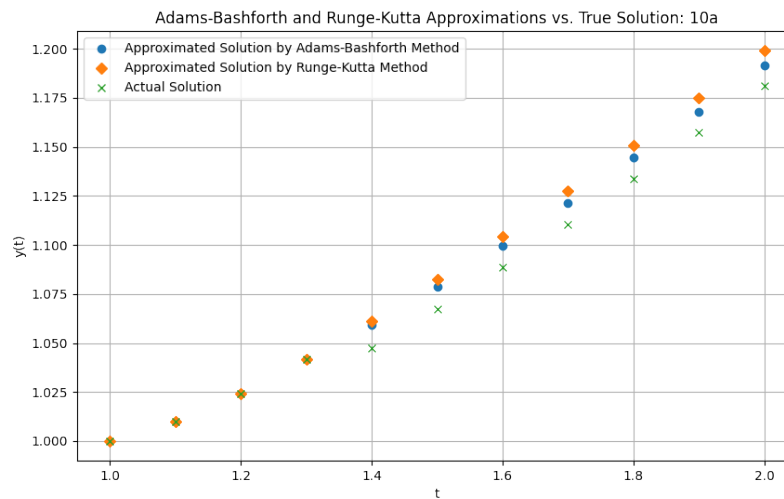
The code for Problem 9

Problem 10. 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.

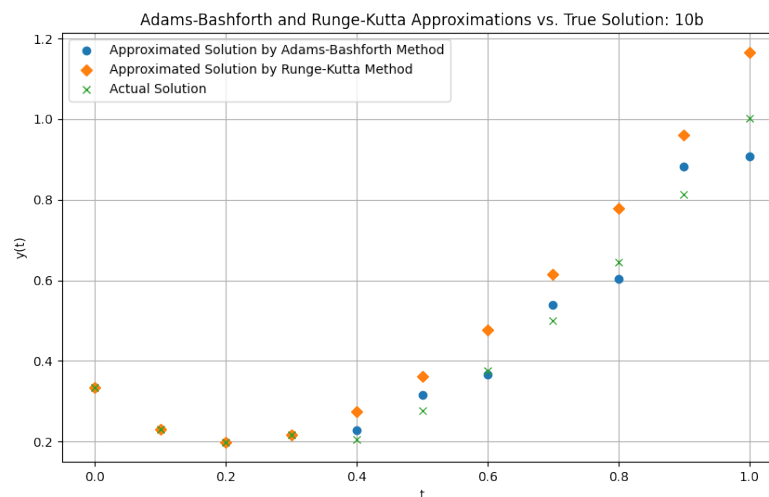
- a. $y' = \frac{y}{t} - \left(\frac{y}{t}\right)^2$, $1 \leq t \leq 2$, $y(1) = 1$ with $h = 0.1$; actual solution $y(t) = \frac{t}{1 + \ln(t)}$; and
- b. $y' = -5y + 5t^2 + 2t$, $0 \leq t \leq 1$, $y(0) = \frac{1}{3}$ with $h = 0.1$; actual solution $y(t) = t^2 + \frac{e^{-5t}}{3}$.

Solution. The code is provided after the two graphs.

a.



b.





```
1. import math
2. import matplotlib.pyplot as plt
3.
4. def plot_arrays(x_axis, y_1s, y_2s, y_3s, title):
5.     plt.figure(figsize=(10, 6))
6.     plt.plot(x_axis, y_1s, 'o', label='Approximated Solution by Adams-Bashforth Method')
7.     plt.plot(x_axis, y_2s, 'D', label='Approximated Solution by Runge-Kutta Method')
8.     plt.plot(x_axis, y_3s, 'x', label='Actual Solution')
9.
10.    plt.xlabel('t')
11.    plt.ylabel('y(t)')
12.    plt.title(title)
13.    plt.legend()
14.    plt.grid(True)
15.    plt.savefig(f"P{title[-3:]} .png", transparent=True)
16.
17. def adams_bashforth_method(f, true_y, alpha, a, b, h):
18.     N = int((b - a) / h)
19.     approx_soln_list = alpha.copy()
20.     real_soln_list = alpha.copy()
21.     t_values = [a, a + h, a + 2 * h, a + 3 * h]
22.
23.     for i in range(4, N + 1):
24.         w = approx_soln_list[-4:]
25.         t = t_values[-4:]
26.         approx_soln_list.append(w[3] + h * (55 * f(t[3], w[3]) - 59 * f(t[2], w[2]) + 37 * f(t[1], w[1]) - 9 * f(t[0], w[0])) / 24)
27.         t_i = a + i * h
28.         t_values.append(t_i)
29.         real_soln_list.append(true_y(t_i))
30.
31.     return t_values, approx_soln_list, real_soln_list
32.
33. def runge_kutta_method(f, true_y, alpha, a, b, h):
34.     N = int((b - a) / h)
35.     approx_soln_list = [alpha]
36.     real_soln_list = [alpha]
37.     t_values = [a]
38.
39.     for i in range(1, N + 1):
40.         t_i = a + i * h
41.         real_soln_list.append(true_y(t_i))
42.         t_values.append(t_i)
43.         if i < 1:
44.             approx_soln_list.append(true_y(t_i))
45.         else:
46.             t = t_values[-1]
47.             w = approx_soln_list[-1]
48.             k_1 = h * f(t, w)
49.             k_2 = h * f(t + h / 2, w + k_1 / 2)
50.             k_3 = h * f(t + h / 2, w + k_2 / 2)
51.             k_4 = h * f(t_i, w + k_3)
52.             approx_soln_list.append(w + (k_1 + 2 * k_2 + 2 * k_3 + k_4) / 6)
53.
54.     return t_values, approx_soln_list, real_soln_list
55.
56. def f_a(t, y):
57.     return y / t - y * y / (t * t)
58.
59. def true_y_a(t):
60.     return t / (1 + math.log(t))
61.
62. def f_b(t, y):
63.     return -5 * y + 5 * t * t + 2 * t
64.
65. def true_y_b(t):
66.     return t * t + math.exp(-5 * t) / 3
67.
68. t_values_a, approx_soln_a_RK, real_soln_a = runge_kutta_method(f_a, true_y_a, 1, 1, 2, 0.1)
69.
70. t_values_a, approx_soln_a_AB, real_soln_a = adams_bashforth_method(f_a, true_y_a, approx_soln_a_RK[0:4], 1, 2, 0.1)
71.
72. plot_arrays(t_values_a, approx_soln_a_AB, approx_soln_a_RK, real_soln_a, "Adams-Bashforth and Runge-Kutta Approximations vs. True Solution: 10a")
73.
74. t_values_b, approx_soln_b_RK, real_soln_b = runge_kutta_method(f_b, true_y_b, 1 / 3, 0, 1, 0.1)
75.
76. t_values_b, approx_soln_b_AB, real_soln_b = adams_bashforth_method(f_b, true_y_b, approx_soln_b_RK[0:4], 0, 1, 0.1)
77.
78. plot_arrays(t_values_b, approx_soln_b_AB, approx_soln_b_RK, real_soln_b, "Adams-Bashforth and Runge-Kutta Approximations vs. True Solution: 10b")
```

The code for Problem 10

Problem 11. The initial-value problem

$$y' = e^y, \quad 0 \leq t \leq 0.2, \quad y(0) = 1$$

has solution $y(t) = 1 - \ln(1 - et)$. Applying the three-step Adams-Moulton method to this problem is equivalent to finding the fixed point ω_{i+1} of

$$g(\omega) = \omega_i + \frac{h}{24} (9e^\omega + 19e^{\omega_i} - 5e^{\omega_{i-1}} + e^{\omega_{i-2}}).$$

- a. With $h = 0.01$, obtain ω_{i+1} by functional iteration for $i = 2, \dots, 19$ using exact starting values ω_0 , ω_1 , and ω_2 . At each step use ω_i to initially approximate ω_{i+1} .
- b. Will Newton's method speed the convergence over functional iteration?

Solution.

Problem 12. Derive the Adams-Bashforth three-step method by the following method. Set

$$y(t_{i+1}) = t(t_i) + ah f(t_i, y(t_i)) + bh f(t_{i-1}, y(t_{i-1})) + ch f(t_{i-2}, y(t_{i-2})).$$

Expand $y(t_{i+1})$, $f(t_{i-2}, y(t_{i-2}))$, and $f(t_{i-1}, y(t_{i-1}))$ in Taylor series about $(t_i, y(t_i))$, and equate the coefficients of h , h^2 , and h^3 to obtain a , b , and c .

Solution.

Problem 13. Use the Adams variable step-size predictor-corrector algorithm with $TOL = 10^{-4}$ to approximate the solutions to the following initial-value problems:

a. $y' = \sin t + e^{-t}$, $0 \leq t \leq 1$, $y(0) = 0$ with $hmax = 0.2$ and $hmin = 0.01$; and

b. $y' = -ty + \frac{4t}{y}$, $0 \leq t \leq 1$, $y(0) = 1$ with $hmax = 0.2$ and $hmin = 0.01$.

Solution.

Problem 14. Let $P(t)$ be the number of individuals in a population at time t , measured in years. If the average birth rate b is constant and the average death rate d is proportional to the size of the population (due to overcrowding), then the growth rate of the population is given by the logistic equation

$$\frac{dP}{dt}(t) = b P(t) - k(P(t))^2,$$

where $d = k P(t)$. Suppose $P(0) = 50976$, $b = 2.9 \times 10^{-2}$, and $k = 1.4 \times 10^{-7}$. Find the population after 5 years using the extrapolation method (based on the Euler method and the midpoint method) with times step $h = 0.1$. Justify the order of truncation error from your numerical answers.

Solution.

Problem 15. Suppose the swinging pendulum described in the lead example of this chapter is 2 ft long and that $g = 32.17 \text{ ft/s}^2$. With $h = 0.1 \text{ s}$, compare the angle θ obtained for the following two initial-value problems at $t = 0, 1, 2$.^a

$$\frac{d^2\theta}{dt^2} + \frac{g}{L} \sin \theta = 0, \quad \theta(0) = \frac{\pi}{6}, \quad \theta'(0) = 0.$$

You shall use Adams fourth-order predictor-corrector algorithm to obtain your numerical answer.

^aI read this as “find values of $\theta(1)$ and $\theta(2)$ given $\theta(0) = \pi/6$.”

Solution.

Problem 16. Consider the differential equation

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha.$$

a. Show that

$$y'(t_i) = \frac{-3y(t_i) + 4y(t_{i+1}) - y(t_{i+2}))}{2h} + \frac{h^2}{3}y'''(\xi_i)$$

for some $\xi \in (t_i, t_{i+2})$.

b. Part (a) suggests the difference method

$$\omega_{i+2} = 4\omega_{i+1} - 3\omega_i - 2h f(t_i, \omega_i), \quad \text{for } i = 0, 1, 2, \dots, N-2.$$

Use this method to solve

$$y' = 1 - y, \quad 0 \leq t \leq 1, \quad y(0) = 0$$

with $h = 0.1$. Use the starting values $\omega_0 = 0$ and $\omega_1 = y(t_1) = 1 - e^{-0.1}$.

c. Repeat part (b) with $h = 0.01$ and $\omega_1 = 1 - e^{-0.01}$.

d. Analyze this method for consistency, stability, and convergence.

Solution.

Problem 17. Given the multistep method

$$\omega_{i+1} = -\frac{3}{2}\omega_i + 3\omega_{i-1} - \frac{1}{2}\omega_{i-2} + 3g f(t_i, \omega_i), \quad \text{for } i = 2, 3, \dots, N-1$$

with starting values ω_0 , ω_1 , and ω_2 :

- a. Find the local truncation error.
- b. Comment on consistency, stability, and convergence.

Solution.

Problem 18. Discuss consistency, stability, and convergence for the implicit trapezoidal method

$$\omega_{i+1} = \omega_i + \frac{h}{2} (f(t_{i+1}, \omega_{i+1}) + f(t_i, \omega_i)), \quad \text{for } i = 0, 1, 2, \dots, N-1$$

with $\omega_0 = \alpha$ applied to the differential equation

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha.$$

Solution.

Problem 19. Show that the fourth-order Runge-Kutta method,

$$k_1 = h f(t_i, \omega_i),$$

$$k_2 = h f(t_i + h/2, \omega_i + k_1/2),$$

$$k_3 = h f(t_i + h/2, \omega_i + k_2/2),$$

$$k_4 = h f(t_i + h, \omega_i + k_3),$$

$$\omega_{i+1} = \omega_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

when applied to the differential equation $y' = \lambda y$, can be written in the form

$$\omega_{i+1} = \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4 \right) \omega_i.$$

Solution. We have $f(t, w) = \lambda w$. Hence, we have

$$k_1 = h\lambda\omega_i,$$

$$k_2 = h\lambda(\omega_i + k_1/2)$$

$$= h\lambda\omega_i + (h\lambda)^2\omega_i/2,$$

$$k_3 = h\lambda(\omega_i + k_2/2)$$

$$= h\lambda\omega_i + h\lambda(h\lambda\omega_i + (h\lambda)^2\omega_i/2)/2$$

$$= h\lambda\omega_i + (h\lambda)^2\omega_i/2 + (h\lambda)^3\omega_i/4,$$

$$k_4 = h\lambda(\omega_i + k_3)$$

$$= h\lambda\omega_i + (h\lambda)^2\omega_i + (h\lambda)^3\omega_i/2 + (h\lambda)^4\omega_i/4.$$

Therefore,

$$\begin{aligned}\omega_{i+1} &= \omega_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ &= \omega_i + \frac{1}{6}[(h\lambda\omega_i) \\ &\quad + 2(h\lambda\omega_i + (h\lambda)^2\omega_i/2) \\ &\quad + 2(h\lambda\omega_i + (h\lambda)^2\omega_i/2 + (h\lambda)^3\omega_i/4) \\ &\quad + (h\lambda\omega_i + (h\lambda)^2\omega_i + (h\lambda)^3\omega_i/2 + (h\lambda)^4\omega_i/4)] \\ &= \omega_i + \frac{1}{6}\left(6 \cdot h\lambda\omega_i + 3 \cdot (h\lambda)^2\omega_i + 1 \cdot (h\lambda)^3\omega_i + \frac{1}{4} \cdot (h\lambda)^4\omega_i\right) \\ &= \omega_i \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \frac{1}{6}(h\lambda)^3 + \frac{1}{24}(h\lambda)^4\right).\end{aligned}$$

□