

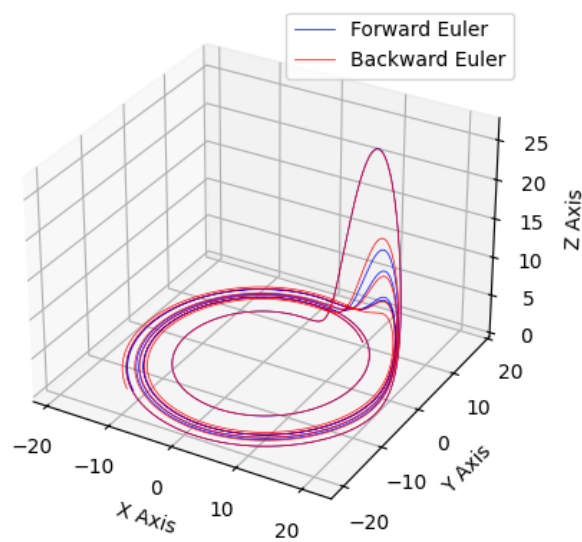
MATH 417 502 HW6

Keegan Smith

October 20, 2024

Problem 1

for $h=.0004$:



I found that a very small step size $h=.00004$ causes backward and forward euler to be nearly equivalent. When h is large, forward euler blows up. Backward euler does not seem to blow up as quickly.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4
5 def plot_rossler_attractor(results, title):
6     x = [result[0] for result in results]
7     y = [result[1] for result in results]
8     z = [result[2] for result in results]
9
10    fig = plt.figure()
11    ax = fig.add_subplot(111, projection='3d')
12    ax.plot(x, y, z, lw=0.5)
13
14    ax.set_xlabel("X Axis")
15    ax.set_ylabel("Y Axis")
16    ax.set_zlabel("Z Axis")
17    ax.set_title(title)
18    plt.savefig("figs/" + title + ".png")
19 def plot_rossler_both(backward_results, forward_results):
20     x_f = [result[0] for result in forward_results]
21     y_f = [result[1] for result in forward_results]
22     z_f = [result[2] for result in forward_results]
23
24     x_b = [result[0] for result in backward_results]
25     y_b = [result[1] for result in backward_results]
26     z_b = [result[2] for result in backward_results]
27
28     fig = plt.figure()
29     ax = fig.add_subplot(111, projection='3d')
30
31     ax.plot(x_f, y_f, z_f, lw=0.5, color='blue', label='
        Forward Euler')
32
33     ax.plot(x_b, y_b, z_b, lw=0.5, color='red', label='
        Backward Euler')
34
35     ax.set_xlabel("X Axis")
36     ax.set_ylabel("Y Axis")
37     ax.set_zlabel("Z Axis")
38
39     ax.legend()
40     plt.savefig("figs/both.png")
41     #plt.show()
42 def function_a(y_n):
43     a = .1
44     b = .1
```

```

45     c = 14
46     if(len(y_n) != 3):
47         raise RuntimeError("input vector for function a does
                                not equal 3")
48     result = [0.0] * 3;
49     result[0] = -y_n[1] - y_n[2];
50     result[1] = y_n[0] + a * y_n[1]
51     result[2] = b + y_n[2] * (y_n[0] - c);
52     return result;
53 def jacobian_a(y_n):
54     a = .1
55     b = .1
56     c = 14
57     if(len(y_n) != 3):
58         raise RuntimeError("input vector for jacobian a does
                                not equal 3")
59     result = [
60         [-1, -1, 0],
61         [1, a, 0],
62         [y_n[2], 0, y_n[0] - c]
63     ]
64     return result;
65 def G(y_n:list, h: float, f, z: list):
66     function_result = f(z);
67     if(len(function_result) != len(z) or len(function_result)
        != len(y_n)):
68         raise RuntimeError("vector sizes do not match in
                                function G")
69     result = [0.0] * len(y_n)
70     for i in range(0, len(y_n)):
71         result[i] = y_n[i] + h * function_result[i] - z[i]
72     return result;
73 def G_jacobian(y_n: list, h: float, f_prime, z: list):
74     function_result = f_prime(z);
75     for i in range(0, len(function_result)):
76         for j in range(0, len(function_result[i])):
77             function_result[i][j] *= h;
78             if(i == j):
79                 function_result[i][j] -= 1
80     return function_result;
81
82 def forward_euler_iteration(y_n : list, h: float, f):
83     function_result = f(y_n)
84     if(len(function_result) != len(y_n)):
85         raise RuntimeError("vector sizes do not match in
                                forward euler.")
86     result = [0.0] * len(y_n);
87     for i in range(0, len(y_n)):
88         result[i] = y_n[i] + h * function_result[i]
89     return result;

```

```

90 def run_forward_euler(y_n : list, h: float, f,
    num_iterations: int):
91     results = [y_n]
92     for i in range(0, num_iterations):
93         y_n = forward_euler_iteration(y_n, h, f)
94         results.append(y_n);
95     return results;
96 def backward_euler_iteration(y_n: list, h: float, f, f_prime
    , z: list):
97     jacobian = G_jacobian(y_n, h, f_prime, z)
98     g_vector = G(y_n, h, f, z)
99     product = np.linalg.solve(jacobian, g_vector)
100    next_z = [0.0] * len(product)
101    for i in range(0, len(product)):
102        next_z[i] = z[i] - product[i]
103    result = [0.0] * len(product)
104    function_result = f(next_z)
105    for i in range(0, len(product)):
106        result[i] = y_n[i] + h * function_result[i]
107    return result, next_z
108
109 def run_backward_euler(y_n: list, h: float, f, f_prime,
    num_iterations: int):
110    z = y_n;
111    results = []
112    for i in range(0, num_iterations):
113        y_n, z = backward_euler_iteration(y_n, h, f, f_prime
            , z)
114        results.append(y_n)
115    return results
116
117 def main():
118     num_iterations = 100000;
119     y_0 = [10.0, 10.0, 0.0]
120     time_interval = [0, 40]
121     h = (time_interval[1] - time_interval[0]) /
        num_iterations
122     print("h is: ", h)
123     print("Performing forward euler:")
124     forward_euler_results = run_forward_euler(y_0, h,
        function_a, num_iterations)
125     #print(forward_euler_results)
126     plot_rossler_attractor(forward_euler_results, "forward
        euler")
127
128     print("Performing backward euler:")
129     y_0 = [10.0, 10.0, 0.0]
130     backward_euler_results = run_backward_euler(y_0, h,
        function_a, jacobian_a, num_iterations)
131     #print(backward_euler_results)

```

```

132     plot_rossler_attractor(backward_euler_results, "backward
        euler")
133     plot_rossler_both(backward_euler_results,
        forward_euler_results)
134 if __name__ == "__main__":
135     main();

```

Problem 2

1. We want a system of equations such that $\frac{d}{dt}y_n(t) = y_{n+1}(t)$. Thus we will have:

$$\begin{aligned}
 y_1(t) &= y(t) \\
 y_2(t) &= y'(t) \\
 y_3(t) &= y''(t) \\
 y_4(t) &= y'''(t)
 \end{aligned}$$

Plugging in the values from the given equation:

$$\begin{aligned}
 y_4'(t) &= t^2 - 3 \cdot y_3(t) + \sin(t) \cdot y_2(t) - 8 \cdot y_1(t) \\
 y_3'(t) &= y_4(t) \\
 y_2'(t) &= y_3(t) \\
 y_1'(t) &= y_2(t)
 \end{aligned}$$

Thus we have $y'(t) = f(t, x)$ where:

$$f(t, x) = \begin{cases} x_2(t) \\ x_3(t) \\ x_4(t) \\ t^2 - 3 \cdot x_3(t) + \sin(t) \cdot x_2(t) - 8 \cdot x_1(t) \end{cases}$$

2. For backward euler we have:

$$y_{n+1} = y_n + h \cdot f(t_{n+1}, y_{n+1})$$

where f is the same function as we derived above.

To find the y_{n+1} on the rightside we will use newton iteration:

$$\begin{aligned}
 0 &= y_n + h \cdot f(t_{n+1}, y_{n+1}) - y_{n+1} \\
 0 &= y_n + h \cdot f(t_{n+1}, z) - z \quad \text{Substituting } y_{n+1} \text{ with } z
 \end{aligned}$$

Thus we are finding the root of the function:

$$G(z) = y_n + h \cdot f(t_{n+1}, z) - z$$

$$G(z) = \begin{bmatrix} y_1 + h \cdot z_2(t) - z_1(t) \\ y_2 + h \cdot z_3(t) - z_2(t) \\ y_3 + h \cdot z_4(t) - z_3(t) \\ y_4 + h \cdot t^2 - 3 \cdot z_3(t) + \sin(t) \cdot z_2(t) - 8 \cdot z_1(t) - z_4(t) \end{bmatrix}$$

Recall the newton scheme is given by:

$$y_{n+1} = y_n - (J^{-1}F(z))F(z)$$

The jacobian of $G(z)$ is:

$$JG(z) = \begin{bmatrix} -1 & h & 0 & 0 \\ 0 & -1 & h & 0 \\ 0 & 0 & -1 & h \\ -8 & \sin(t) & -3 & -1 \end{bmatrix}$$

Thus we have:

$$z_{n+1} = y_n - \begin{bmatrix} -1 & h & 0 & 0 \\ 0 & -1 & h & 0 \\ 0 & 0 & -1 & h \\ -8 & \sin(t) & -3 & -1 \end{bmatrix}^{-1} \begin{bmatrix} y_1 + h \cdot z_2(t) - z_1(t) \\ y_2 + h \cdot z_3(t) - z_2(t) \\ y_3 + h \cdot z_4(t) - z_3(t) \\ y_4 + h \cdot t^2 - 3 \cdot z_3(t) + \sin(t) \cdot z_2(t) - 8 \cdot z_1(t) - z_4(t) \end{bmatrix}$$

And whatever we get for z_{n+1} we plug into y_{n+1} on the rightside of the original backward euler scheme. We can then evaluate that equation to find the true y_{n+1} .