

MATH 151B Homework 3

Zheng Wang (404855295)

May 22, 2019

Question 1

(a) The Euler's Method gives

$$\begin{cases} w_{i+1} = w_i + h \cdot f(x_i, w_i) \\ w_0 = 0 \end{cases}$$

Since $x_i = ih$, we then break down the original formula to

$$\begin{cases} w_{i+1} = w_i + h(ih - (ih)^2) \\ w_0 = 0 \end{cases}$$

Therefore, we have

$$\begin{aligned} w_{i+1} &= w_i + h \cdot (ih - (ih)^2) \\ &= w_{i-1} + h \cdot ((i-1)h - (i-1)^2h^2) + h \cdot (ih - (ih)^2) \\ &= w_{i-1} + (i-1) \cdot h^2 - (i-1)^2 \cdot h^3 + i \cdot h^2 - i^2 \cdot h^3 \\ &= w_{i-1} + h^2 \cdot [i + (i-1)] - h^3 \cdot [i^2 + (i-1)^2] \\ &= w_0 + h^2 \cdot \left(\sum_{j=0}^i j \right) - h^3 \cdot \left(\sum_{j=0}^i j^2 \right) \\ &= h^2 \cdot \frac{i \cdot (i+1)}{2} - h^3 \cdot \frac{i(i+1)(2i+1)}{6} \end{aligned} \quad \text{(By telescoping)}$$

Therefore, $\boxed{w_i = h^2 \cdot \frac{i(i-1)}{2} - h^3 \cdot \frac{i(i-1)(2i-1)}{6}}$

■

(b) We take $x = ih$, where x is an arbitrary fixed point

Then we have the following:

$$\begin{aligned}
 \lim_{h \rightarrow 0} |w_i - y(x)| &= \lim_{h \rightarrow 0} \left| \frac{i(i-1)}{2} \cdot h^2 - \frac{i(i-1)(2i-1)}{6} \cdot h^3 - \frac{i^2}{2} \cdot h^2 + \frac{i^3}{3} \cdot h^3 \right| \\
 &= \lim_{h \rightarrow 0} \left| \frac{i^2 - i - i^2}{2} \cdot h^2 + \frac{-2i^3 + 3i^2 - i + 2i^3}{6} \cdot h^3 \right| \\
 &= \lim_{h \rightarrow 0} \left| \frac{-ih}{2} \cdot h + \frac{(ih)^2}{2} \cdot h - \frac{ih}{6} \cdot h^2 \right| \\
 &= \lim_{h \rightarrow 0} \left| -\frac{x}{2} \cdot h + \frac{x^2}{2} \cdot h - \frac{x}{6} \cdot h^2 \right| \\
 &= \left| \lim_{h \rightarrow 0} \left(-\frac{x}{2} \cdot h + \frac{x^2}{2} \cdot h - \frac{x}{6} \cdot h^2 \right) \right| && \text{(By continuity)} \\
 &= 0 && \text{(Since } x \text{ is a constant)}
 \end{aligned}$$

Thus, Euler's method is convergent. ■

Question 2

The RKF12 method is implemented with the following code.

```

1 % Math 151b
2 % Homework 3, Question 2
3
4 % run the program with TOL = 10^-4
5 disp(run_rkf12(0, 1, 1, 10^-4, 0.5, 10^-7, @f))
6
7 % a demo function, a = 0, b = 1, alpha = 1
8 function dydt = f(t,y)
9     dydt = y^2*exp(-t);
10 end
11
12 % Implement RKF12 method with Euler's method and Modified Euler Method
13 % INPUTS:
14 % a,b - endpoints; alpha - initial condition; TOL - tolerance
15 % hmax - maximum step size; hmin - min step size
16 % func - function to be solved
17 function y = run_rkf12(a, b, alpha, TOL, hmax, hmin, func)
18     t = a;
19     w = alpha;
20     h = hmax;
21     FLAG = 1;
22     disp([t, w])
23     while FLAG == 1
24         K1 = h * func(t,w);
25         K2 = func(t + h, w + K1);
26         K3 = h/2 * ( func(t,w) + K2 );
27         % Euler: w_i+1 = w_i + K1

```

```

28 % M_Euler: w_i+1 = w_i + K3
29 R = 1/h * abs(K3 - K1);
30 if R <= TOL
31     t = t + h;
32     w = w + K1;
33     disp([t,w,h]);
34 end
35 q = (1/2) * (TOL/R);
36 % Adjust the step size
37 if q <= 0.1
38     h = 0.1 * h; % prevent delta to be too small and h goes to 0
39 elseif q >= 4
40     h = 4 * h; % prevent delta become to large and h increase to fast
41 else
42     h = q * h; % normal case, just set h = qh
43 end
44 % bound h by hmax
45 if h > hmax
46     h = hmax;
47 end
48 % terminating conditions
49 if t >= b
50     FLAG = 0; % reach the end point
51 elseif t + h > b
52     h = b - t; % adjust the final step
53 elseif h < hmin
54     FLAG = 0;
55     fprintf('Minimun h exceeded, Procedure completed unsuccessfully.')
56 end
57 end
58 y = w;
59 end

```

Explanation:

From formula of Modified Euler Method and Euler Method, we have the following:

$$\begin{aligned}\tilde{w}_{i+1} &= w_i + \frac{h}{2} [f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i))] \\ w_{i+1} &= w_i + h \cdot f(t_i, w_i)\end{aligned}$$

Then, we compute τ_{i+1} and $\tilde{\tau}_{i+1}$:

$$\begin{aligned}\tau_{i+1} &= \frac{y(t_{i+1}) - w_i}{h} - f(t_i, w_i) \\ &= \frac{y(t_{i+1}) - (w_i + hf(t_i, w_i))}{h} = \frac{1}{h} (y(t_{i+1}) - w_{i+1}) \\ \tilde{\tau}_{i+1} &= \frac{y(t_{i+1}) - (w_i + \frac{h}{2} [f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i))])}{h} \\ &= \frac{1}{h} (y(t_{i+1}) - \tilde{w}_{i+1})\end{aligned}$$

Next, we arrive at the following:

$$\begin{aligned}\tau_{i+1} &= \frac{1}{h} (y(t_{i+1}) - w_{i+1}) = \frac{1}{h} [(y(t_{i+1}) - \tilde{w}_{i+1}) + (\tilde{w}_{i+1} - w_{i+1})] \\ &= \tilde{\tau}_{i+1} + \frac{1}{h} (\tilde{w}_{i+1} - w_{i+1})\end{aligned}$$

As $\tilde{\tau}_{i+1}$ is $\mathcal{O}(h^2)$, but τ_{i+1} is $\mathcal{O}(h)$. Thus, $\frac{1}{h}(\tilde{w}_{i+1} - w_{i+1})$ is $\mathcal{O}(h)$. Therefore, $\tau_{i+1} \approx \frac{1}{h}(\tilde{w}_{i+1} - w_{i+1})$. With this equation, we can estimate the local truncation error as:

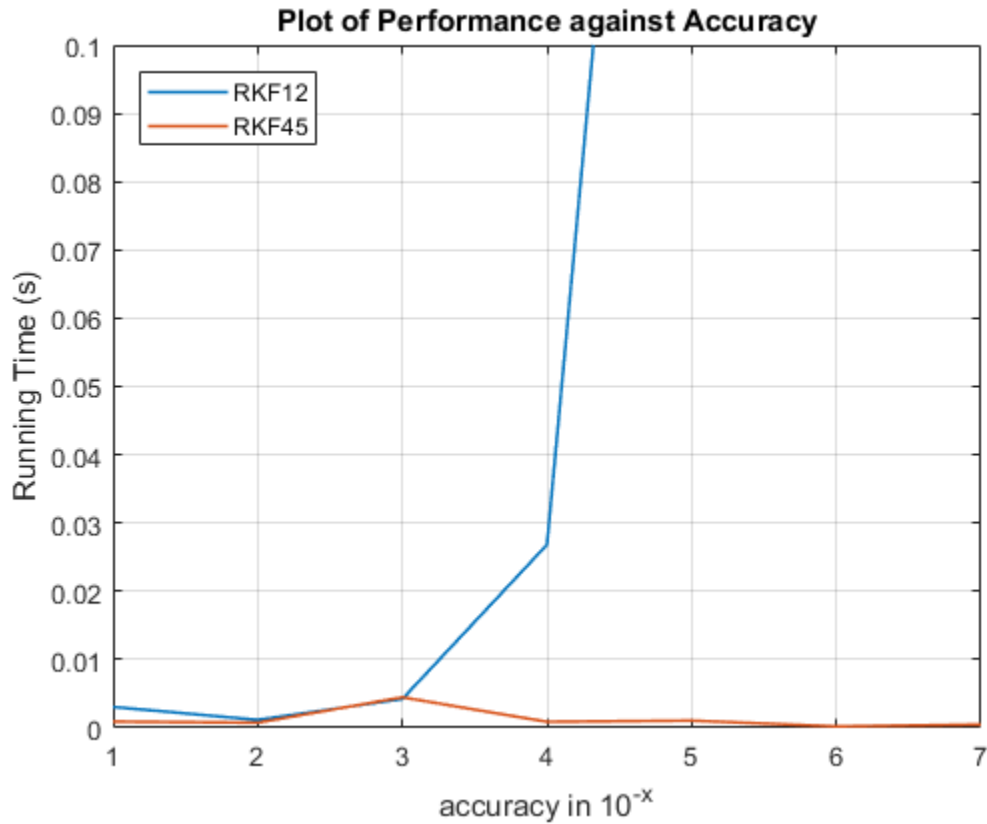
$$R = \frac{1}{h} |\tilde{w}_{i+1} - w_{i+1}| \approx \tau_{i+1}$$

In our program, since we have $\tilde{w}_{i+1} = w_i + K3$ and $w_{i+1} = w_i + K1$, Therefore, we have $R = \frac{1}{h} |K3 - K1|$. Finally, by taking $\tau_{i+1}(h) \approx Kh$, by solving $\tau_{i+1}(qh) \approx Kqh \approx q\tau_{i+1}(h) \approx qR \leq \epsilon$, we get $q \leq \frac{\epsilon}{R}$. To be more conservative, we make q even smaller by let $q = \frac{\epsilon}{2R}$. Since in the program *TOL* represent ϵ , we use $q = TOL/2R$ in the program.

After these, to make sure the q is not too small nor too large, we also bound q with 0.1 and 4. To make sure that h is reasonable, we also fix $hmin \leq h \leq hmax$ in the program. From the demo I did in the program, the final estimation is 2.7181, which is close to the real solution 2.7183 to degree 10^{-4} .

In terms of the run-time of the RKF12 and RKF45 method, when the accuracy is low (like 10^{-2}), the two method will have similar performance, sometimes RKF12 can run faster than RKF45. This is because we have fixed the maximum step size in the program, and even though RKF45 can achieve the desired accuracy level with large step size, the program with still have to run the a minimum number of steps. Moreover, since RKF45 must take more calculations per iteration than RKF12, it might have very similar or even lower performance than RKF12.

But when accuracy is fixed high (like 10^{-7}), the RKF45 can run much faster than RKF12. This is because to achieve the desired accuracy level, RKF12 will need to take much smaller step size than RKF45. Even though RKF45 needs more evaluation per iteration, there are much less iterations to run. Therefore, it is much faster than RKF12. The general trend in the performance is given by the following plot:



The following code is used to do the test:

```

1 % Math 151b
2 % Homework 3, Question 2
3 time_rkf12 = zeros(1,7);
4 time_rkf45 = zeros(1,7);
5
6 for i=1:7
7     tic
8     disp(run_rkf12(0, 1, 1, 10^-i, 0.5, 10^-10, @f))
9     time_rkf12(i) = toc;
10
11     tic
12     disp(run_rkf45(0, 1, 1, 10^-i, 0.5, 10^-10, @f))
13     time_rkf45(i) = toc;
14 end
15 % make the plot
16 figure;
17 plot((1:7), time_rkf12, 'Linewidth', 1.1);
18 hold on;
19 plot((1:7), time_rkf45, 'Linewidth', 1.1);
20 ylim([0,0.1]);
21 xlabel('accuracy in 10^{-x}');
22 ylabel('Running Time (s)');
23 legend({'RKf12', 'RKf45'}, 'Location', 'northwest')

```

```

24 title('Plot of Performance against Accuracy');
25 grid on;
26 hold off;
27
28 % a demo function , a = 0, b = 1, alpha = 1
29 function dydt = f(t,y)
30     dydt = y^2*exp(-t);
31 end
32
33 % Implement RKF12 method with Euler's method and Modified Euler Method
34 % INPUTS:
35 % a,b - endpoints; alpha - initial condition; TOL - tolerance
36 % hmax - maximum step size; hmin - min step size
37 % func - function to be solved
38 function y = run_rkf12(a, b, alpha, TOL, hmax, hmin, func)
39     t = a;
40     w = alpha;
41     h = hmax;
42     FLAG = 1;
43     %disp([t, w])
44     while FLAG == 1
45         K1 = h * func(t,w);
46         K2 = func(t + h, w + K1);
47         K3 = h/2 * ( func(t,w) + K2 );
48         % Euler: w_i+1 = w_i + K1
49         % M_Euler: w_i+1 = w_i + K3
50         R = 1/h * abs(K3 - K1);
51         if R <= TOL
52             t = t + h;
53             w = w + K1;
54             %disp([t,w,h]);
55         end
56         q = (1/2) * (TOL/R);
57         % Adjust the step size
58         if q <= 0.1
59             h = 0.1 * h; % prevent delta to be too small and h goes to 0
60         elseif q >= 4
61             h = 4 * h; % prevent delta become to large and h increase to fast
62         else
63             h = q * h; % normal case , just set h = qh
64         end
65         % bound h by hmax
66         if h > hmax
67             h = hmax;
68         end
69         % terminating conditions
70         if t >= b
71             FLAG = 0; % reach the end point
72         elseif t + h > b
73             h = b - t; % adjust the final step
74         elseif h < hmin
75             FLAG = 0;
76             fprintf("Minimun h exceeded , Procedure completed unsuccessfully.")
77         end
78     end
79     y = w;

```

```

80 end
81
82
83 % Implement RKF45 method
84 % INPUTS:
85 % a,b – endpoints; alpha – initial condition; TOL – tolerance
86 % hmax – maximum step size; hmin – min step size
87 % func – function to be solved
88 function y = run_rkf45(a, b, alpha, TOL, hmax, hmin, func)
89     t = a;
90     w = alpha;
91     h = hmax;
92     FLAG = 1;
93     %disp([t, w])
94     while FLAG == 1
95         K1 = h * func(t,w);
96         K2 = h * func(t + h/4, w + K1/4);
97         K3 = h * func(t + 3/8*h, w + 3/32*K1 + 9/32*K2 );
98         K4 = h * func(t + 12/13*h, w + 1932/2197*K1 - 7200/2197*K2 + 7296/2197*K3);
99         K5 = h * func(t + h, w + 439/216*K1 - 8*K2 + 3680/513*K3 - 845/4104*K4);
100        K6 = h * func(t + h/2, w - 8/27*K1 + 2*K2 - 3544/2565*K3 + 1859/4104*K4 -
            11/40*K5);
101        R = 1/h * abs(1/360*K1 - 128/4275*K3 - 2197/75240*K4 + 1/50*K5 + 2/55*K6);
102        if R <= TOL
103            t = t + h;
104            w = w + 25/216*K1 + 1408/2565*K3 + 2197/4104*K4 - 1/5*K5;
105            %disp([t,w,h]);
106        end
107        q = 0.84 * (TOL/R)^(1/4);
108        % Adjust the step size
109        if q <= 0.1
110            h = 0.1 * h; % prevent delta to be too small and h goes to 0
111        elseif q >= 4
112            h = 4 * h; % prevent delta become to large and h increase to fast
113        else
114            h = q * h; % normal case, just set h = qh
115        end
116        % bound h by hmax
117        if h > hmax
118            h = hmax;
119        end
120        % terminating conditions
121        if t >= b
122            FLAG = 0; % reach the end point
123        elseif t + h > b
124            h = b - t; % adjust the final step
125        elseif h < hmin
126            FLAG = 0;
127            fprintf("Minimun h exceeded, Procedure completed unsuccessfully.")
128        end
129    end
130    y = w;
131 end

```