

1 Introduction

We solve the following problems to better understand numerical techniques for solving ordinary differential equations. As will be described in the methods section, our tools primarily consist of second- and fourth-order Runge-Kutta methods, and Euler stability analysis.

1.1 Problem 1

Equations of motion for a rocket's vertical speed v can be written as

$$(m_c + m_p) \frac{dv}{dt} = -(m_c + m_p)g + \dot{m}_p v_e - \frac{1}{2} \rho v |v| A C_D, \quad (1)$$

where z is the vertical coordinate, $v = dz/dt$ is the vertical speed, and

$$\begin{aligned} m_c &= 51.02 \text{ kg} && \text{(rocket casing mass)} \\ g &= 9.8 \text{ m/s}^2 && \text{(gravitational acceleration)} \\ \rho &= 1.23 \text{ kg/m}^3 && \text{(air density)} \\ A &= 0.1 \text{ m}^2 && \text{(maximum cross-sectional area)} \\ v_e &= 360 \text{ m/s} && \text{(exhaust speed)} \\ C_D &= 0.15 && \text{(drag coefficient)} \\ m_{p0} &= 102.04 \text{ kg} && \text{(initial propellant mass)}. \end{aligned}$$

Furthermore, the instantaneous propellant mass at time t is given by

$$m_p(t) = m_{p0} - \int_0^t \dot{m}_p dt, \quad (2)$$

and the time-varying burn rate is

$$\dot{m}_p = \frac{m_{p0}}{4} \cdot \begin{cases} t & 0 \leq t \leq 1 \\ 1 & 1 \leq t \leq 4 \\ 5 - t & 4 \leq t \leq 5 \\ 0 & 5 \leq t. \end{cases} \quad (3)$$

Use a second-order Runge-Kutta (RK2) integrator with $\Delta t = 0.1$ s to plot $z(t)$ and $v(t)$. Use these plots to find the maximum speed, and the height and time at which it is reached; the maximum height, and the time at which it is reached; and the time and velocity when the rocket hits the ground. Check these results with those obtained from MATLAB's ode45 solver.

This problem is fairly straight-forward. An RK2 scheme is relatively easy to implement, but we will need to be careful when accounting for the time-dependence of m_p . Furthermore, though it is

not stated in the problem, when m_p reaches zero, the thrust term involving the exhaust speed v_e in (1) needs to “turn off.” Lastly, it would be un-physical for the rocket to initially fall through the launch pad before thrust overcomes gravity, and we must account for this when coding the solution.

1.2 Problem 2

Consider the stream function for a two-dimensional jet in self-similar form, which can be written as

$$f''(\eta) + f(\eta)f'(\eta) = 0, \quad f(0) = 0, \quad f'(0) = 1, \quad \Rightarrow f''(0) = 0. \quad (4)$$

The velocities in the jet can be obtained via

$$\frac{U}{U_0} = f'(\eta), \quad \frac{V}{U_0} = \eta f'(\eta) - \frac{1}{2}f(\eta). \quad (5)$$

Using numerical integration for $0 \leq \eta \leq 4$, with a step size of $\Delta\eta = 0.05$, plot the quantities U/U_0 , V/U_0 , and f as functions of η . Perform this analysis with both the second- (custom) and fourth-order (ode45) Runge-Kutta methods, and compare their performance. Both solutions may be compared to those obtained from the best-fit expression

$$f'(\eta) = \exp(-0.682\eta^2). \quad (6)$$

1.3 Problem 3

Show that the equation

$$y'' = -\frac{19}{4}y - 10y', \quad y(0) = -9, \quad y'(0) = 0 \quad (7)$$

is moderately stiff. Use Euler stability analysis to estimate the largest step size h_{\max} for which the Runge-Kutta method will be stable. Then confirm this estimate by computing y using the fourth-order Runge-Kutta method with $h = \{\frac{1}{2}h_{\max}, 2h_{\max}\}$. Compare these solutions with the analytical solution.

2 Methodology

2.1 Problem 1

We first re-write the system of equations governing rocket altitude as

$$\begin{aligned} \frac{dz}{dt} &= f(t, v) = v \\ \frac{dv}{dt} &= g(t, v) = \underbrace{-g + T \frac{\dot{m}_p v_e}{(m_c + m_p)}}_{\alpha(t)} + \underbrace{\frac{-\rho A C_D}{2(m_c + m_p)} v |v|}_{\beta(t)}, \end{aligned} \quad (8)$$

where α and β are constants that directly depend only on time, and T toggles the thrust term on and off based on whether the rocket is still burning propellant. In the MATLAB code, the functions that calculate $\alpha(t)$ and $\beta(t)$ call subsequent functions to calculate $m_p(t)$ and $\dot{m}_p(t)$.

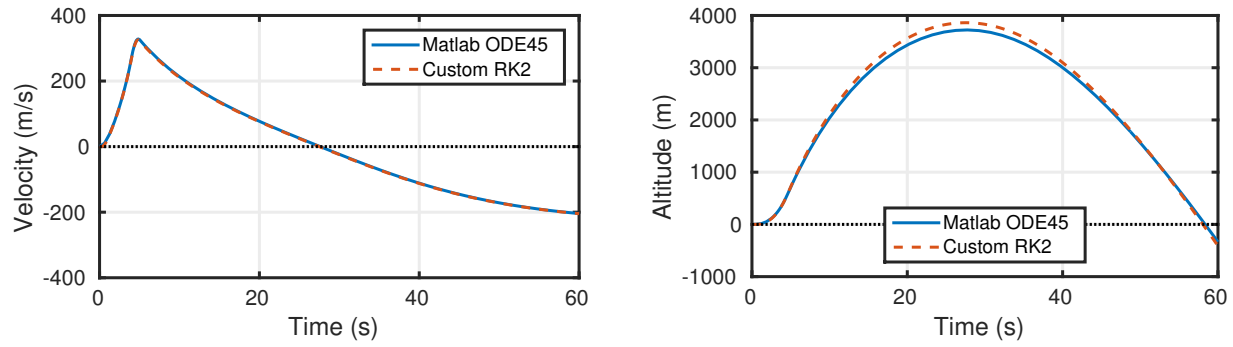


Figure 1: Comparison of our own 2nd-order Runge-Kutta scheme to MATLAB's ode45 solver, for the rocket velocity and altitude.

Integrator	Max Velocity			Max Altitude		Crash	
	Time	Velocity	Altitude	Time	Altitude	Time	Velocity
Custom RK2	4.80	325.74	653.47	27.50	3861.61	58.06	-199.79
MATLAB ode45	4.80	328.71	648.04	27.74	3723.21	58.34	-200.19

Table 1: Comparison of notable flight events between integration methods.

Given the initial values of our variables at $t = 0$, the second-order Runge-Kutta method allows us to integrate the system of equations (8) with respect to time using a time step h . At a given time t_n , the velocity and position at the next time step, v_{n+1} and z_{n+1} , can be approximated by applying the following formulae to each successive time step: calculating

$$\begin{aligned}
 k_1 &= hf(t_n, v_n) &= hv_n \\
 l_1 &= hg(t_n, v_n) &= h[\alpha(t_n) + \beta(t_n)v_n|v_n|] \\
 k_2 &= hf(t_n + h/2, v_n + k_1/2) &= h[v_n + k_1/2] \\
 l_2 &= hg(t_n + h/2, v_n + l_1/2) &= h[\alpha(t_n + h/2) + \beta(t_n + h/2)(v_n + l_1/2)|v_n + l_1/2|]
 \end{aligned} \tag{9}$$

and then updating the next step's values of z and v as

$$\begin{aligned}
 z_{n+1} &= z_n + k_2 \\
 v_{n+1} &= v_n + l_2.
 \end{aligned} \tag{10}$$

For early time steps, say $t < 3$, we ensure that the rocket does not fall through the launch pad by setting v_{n+1} and z_{n+1} to zero if they are negative after the RK2 iteration.

3 Results

3.1 Problem 1

Plots of the velocity and altitude time histories are presented in Figure 1. Statistics on notable flight events are shown in Table 1.

4 Discussion

Problem 1

Using both integration strategies, we physically see the rocket accelerate at an increasing rate as propellant is burned, and then decelerate as the rocket engine shuts down and the forces of gravity and air drag dominate its dynamics.

Agreement of time-histories of rocket velocity and altitude is strong between the 2nd-order Runge-Kutta method we implemented and MATLAB's own ode45 solver. This holds true both visually and quantitatively, based on Figure 1 and Table 1 respectively. Errors between methods are within 3% for time, within 1% for velocity, and within 9% for altitude. This can be attributed to two phenomena. First, MATLAB's integrator uses an adaptive time-step coupled with a higher-order integration scheme. It is thus able to capture the behavior that occurs during the take-off stage—and throughout the flight—more accurately. Second, errors in position are higher than errors in velocity, because errors from the velocity calculations are compounded when estimating the altitude at each time step.

5 References

No external references were used other than the course notes for this assignment.

Appendix: MATLAB Code

The following code listings generate all figures presented in this homework assignment.

Listing 1: Problem_1.m

```
1 function [] = Problem_1()
2
3     %%%%%%
4     % Solves the rocket equation using a second-order Runge-Kutta method.
5     % Ryan Skinner, September 2015
6     %%%
7
8     Set_Default_Plot_Properties();
9
10    % Define constants.
11    c.mc = 51.02; % Rocket casing mass
12    c.g = 9.8; % Gravitational accel
13    c.rho = 1.23; % Air density
14    c.A = 0.1; % Max cross-sectional area
15    c.ve = 360; % Exhaust speed
16    c.CD = 0.15; % Drag coefficient
17    c.mp0 = 102.04; % Initial propellant mass
18    c.v0 = 0; % Initial velocity
19    c.z0 = 0; % Initial altitude
20
21    % Initialize times at which to evaluate solution.
22    t0 = 0;
23    tf = 60;
24    dt = 0.1;
25    t = t0:dt:tf;
26
27    % Initialize velocities and positions.
28    v = zeros(length(t),1);
29    z = zeros(length(t),1);
```

```

30 v(1) = c.v0;
31 z(1) = c.z0;
32
33 % Perform integration using an RK2 method.
34 for n = 1:(length(t)-1)
35     tn = t(n);
36     vn = v(n);
37     zn = z(n);
38     k1 = dt * (alpha(c,tn) + beta(c,tn) * vn * abs(vn));
39     l1 = dt * vn;
40     k2 = dt * (alpha(c,tn + dt/2) + beta(c,tn + dt/2) * (vn + k1/2) * abs(vn + k1/2));
41     l2 = dt * (vn + l1/2);
42     v(n+1) = vn + k2;
43     z(n+1) = zn + l2;
44     % Prevent rocket from falling into the ground before thrust overcomes gravity.
45     if n * dt < 3
46         if v(n+1) < 0
47             v(n+1) = 0;
48         end
49         if z(n+1) < 0
50             z(n+1) = 0;
51         end
52     end
53 end
54
55 % Perform validation using Matlab's ODE45.
56 t_span = [t0, tf];
57 initials = [v(1), z(1)];
58 [t_ml, sol_ml] = ode45(@(t,y) rocket(t,y,c), t_span, initials);
59 v_ml = sol_ml(:,1);
60 z_ml = sol_ml(:,2);
61
62 % Plot velocity.
63 figure();
64 hold on;
65 plot(t_ml,v_ml,'DisplayName','Matlab ODE45');
66 plot(t,v,'--','DisplayName','Custom RK2');
67 legend('show');
68 plot(t_span,[0,0],'k:');
69 xlim(t_span);
70 xlabel('Time (s)');
71 ylabel('Velocity (m/s)');
72
73 % Plot altitude.
74 figure();
75 hold on;
76 plot(t_ml,z_ml,'DisplayName','Matlab ODE45');
77 plot(t,z,'--','DisplayName','Custom RK2');
78 hleg = legend('show');
79 set(hleg,'location','south');
80 plot(t_span,[0,0],'k:');
81 xlim(t_span);
82 xlabel('Time (s)');
83 ylabel('Altitude (m)');
84
85 % Stats on rocket flight (Custom RK2)
86 i = find(v == max(v));
87 fprintf('RK2 : Max velocity is %.2f at time %.2f and height %.2f.\n',max(v),t(i),z(i));
88 i = find(z == max(z));
89 fprintf('RK2 : Max altitude is %.2f at time %.2f.\n',max(z),t(i));
90 ignore = 100;
91 t_crash = interp1(z(ignore:end),t(ignore:end),0);
92 fprintf('RK2 : Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t,v,t_crash));
93
94 % Stats on rocket flight (Matlab's ODE45)
95 i = find(v_ml == max(v_ml));
96 fprintf('ODE45: Max velocity is %.2f at time %.2f and height %.2f.\n',max(v_ml),t_ml(i),z_ml(i));
97 i = find(z_ml == max(z_ml));
98 fprintf('ODE45: Max altitude is %.2f at time %.2f.\n',max(z_ml),t_ml(i));

```

```

99     ignore = 50;
100     t_crash = interp1(z_ml(ignore:end),t_ml(ignore:end),0);
101     fprintf('ODE45: Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t_ml,v_ml,t_crash));
102
103 end
104
105 function [ mp_dot ] = mp_dot ( c, t )
106 % Calculates time derivative of propellant mass.
107 % This is the exact value of mp(t) given in the problem statement.
108     if 0 <= t && t < 1
109         mp_dot = t;
110     elseif 1 <= t && t < 4
111         mp_dot = 1;
112     elseif 4 <= t && t < 5
113         mp_dot = 5-t;
114     else
115         mp_dot = 0;
116     end
117     mp_dot = mp_dot * c.mp0 / 4;
118 end
119
120 function [ mp ] = mp ( c, t )
121 % Calculates instantaneous propellant mass.
122 % This is the exact value of mp(t) given in the problem statement.
123     if t < 0
124         intg = 0;
125     elseif 0 <= t && t < 1
126         intg = 0.5 * t^2;
127     elseif 1 <= t && t < 4
128         intg = t - 0.5;
129     elseif 4 <= t && t < 5
130         intg = 3.5 + (t-4) - 0.5 * (t-4)^2;
131     else
132         intg = 4;
133     end
134     mp = c.mp0 - intg * c.mp0 / 4;
135 end
136
137 function [ a ] = alpha( c, t )
138 % Calculates the time-dependent constant alpha.
139     engines_on = 0 <= t && t < 5;
140     a = -c.g + (engines_on * c.ve) * mp_dot(c,t) / (mp(c,t) + c.mc);
141 end
142
143 function [ b ] = beta( c, t )
144 % Calculates the time-dependent constant beta.
145     b = -0.5 * c.rho * c.A * c.CD / (mp(c,t) + c.mc);
146 end
147
148 function [ dy ] = rocket( t, y, c )
149 % Calculates values for ODE45-based integration of the rocket equation.
150     dy = zeros(2,1);
151     dy(1) = alpha(c,t) + beta(c,t) * y(1) * abs(y(1));
152     dy(2) = y(1);
153     % Prevent rocket from falling into the ground before thrust overcomes gravity.
154     if t < 3
155         if dy(1) < 0
156             dy(1) = 0;
157         end
158         if dy(2) < 0
159             dy(2) = 0;
160         end
161     end
162 end

```