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

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

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

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

and the time-varying burn rate is

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

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."

1.2 Problem 2

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

$$f''(\eta) + f(\eta)f'(\eta) = 0, \qquad f(0) = 0, \qquad f'(0) = 1. \tag{4}$$

The velocities in the jet can be obtained via

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

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/V_0, and f as functions of η. Perform this analysis with both the second- and fourth-order Runge-Kutta methods, and compare their performance. Both solutions may be compared to those obtained from the best-fit expression

$$f(\eta) = \exp(-0.682\eta^2). \tag{6}$$

1.3 Problem 3

Show that the equation

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

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

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

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.

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

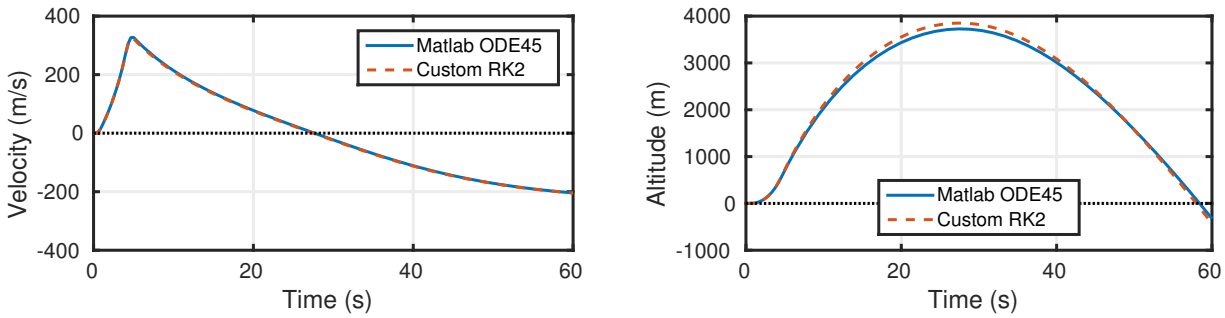

Figure 1: Comparison of our own 2nd-order Runge-Kutta scheme to MATLAB's `ode45` solver, for the rocket velocity and altitude.

Integrator	Max Velocity			Max Altitude		Crash	
	Time	Velocity	Altitude	Time	Altitude	Time	Velocity
Custom RK2	4.80	325.10	649.86	27.50	3850.40	57.98	-199.65
MATLAB `ode45`	4.67	327.01	597.32	27.36	3726.67	58.40	-200.27

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

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

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

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

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

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

```
function [] = Problem_1()

    %%%%%%
    % Solves the rocket equation using a second-order Runge-Kutta method.
    %   Ryan Skinner, September 2015
    %%%

    Set_Default_Plot_Properties();

    % Define constants.
    c.mc    = 51.02;    % Rocket casing mass
    c.g     = 9.8;      % Gravitational accel
    c.rho   = 1.23;     % Air density
    c.A     = 0.1;      % Max cross-sectional area
    c.ve    = 360;      % Exhaust speed
    c.CD    = 0.15;     % Drag coefficient
    c.mp0   = 102.04;   % Initial propellant mass
    c.v0    = 0;        % Initial velocity
    c.z0    = 0;        % Initial altitude

    % Initialize times at which to evaluate solution.
    t0 = 0;
    tf = 60;
    dt = 0.1;
    t = t0:dt:tf;

    % Initialize velocities and positions.
    v = zeros(length(t),1);
    z = zeros(length(t),1);
    v(1) = c.v0;
    z(1) = c.z0;

    % Perform integration using an RK2 method.
    for n = 1:(length(t)-1)
        tn = t(n);
        vn = v(n);
        zn = z(n);
        k1 = dt * (alpha(c,tn)      + beta(c,tn)      * vn   * abs(vn));
```

```
        l1 = dt * vn;
        k2 = dt * (alpha(c,tn + dt/2) + beta(c,tn + dt/2) * (vn + k1/2) * abs(vn + k1/2));
        l2 = dt * (vn + l1/2);
        v(n+1) = vn + k2;
        z(n+1) = zn + l2;
    end

    % Perform validation using Matlab's ODE45.
    t_span = [t0, tf];
    initials = [v(1), z(1)];
    [t_ml, sol_ml] = ode45(@(t,y) rocket(t,y,c), t_span, initials);
    v_ml = sol_ml(:,1);
    z_ml = sol_ml(:,2);

    % Plot velocity.
    figure();
    hold on;
    plot(t_ml,v_ml,'DisplayName','Matlab ODE45');
    plot(t,v,'--','DisplayName','Custom RK2');
    legend('show');
    plot(t_span,[0,0],'k:');
    xlim(t_span);
    xlabel('Time (s)');
    ylabel('Velocity (m/s)');

    % Plot altitude.
    figure();
    hold on;
    plot(t_ml,z_ml,'DisplayName','Matlab ODE45');
    plot(t,z,'--','DisplayName','Custom RK2');
    hleg = legend('show');
    set(hleg,'location','south');
    plot(t_span,[0,0],'k:');
    xlim(t_span);
    xlabel('Time (s)');
    ylabel('Altitude (m)');

    % Stats on rocket flight (Custom RK2)
    i = find(v == max(v));
    fprintf('RK2  : Max velocity is %.2f at time %.2f and height %.2f.\n',max(v),t(i),z(i));
    i = find(z == max(z));
    fprintf('RK2  : Max altitude is %.2f at time %.2f.\n',max(z),t(i));
    ignore = 100;
    t_crash = interp1(z(ignore:end),t(ignore:end),0);
    fprintf('RK2  : Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t,v,t_crash));

    % Stats on rocket flight (Matlab's ODE45)
    i = find(v_ml == max(v_ml));
    fprintf('ODE45: Max velocity is %.2f at time %.2f and height %.2f.\n',max(v_ml),t_ml(i),z_ml(i));
    i = find(z_ml == max(z_ml));
    fprintf('ODE45: Max altitude is %.2f at time %.2f.\n',max(z_ml),t_ml(i));
    ignore = 50;
    t_crash = interp1(z_ml(ignore:end),t_ml(ignore:end),0);
    fprintf('ODE45: Crash occurs at time %.2f with velocity %.2f\n',t_crash,interp1(t_ml,v_ml,t_crash));

end

function [ mp_dot ] = mp_dot ( c, t )
% Calculates time derivative of propellant mass.
% This is the exact value of mp(t) given in the problem statement.
    if 0 <= t && t < 1
        mp_dot = t;
    elseif 1 <= t && t < 4
        mp_dot = 1;
    elseif 4 <= t && t < 5
        mp_dot = 5-t;
    else
        mp_dot = 0;
    end
```

```
    mp_dot = mp_dot * c.mp0 / 4;
end

function [ mp ] = mp ( c, t )
% Calculates instantaneous propellant mass.
% This is the exact value of mp(t) given in the problem statement.
    if t < 0
        intg = 0;
    elseif 0 <= t && t < 1
        intg = 0.5 * t^2;
    elseif 1 <= t && t < 4
        intg = t - 0.5;
    elseif 4 <= t && t < 5
        intg = 3.5 + (t-4) - 0.5 * (t-4)^2;
    else
        intg = 4;
    end
    mp = c.mp0 - intg * c.mp0 / 4;
end

function [ a ] = alpha( c, t )
% Calculates the time-dependent constant alpha.
    engines_on = 0 <= t && t < 5;
    a = -c.g + (engines_on * c.ve) * mp_dot(c,t) / (mp(c,t) + c.mc);
end

function [ b ] = beta( c, t )
% Calculates the time-dependent constant beta.
    b = -0.5 * c.rho * c.A * c.CD / (mp(c,t) + c.mc);
end

function [ dy ] = rocket( t, y, c )
% Calculates values for ODE45-based integration of the rocket equation.
    dy = zeros(2,1);
    dy(1) = alpha(c,t) + beta(c,t) * y(1) * abs(y(1));
    dy(2) = y(1);
end
```