MAE 270C Project Report: Rocket Optimization

Nathan Alson

University of California, Los Angeles

Mechanical and Aerospace Engineering Department

Professor Moshe Idan

MAE 270C

Problem Statement

This project focuses on optimizing the trajectory of a vertically launched rocket, as described in Example 4.3.1 from the textbook:

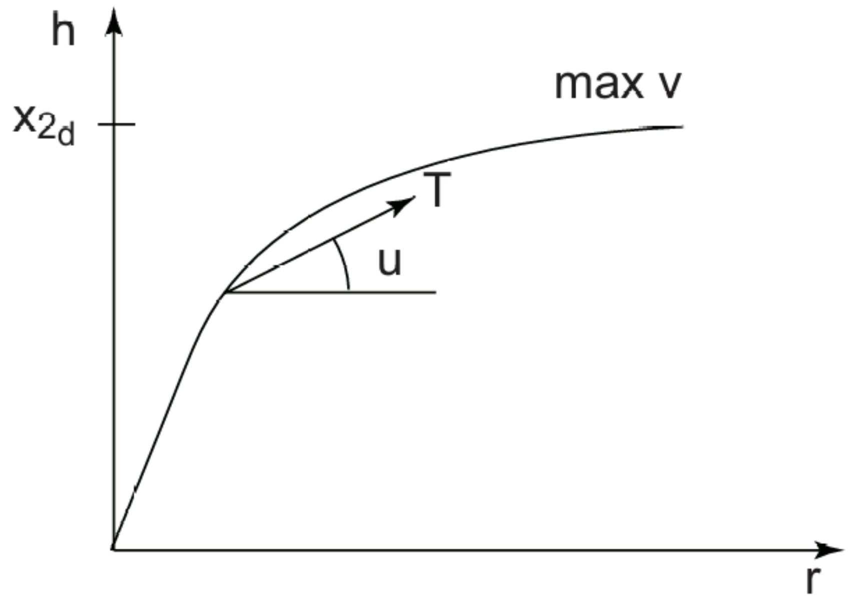

Figure 1: Rocket Launch Example from Textbook

The goal is to maximize the final horizontal velocity of the rocket, subject to both dynamic and terminal constraints. The state dynamics of the system are given as:

$$\dot{x}_1(t) = \dot{r} = x_3(t),$$
$$\dot{x}_2(t) = \dot{h} = x_4(t),$$
$$\dot{x}_3(t) = \dot{v} = T\cos(u(t)),$$
$$\dot{x}_4(t) = \dot{w} = T\sin((t)) \; - \; g,$$

where $x(t) = [r(t), h(t), v(t), w(t)]^T$ represent horizontal position, vertical position, horizontal velocity, and vertical velocity, respectively, $u(t)$ is the flight path angle, T is thrust given to be $64\ ft/s^2$, and g is gravity given as $32\ ft/s^2$. The desired state values from the problem are $h(t_f) = h_f = 320,000\ ft$ and $w(t_f) = w_f = 0\ ft/s$. This then means that the system is subject to the terminal constraints:

$$\psi(x(t_f)) = [x_2(t_f) - h_f,\ x_4 - w_f]^T = 0$$

and the cost function to be minimized is $J =- \ v(t_f)$. This shows that $\phi =- \ x_3(t_f)$ and that $L(x, u, t) = 0$. The initial conditions of the system were all set to 0 and the time horizon was set from 0 to 900 seconds. To solve this problem, I implemented two methods, the built-in numerical function solver approach (Method A) and a steepest descent approach (Method B).

Theory and Implementation

Method A: Built-in numerical function solver

In part A, we apply the indirect method using the weak Pontyragin's Principle. According to the textbook, the optimal control should minimize the Hamiltonian:

$$H(x, u, \lambda, t) = \lambda^T \dot{x} = \lambda_1 x_3 + \lambda_2 x_4 + \lambda_3 T cos(u) + \lambda_4 T sin(u) - \lambda_4 g.$$

The associated costate dynamics are derived from:

$$\dot{\lambda} = - \frac{\partial H}{\partial x} ,$$

$$\lambda(t_f) = \phi_x(x^o(t_f)) + \nu^T \psi_x(x^o(t_f))$$

which yield:

$$\dot{\lambda}_1(t) = 0, \qquad \lambda_1(t_f) = 0,$$

$$\dot{\lambda}_2(t) = 0, \qquad \lambda_2(t_f) = \nu_2,$$

$$\dot{\lambda}_3(t) = - \lambda_1(t), \quad \lambda_3(t_f) = - 1,$$

$$\dot{\lambda}_4(t) = - \lambda_2(t). \quad \lambda_4(t_f) = \nu_4.$$

We can then come to obtain the costates:

$$\lambda_1(t) = 0,$$

$$\lambda_2(t) = \nu_2,$$

$$\lambda_3(t) = - 1,$$

$$\lambda_4(t) = \nu_4 + (t_f - t)\nu_2.$$

Following the principle's condition, the optimal control $u^o(t)$ is found from

$$\frac{\partial H}{\partial u} = 0 \Rightarrow u^o(t) = arctan(- \nu_4 - (t_f - t)\nu_2)$$

The approach in Part A formulates a root-finding problem where ν is adjusted to drive the terminal residuals, $\psi(x(t_f))$, to zero. I used Matlab's fsolve function to solve for ν_2 and ν_4 with an initial guess of $\nu_2 = 0$ and $\nu_4 = - 2$. The resulting optimal trajectory is simulated using ode45, and the Hamiltonian is computed along the trajectory to validate the solution.

Method B: Steepest-Descent

In part B, we implement an iterative gradient based method. Instead of solving for multipliers directly, this approach refines the control trajectory $u(t)$ over multiple iterations.

First, a nominal control $u_N(t)$ is defined and is initialized using the Part A analytical form as well as the same initial guess of $\nu_2 = 0$ and $\nu_4 =- 2$ and with a time step of 0.5 seconds. It is then placed in a loop where, at each iteration, the process consists of:

1. Forward simulation of the system using $u_N(t)$, yielding the final state $x(t_f)$ using Matlab's ode45.
2. The terminal constraints, $\psi(x(t_f))$, are checked to see if they have converged within a tolerance of 1e-6. Additionally, the change in cost function, $\delta\phi$, is also checked against the same tolerance. If both conditions are satisfied, the loop terminates. Otherwise, the method proceeds to the next iteration.
3. The desired change in terminal constraint is then defined as:
$$\delta\psi =- \alpha\psi$$
where α was set to 0.15 for every iteration. This controls the aggressiveness of the update and ensures constraints are approached gradually over iterations. The desired change in cost was chosen through ϵ which remained at a value of 0.005 at each iteration.
4. Compute the lagrange multipliers ν:
$$\nu =- \left[\int_{t_0}^{t_f} \lambda^{\psi^T} f_u f_u^T \lambda^{\psi} dt\right]^{-1} \bullet \left[\frac{\delta\psi}{\epsilon} + \int_{t_0}^{t_f} \lambda^{\psi^T} f_u f_u^T \lambda^{\phi} dt\right]$$
where λ^{ψ} and λ^{ϕ} are backwards integrated using:
$$\dot{\lambda}^{\psi}(t) =- f_x^{\ T}\lambda^{\psi}(t), \quad \lambda^{\psi}(t_f) = \psi_x^T,$$
$$\dot{\lambda}^{\phi}(t) =- f_x^{\ T}\lambda^{\phi}(t), \quad \lambda^{\phi}(t_f) = \phi_x^T,$$
and matrices f_x, f_u, ϕ_x, and ψ_x are defined as:
$$f_x = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$
$$f_u = \begin{bmatrix} 0 \\ 0 \\ -T\sin(u_N) \\ T\cos(u_N) \end{bmatrix},$$
$$\phi_x = [0\ 0\ -1\ 0],$$
$$\psi_x = [0\ 1\ 0\ 0;\ 0\ 0\ 0\ 1].$$
Since these matrices are constant, λ^{ψ} and λ^{ϕ} are only calculated once and are reused every iteration.
5. Compute updated control using:

$$\delta u(t) = -\epsilon(\lambda^{\psi^T}(t) + \nu^T\lambda^{\psi^T}(t))f_u,$$
$$u_{N+1}(t) = u_N(t) + \delta u(t).$$

This is then given back into the loop and is repeated until the convergence in step 2 is satisfied.

With a timestep of 0.5 second, the loop was able to converge in 185 iterations.

Results

Convergence History of the Two Methods (Question 1):

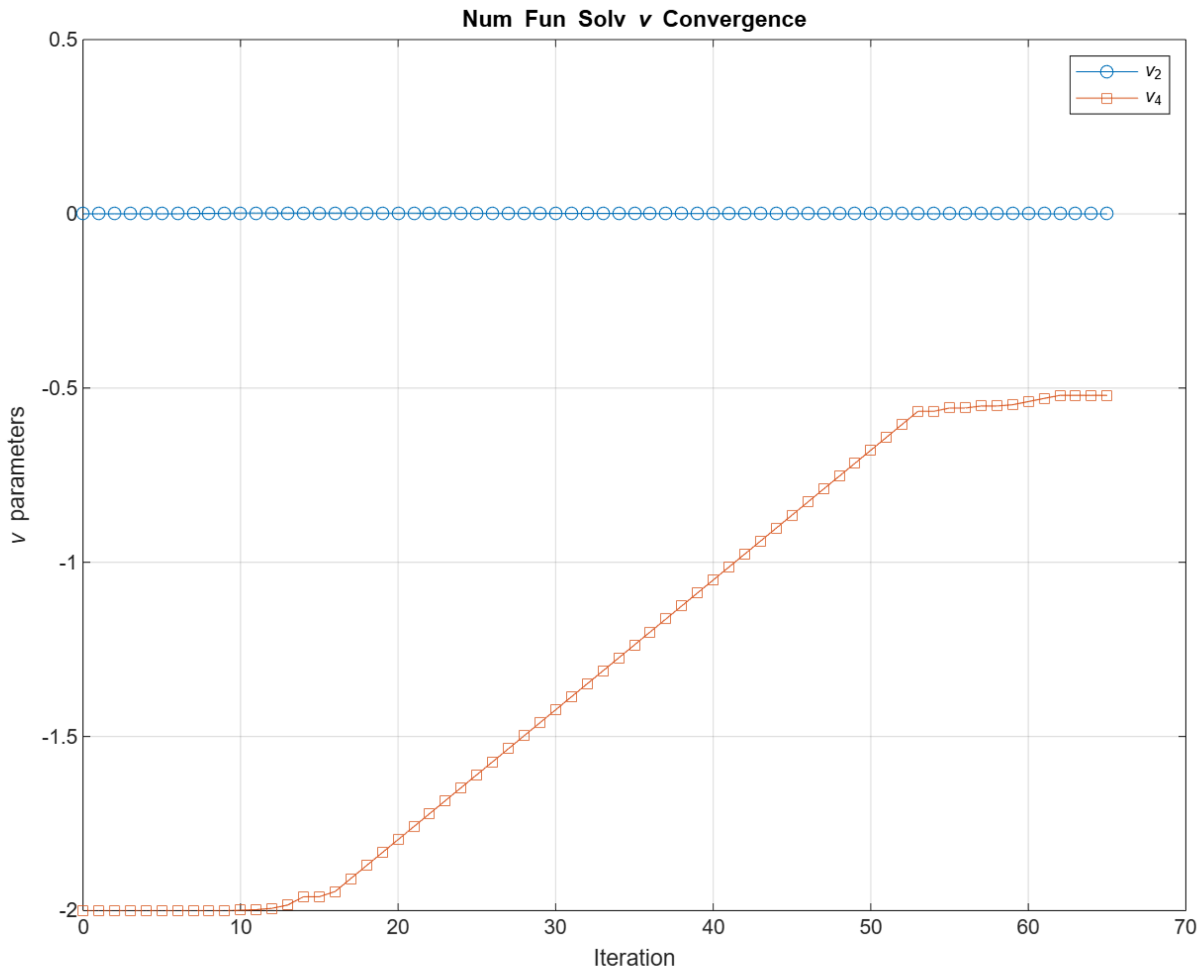

Figure 2: ν Values Obtained from Num Fun Solv

As shown above in **Figure 2**, ν_2 stays at a constant 0 for all iterations but ν_4 stays at the initial guess for about 12 iterations then gradually increases steadily until about the 53rd iteration and then settles down at -0.521 around the 65th iteration.

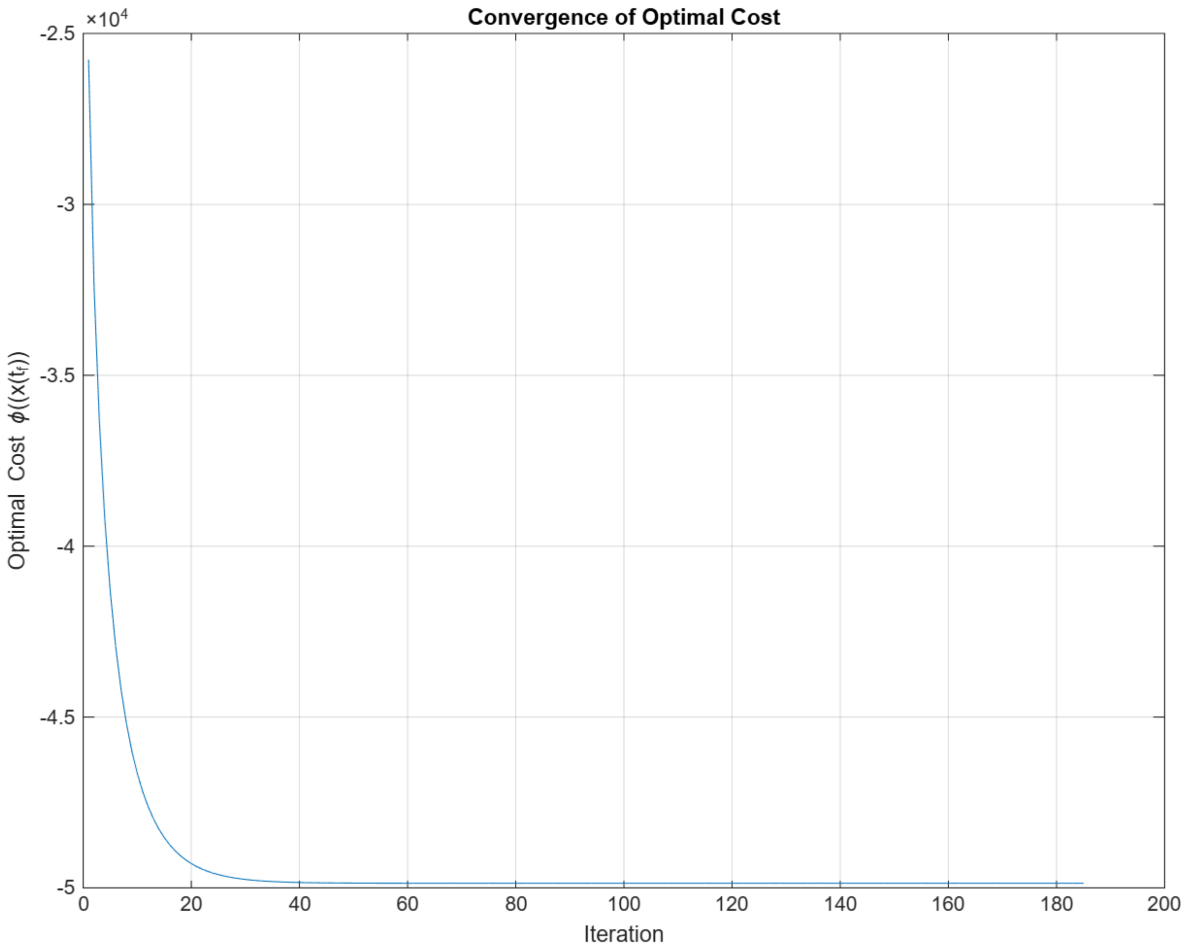

Figure 3: Cost History for Steepest Descent

The cost plot shown above in **Figure 3** demonstrates a rapid initial decrease in the cost during the first 20-30 iterations, after which the cost gradually levels out and converges to a stable value near -5e4. The negative values of ϕ is expected, since ϕ in this problem is defined as the negative of the horizontal velocity at the final time. The smooth and monotonic decrease in ϕ indicates that the optimization in method B is performing correctly and steadily improving the control input to meet the terminal constraints. The fact that ϕ converges to a steady value further confirms that the algorithm has reached an optimal solution within the given tolerance.

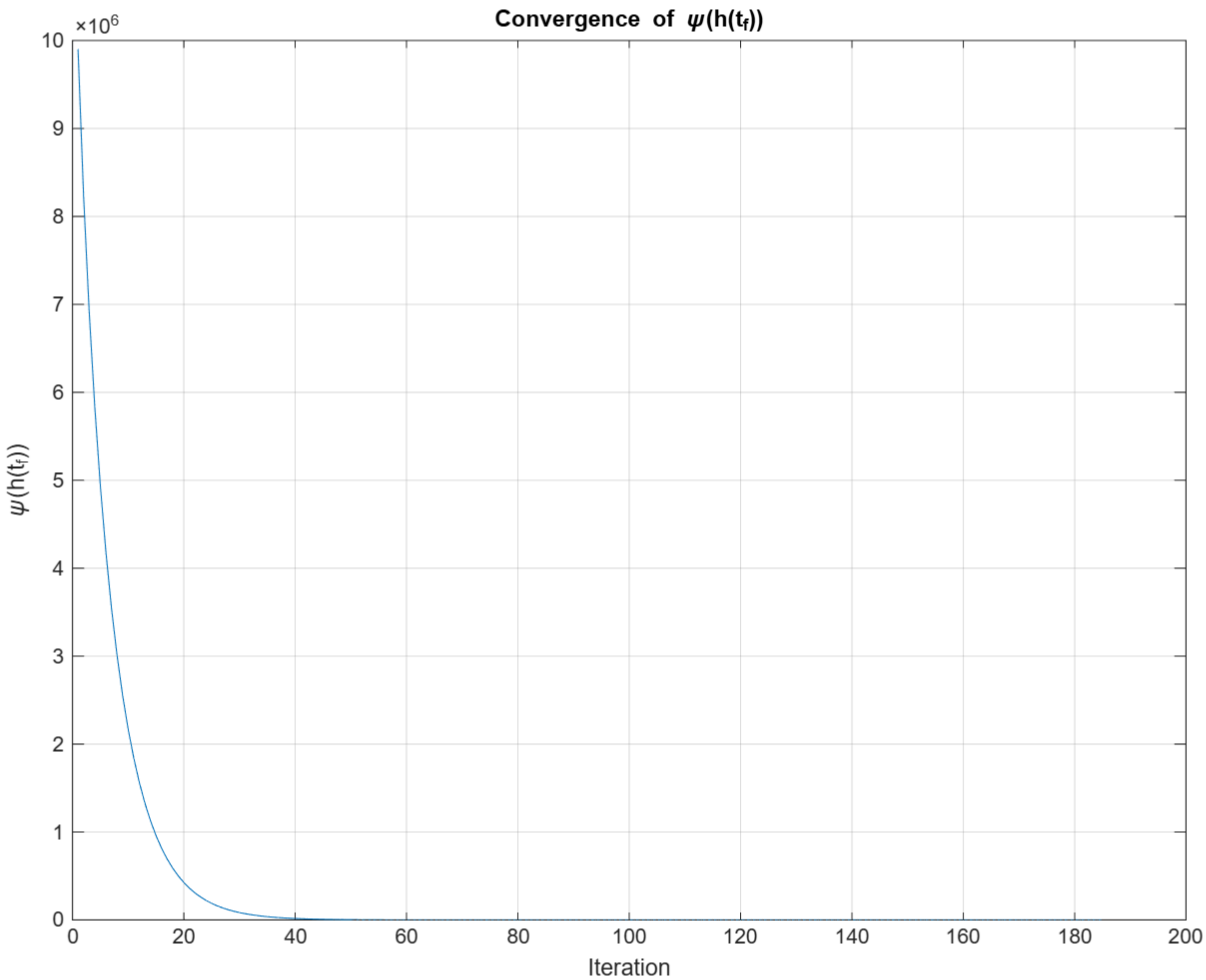

Figure 4: Terminal constraint on height vs iteration

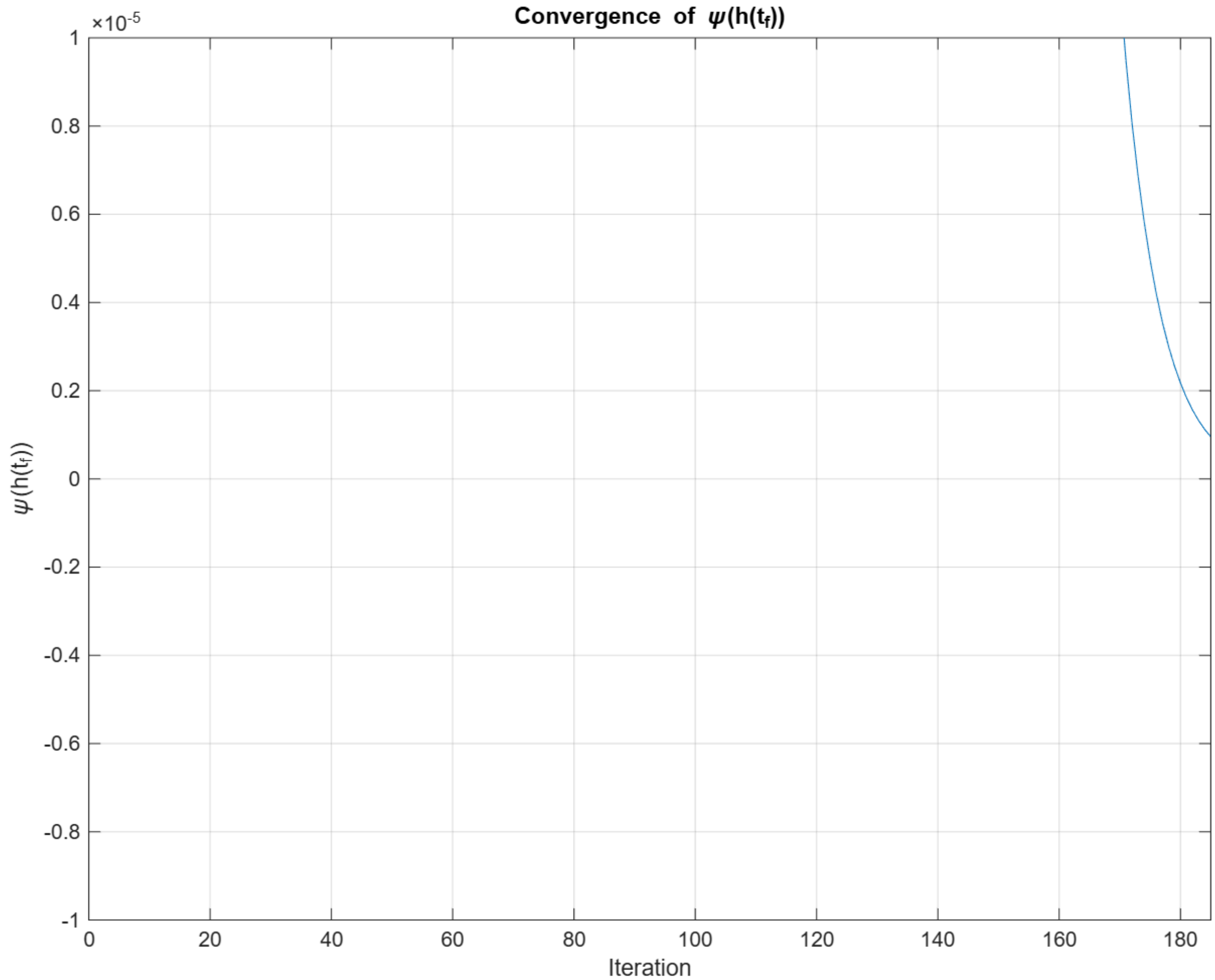

Figure 5: Zoomed-in Terminal constraint on height vs iteration

Figures 4 and 5 provide a visual representation of the compliance of the terminal constraint on height. Starting from a large initial value, the constraint decreases rapidly during the first 20-30 iterations, then continues to decrease gradually. This behaviour demonstrates that the optimizer is progressively fine tuning the control input to reduce the terminal altitude error. The tolerance was set to 1e-6 and the zoomed in plot in **Figure 5** shows that this was achieved at the 185th iteration.

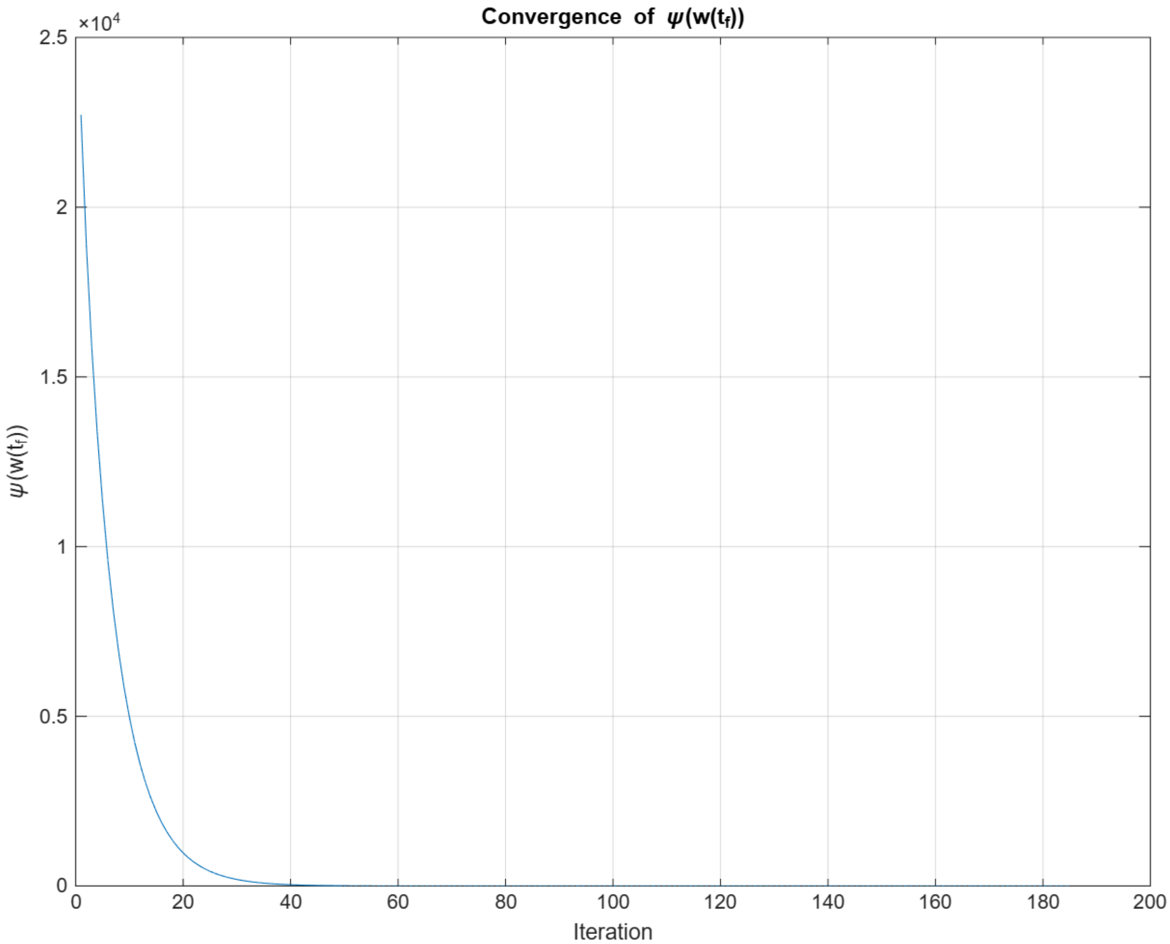

Figure 6: Terminal constraint on vertical velocity vs iteration

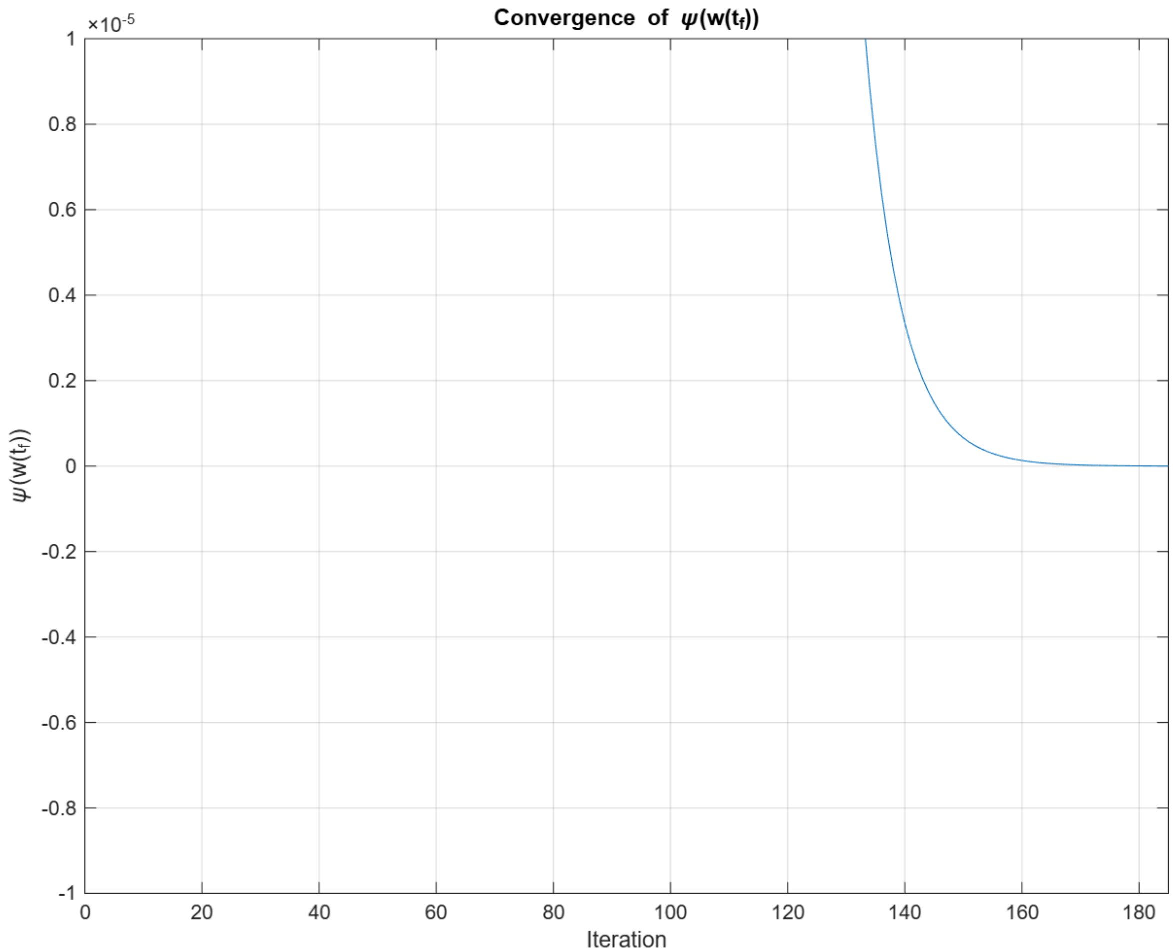

Figure 7: Zoomed-in Terminal constraint on vertical velocity vs iteration

As shown in **Figure 6** and **Figure 7**, the terminal constraint on vertical velocity was also met. Similarly to the graphs for $\psi(h(tf_f))$, the plot for $\psi(w(t_f))$ also demonstrates the progressive refining of the control input over iterations. However, something interesting to note was that this constraint seemed to have been met at around iteration 165 which means that the terminal constraint on height took longer to converge.

At convergence, method A obtained ν_2= -0.000126808895869158, ν_4= -0.520990845854167, $\psi(h(t_f))$ = -2.3283064365387e-10, and $\psi(w(t_f))$ = -8.38440428196918e-13. Alternatively, method B obtained ν_2= -0.000126808895859826, ν_4= -0.520990846124491 $\psi(h(t_f))$= 9.53266862779856e-07, and $\psi(w(t_f))$= 2.20862261812727e-09. This demonstrates that both methods performed quite similarly.

Plots of Optimal Control and States for both Methods (Question 2):

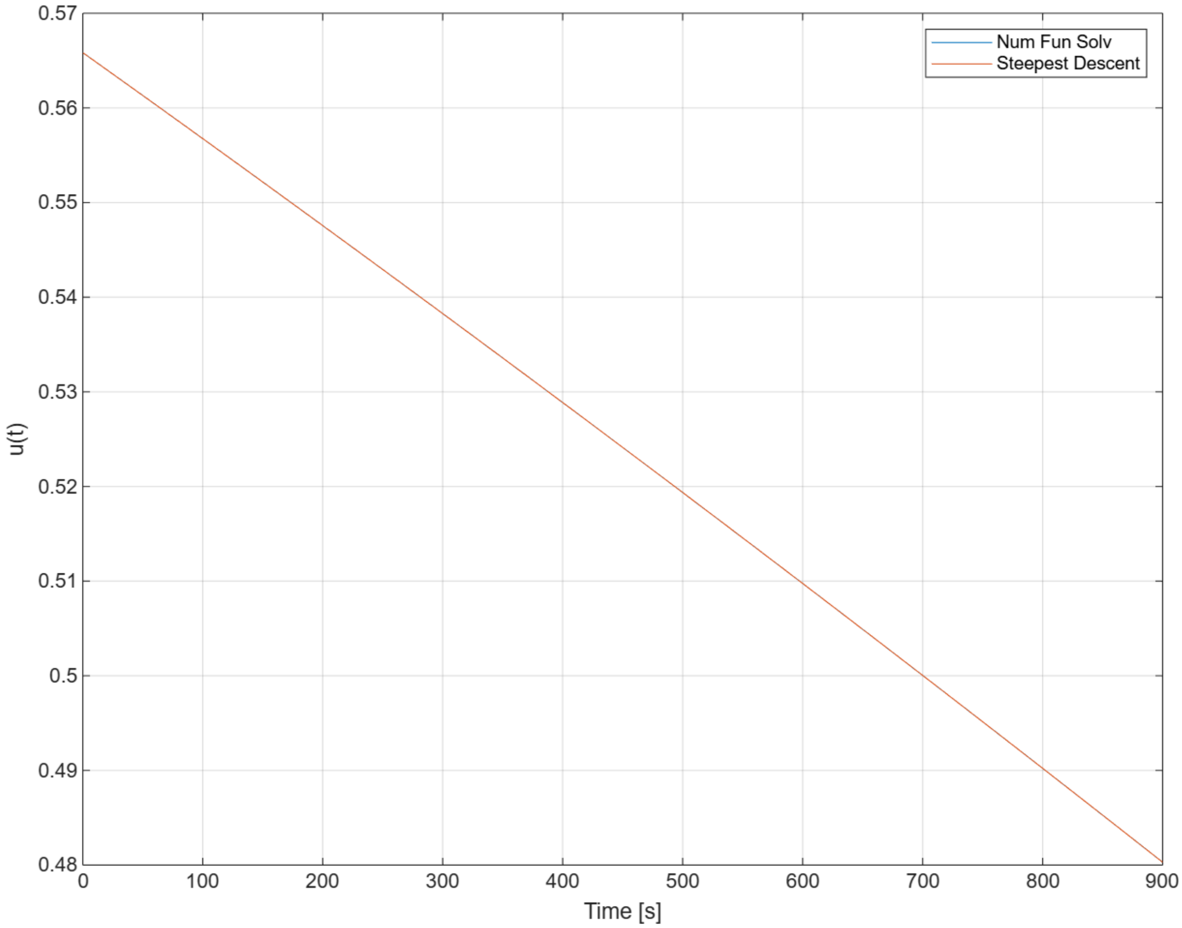

Figure 8: Optimal Cost Between method A and method B

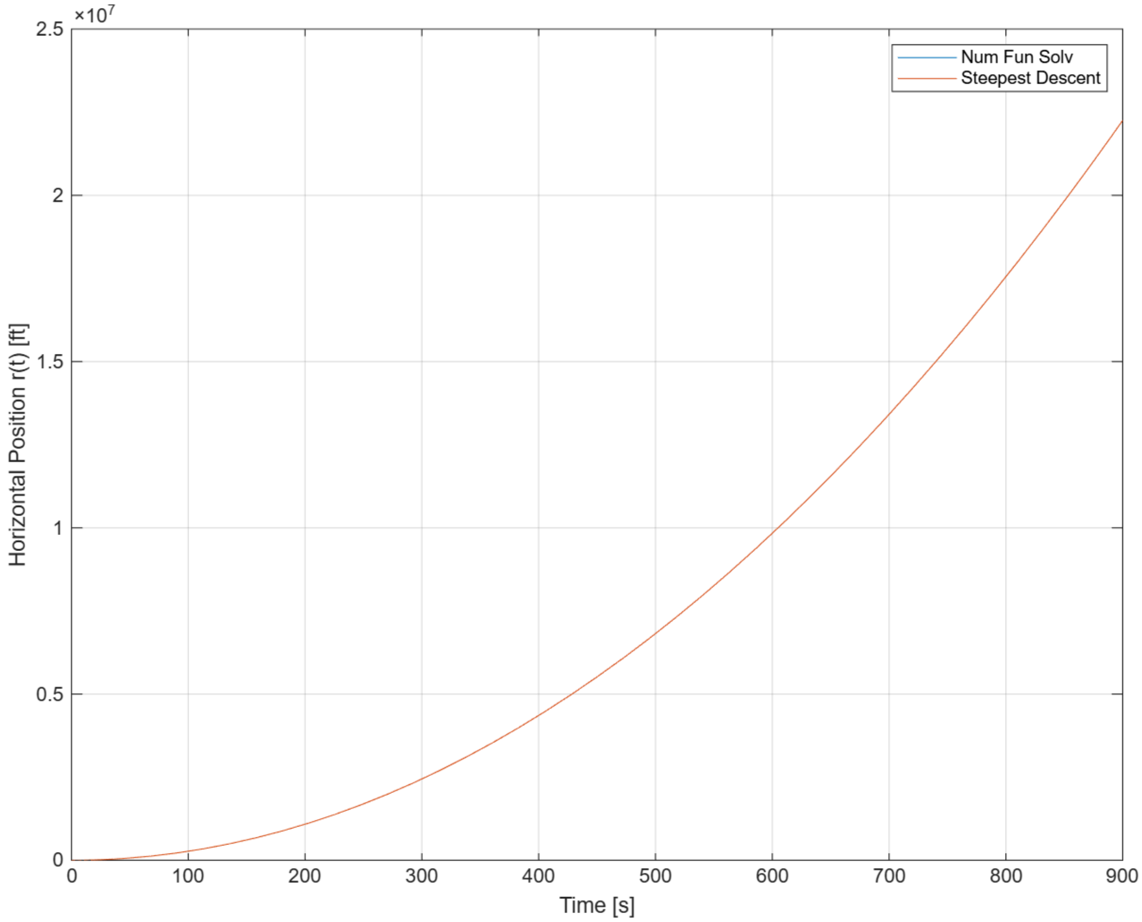

Figure 9: Optimal Horizontal Position vs Time for method A and method B

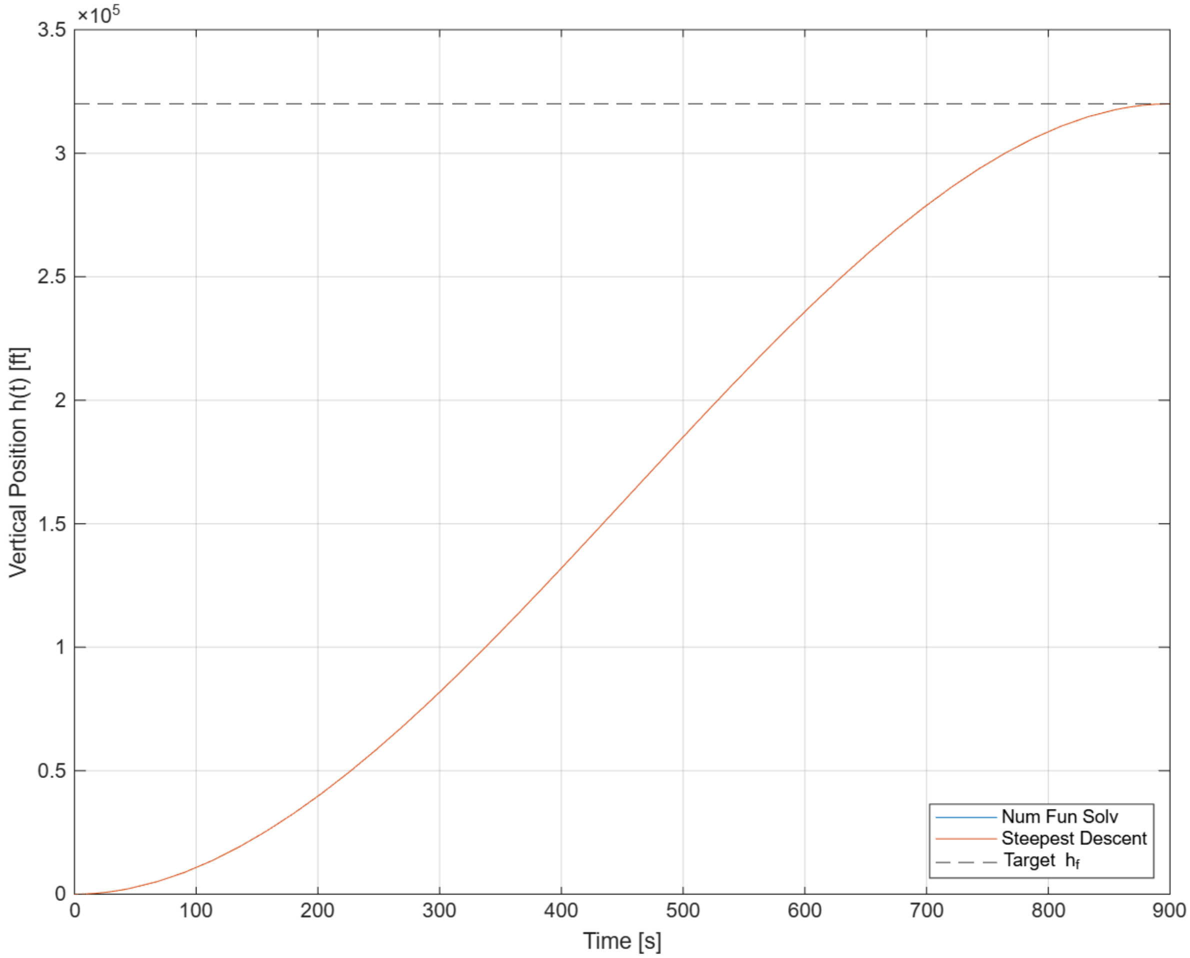

Figure 10: Optimal Vertical Position vs Time for method A and method B

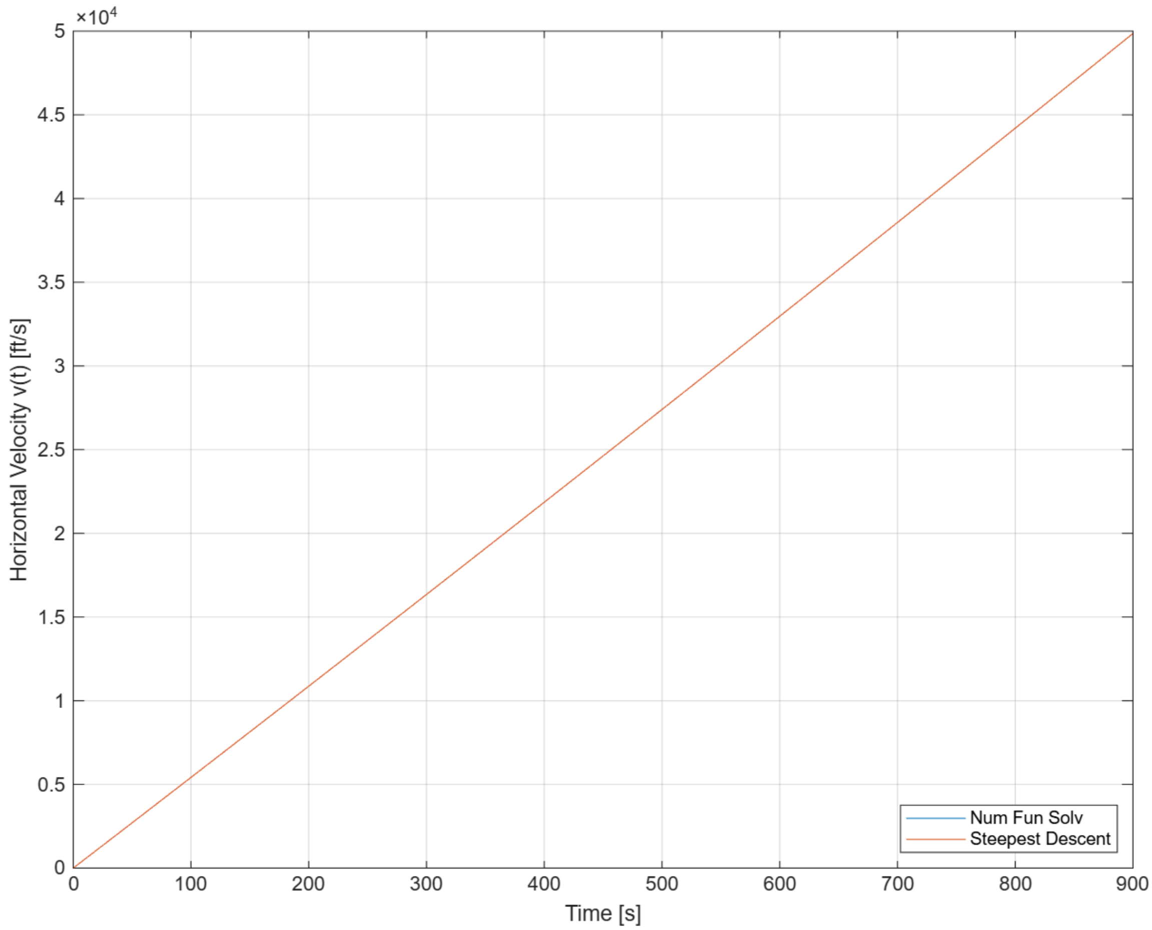

Figure 11: Optimal Horizontal Velocity vs Time for method A and method B

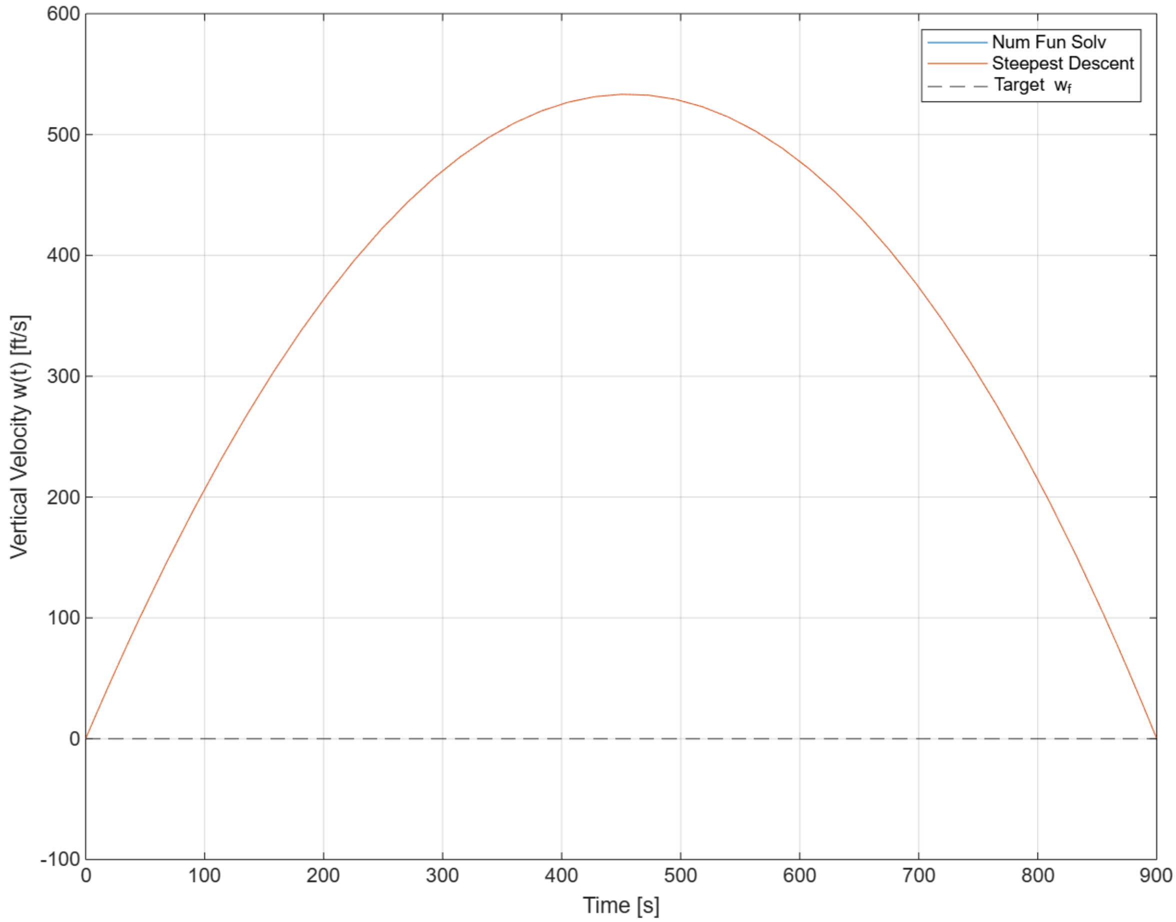

Figure 12: Optimal Vertical Velocity vs Time for method A and method B

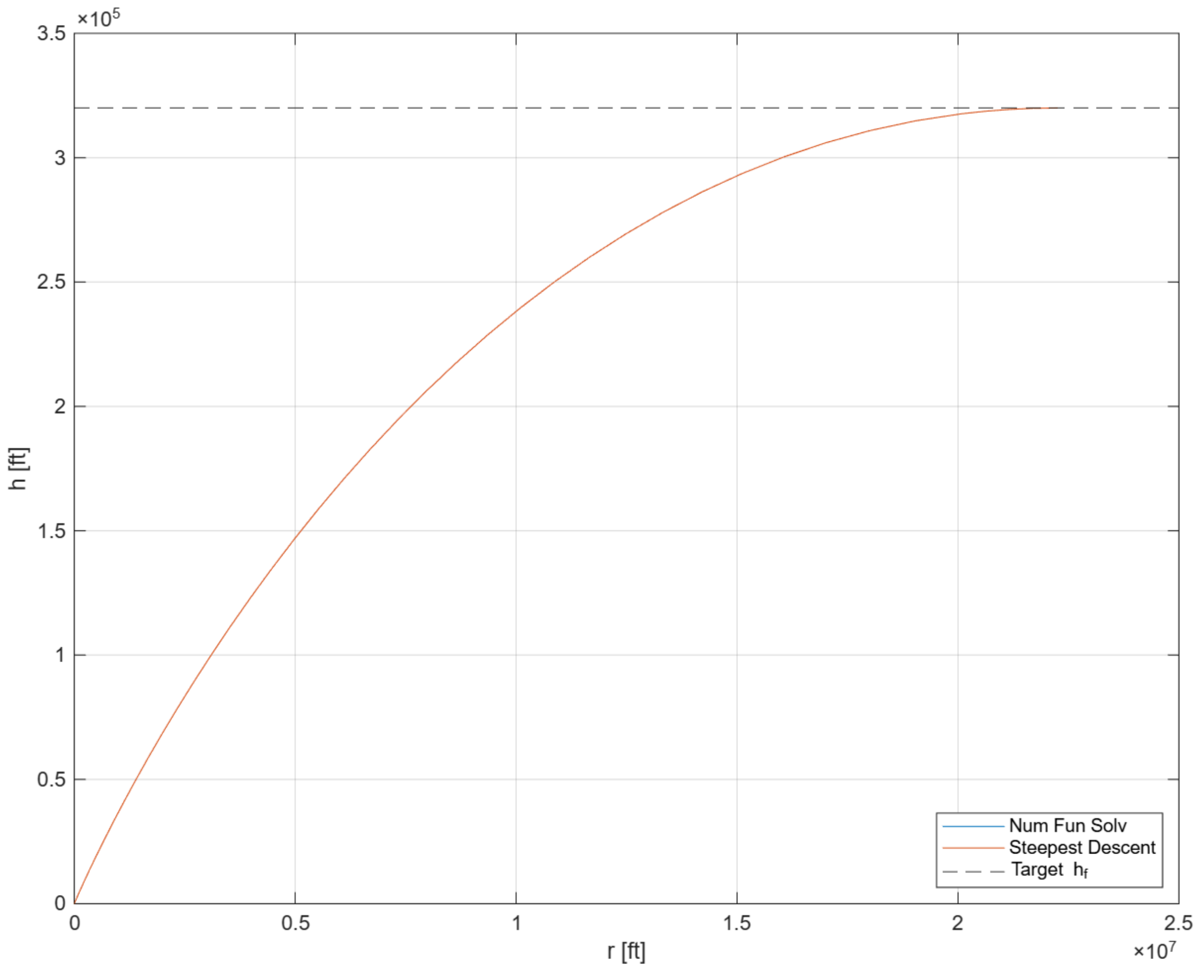

Figure 13: Optimal Vertical Position vs Horizontal Position for method A and method B

Figures 8 - 13 demonstrate that both methods basically achieved the same results. It is also clear to see that in **Figures 10, 12, and 13** that the terminal constraints were all met. **Figure 8** also shows that the optimal cost is decreasing over time.

Hamiltonian from method A (Question 3):

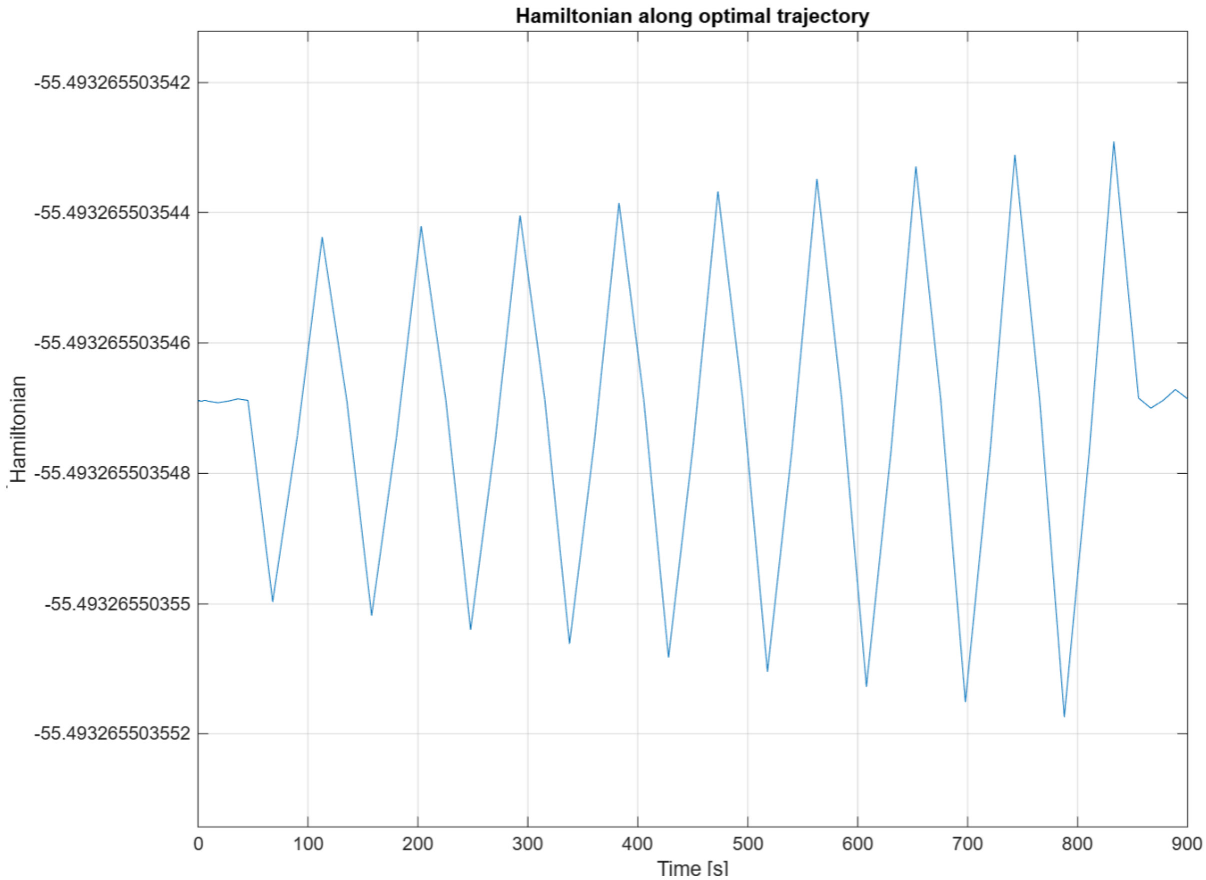

Figure 14: Hamiltonian Matrix from Numerical Function Solver

Figure 14 shows the Hamiltonian that was obtained from method A as a function of time along the optimal solution. Theoretically, the Hamiltonian should remain constant over time and this behavior is largely captured within this figure with the Hamiltonian staying within a very narrow range. The slight oscillations are likely the result of numerical integration effects.

Conclusion

Both numerical methods implemented in this project successfully computed optimal trajectories for the rocket problem. Method A (Numerical Function Solver) produced a solution that matches theoretical expectations, with the Hamiltonian remaining essentially constant along the optimal trajectory. Method B (Steepest Descent) also converged to an optimal solution, with terminal constraints satisfied to a very small tolerance and cost function stabilizing over iterations. The close agreement between the two methods demonstrates the validity of both approaches. Minor numerical variations were observed but are typical in such numerical optimizer problems

Appendix

Matlab Code:

```
% MAE 270C: Rocket Optimization
clear; clc; close all;
% Problem Parameters
g = 32;
T = 2 * g;
tf = 900;
x_init = [0; 0; 0; 0];
target_h = 320000;
target_w = 0;
% Function used for dynamics of model
function dx = rocket_dynamics(x, u, g, T)
   dx = [ x(3);
          x(4);
          T * cos(u);
          T * sin(u) - g ];
end
% Function used to calculate residual from opt function in method A
function err = compute_residual(nu, tf, x_init, g, T, target_h, target_w, u_fn)
   [~, x_sim] = ode45(@(t,x) rocket_dynamics(x, u_fn(t,nu), g, T), [0 tf],
x_init);
   x_tf = x_sim(end,:);
   err = [ x_tf(2) - target_h;
           x_tf(4) - target_w ];
end
% Function to log nu from opt function in method A
function stop = log_nu_progress(x,~,state)
   persistent hist_nu
   if strcmp(state,'init')
       hist_nu = x(:)';
   elseif strcmp(state,'iter')
       hist_nu(end+1,:) = x(:)';
   elseif strcmp(state,'done')
       assignin('base','history_nu', hist_nu);
   end
   stop = false;
```

```
end
% Method A: Optimal Control via Numerical Function Solver
opts = optimoptions('fsolve','Display','iter','OutputFcn', @log_nu_progress);
u_fn = @(t,nu) atan( -nu(2) - (tf - t) * nu(1) );
nu_guess = [0; -2]; % Initial Guess
nu_opt = fsolve(@(nu) compute_residual(nu, tf, x_init, g, T, target_h,
target_w, u_fn), nu_guess, opts);
% Simulate optimal trajectory
u_o = @(t) atan( -nu_opt(2) - (tf - t) * nu_opt(1) );
[t_o, x_o] = ode45(@(t,x) rocket_dynamics(x, u_o(t), g, T), [0 tf], x_init);
% Hamiltonian computation
H_vals = zeros(length(t_o),1);
for i = 1:length(t_o)
   lam1 = 0;
   lam2 = nu_opt(1);
   lam3 = -1;
   lam4 = nu_opt(2) + (tf - t_o(i)) * nu_opt(1);

   u_now = u_o(t_o(i));
   dx3 = T * cos(u_now);
   dx4 = T * sin(u_now) - g;

   H_vals(i) = lam1 * x_o(i,3) + lam2 * x_o(i,4) + lam3 * dx3 + lam4 * dx4;
end
% Plot Hamiltonian Evolution
figure;
plot(t_o, H_vals);
xlabel('Time [s]'); ylabel('Hamiltonian');
title('Evolution of Hamiltonian along Optimal Trajectory');
grid on;
% Load nu history
history_nu = evalin('base','history_nu');
% Plot nu history
figure;
plot(0:length(history_nu)-1, history_nu(:,1), 'o-');
hold on;
plot(0:length(history_nu)-1, history_nu(:,2), 's-');
xlabel('Iteration'); ylabel('\nu parameters');
legend('\nu_2','\nu_4');
title('Num Fun Solv \nu Convergence');
grid on;
% Method B: Steepest Descent
A_mat = [0 0 1 0;
        0 0 0 1;
        0 0 0 0;
        0 0 0 0];
psi_x_tf = [0 1 0 0; 0 0 0 1];
phi_x_tf = [0 0 -1 0];
dt_B = 0.5;
```

```
time_grid = 0:dt_B:tf;
num_grid = length(time_grid);
u_guess = atan( -nu_guess(2) - (tf - time_grid) * nu_guess(1) );
lambda_psi = zeros(4,2,num_grid);
lambda_psi(:,:,num_grid) = psi_x_tf';
for j = num_grid-1:-1:1
   lambda_psi(:,:,j) = lambda_psi(:,:,j+1) - dt_B * ( -A_mat' *
lambda_psi(:,:,j+1) );
end
lambda_phi = zeros(4,num_grid);
lambda_phi(:,num_grid) = phi_x_tf';
for j = num_grid-1:-1:1
   lambda_phi(:,j) = lambda_phi(:,j+1) - dt_B * ( -A_mat' * lambda_phi(:,j+1)
);
end
max_iter = 300;
tol = 1e-6;
eps_step = 5e-3;
psi_log = zeros(max_iter, 2);
phi_log = zeros(max_iter, 1);
u_opt_traj = u_guess;
for iter = 1:max_iter
   [~, x_traj] = ode45(@(t,x) rocket_dynamics(x, interp1(time_grid, u_opt_traj,
t), g, T), [0 tf], x_init);
   x_traj = x_traj';
   x_final = x_traj(:, end);

   psi_curr = [x_final(2) - target_h;
               x_final(4) - target_w];
   phi_curr = -x_final(3);

   psi_log(iter, :) = psi_curr';
   phi_log(iter) = phi_curr;

   % Convergence Check
   if all(abs(psi_curr) < tol) && all(abs(phi_log(iter)-phi_log(iter-1)) < tol)
       fprintf('Converged at Iteration %d\n', iter);
       break;
   end

   delta_psi = -0.15 * psi_curr;

   f_u_grid = zeros(4, num_grid);
   f_u_grid(3,:) = -T * sin(u_opt_traj);
   f_u_grid(4,:) = T * cos(u_opt_traj);

   integ1 = zeros(2,2);
   integ3 = zeros(2,1);
```

```
    for j = 1:num_grid
        L_psi = lambda_psi(:,:,j);
        L_phi = lambda_phi(:,j);
        fu_j  = f_u_grid(:,j);

        integ1 = integ1 + (L_psi' * fu_j) * (fu_j' * L_psi) * dt_B;
        integ3 = integ3 + (L_psi' * fu_j) * (fu_j' * L_phi) * dt_B;
    end

    nu_update = -integ1 \ (delta_psi / eps_step + integ3);

    du = zeros(1,num_grid);
    for j = 1:num_grid
        L_psi = lambda_psi(:,:,j);
        L_phi = lambda_phi(:,j);
        fu_j  = f_u_grid(:,j);

        du(j) = -eps_step * (L_phi' + nu_update' * L_psi') * fu_j;
    end

    u_opt_traj = u_opt_traj + du;
end
% Plot: Steepest Descent Results
k_axis = 1:iter;
% Plot Cost History
figure;
plot(k_axis, phi_log(1:iter));
xlabel('Iteration'); ylabel('Optimal Cost \phi((x(t_f))');
title('Convergence of Optimal Cost');
grid on;
% Plot Psi(h(tf)) over Iterations
figure;
plot(k_axis, psi_log(1:iter,1));
xlabel('Iteration'); ylabel('\psi(h(t_f))');
title('Convergence of \psi(h(t_f))');
%ylim([-1e-5, 1e-5]);
%xlim([0,iter(end)]);
grid on;
% Plot Psi(w(tf)) over Iterations
figure;
plot(k_axis, psi_log(1:iter,2));
xlabel('Iteration'); ylabel('\psi(w(t_f))');
title('Convergence of \psi(w(t_f))');
%ylim([-1e-5, 1e-5]);
%xlim([0,iter(end)]);
grid on;
% Plot Input Trajectory Comparison
figure;
plot(t_o, u_o(t_o)); hold on;
```

```matlab
plot(time_grid, u_opt_traj);
legend('Num Fun Solv','Steepest Descent Optimization');
xlabel('Time [s]'); ylabel('Control u(t)');
grid on;
% State and Input Comparisons
[tB_traj, xB_traj] = ode45(@(t,x) rocket_dynamics(x, interp1(time_grid,
u_opt_traj, t), g, T), [0 tf], x_init);
% r(t)
figure;
plot(t_o, x_o(:,1)); hold on;
plot(tB_traj, xB_traj(:,1));
legend('Num Fun Solv','Steepest Descent');
xlabel('Time [s]'); ylabel('Horizontal Position r(t) [ft]');
grid on;
% h(t)
figure;
plot(t_o, x_o(:,2)); hold on;
plot(tB_traj, xB_traj(:,2));
yline(target_h, 'k--');
legend('Num Fun Solv','Steepest Descent','Target h_f','Location','southeast');
xlabel('Time [s]'); ylabel('Vertical Position h(t) [ft]');
grid on;
% v(t)
figure;
plot(t_o, x_o(:,3)); hold on;
plot(tB_traj, xB_traj(:,3));
legend('Num Fun Solv','Steepest Descent','Location','southeast');
xlabel('Time [s]'); ylabel('Horizontal Velocity v(t) [ft/s]');
grid on;
% w(t)
figure;
plot(t_o, x_o(:,4)); hold on;
plot(tB_traj, xB_traj(:,4));
yline(target_w, 'k--');
legend('Num Fun Solv','Steepest Descent','Target w_f');
xlabel('Time [s]'); ylabel('Vertical Velocity w(t) [ft/s]');
grid on;
% h(t) vs r(t)
figure;
plot(x_o(:,1), x_o(:,2));
hold on;
plot(xB_traj(:,1), xB_traj(:,2));
yline(target_h, 'k--');
legend('Num Fun Solv','Steepest Descent','Target h_f','Location','southeast');
xlabel('r [ft]'); ylabel('h [ft]');
grid on;
% Numeric Calculations
dH_A = x_o(end,2) - target_h;
dW_A = x_o(end,4) - target_w;
```

```
dH_B = psi_curr(1);
dW_B = psi_curr(2);
fprintf('\nPART A Results: ν2 = %.15g   ν4 = %.15g   delta_h = %.15g   delta_w
= %.15g\n', ...
       nu_opt(1), nu_opt(2), dH_A, dW_A);
fprintf('PART B Results: ν2 = %.15g   ν4 = %.15g   delta_h = %.15g   delta_w =
%.15g   Iterations = %d\n\n', ...
       nu_update(1), nu_update(2), dH_B, dW_B, iter);
```