**AE 551: Introduction to Optimal Control** **(Due: 2020/04/24)**

# Homework #5 Submission

*Instructor:* Min-Jea Tahk *Name:* Joshua Julian Damanik, *Student ID:* 20194701

**Problem**

## Equation of Motion

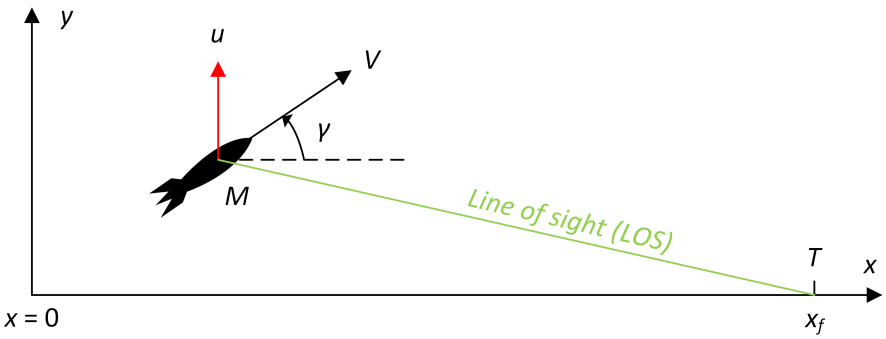



Figure 1: Problem definition

Missile position and velocity is defined as

$$P = (x, y) \tag{1}$$
$$V = (V_x, V_y), \quad V_x = const \tag{2}$$

Guidance input

$$U = (0, u) \tag{3}$$

Equations of Motion

$$\dot{x} = V_x, \quad \dot{V}_x = 0, \quad x(0) = 0, \quad V_x(0) = V_{x_0} \tag{4}$$
$$\dot{y} = V_y, \quad \dot{V}_y = 0, \quad y(0) = 0, \quad V_y(0) = V_{y_0} \tag{5}$$

## Cost Function

$$\min J = \frac{c}{2}\left[y(t_f)\right]^2 + \frac{1}{2}\int^{t_f}\left[ay(t)^2 + u(t)^2\right] dt \tag{6}$$

We want:

1. the missile to stay close to the initial LOS
2. the missile to save the control energy
3. the missle to hit the target at the end

## Problem Formulation II

The cost function for this homework is defined as:

$$\min J = \frac{1}{2}\sum_{k=0}^{N-1} u_k^2 \tag{7}$$

with parameter vector $X$:

$$X = [u_0, u_1, \ldots, u_{N-1}, v_{y_1}, v_{y_2}, \ldots, v_{y_N}, y_1, y_2, \ldots, y_N]^T \tag{8}$$

Constraints:

1. Equation of motion

$$y_k = y_{k-1} + v_{y,k-1}\Delta t, \quad n = 1, 2, \ldots, N \tag{9}$$
$$v_{y,k} = v_{y,k-1} + u_{k-1}\Delta t, \quad n = 1, 2, \ldots, N \tag{10}$$

2. Terminal condition

   (a) CASE A:

$$y_N = 0 \tag{11}$$

   for intercept

   (b) CASE B:

$$\begin{aligned} y_N &= 0 \\ v_{y_N} &= 0 \end{aligned} \tag{12}$$

   for intercept with $v_f = 0$

Instead of integrating the equation of motion, the dynamics of the missile is treated as EQ constraints. Optimal control theory treats it as dynamic constraints. Use $N >= 5$ and use your own code except for matrix inversion and null space calculation.

***(Solution)*** Full code used for this homework is attached at appendix and also can be accessed from https://github.com/joshuadamanik/Homework-5

The problem in this homework is solved by using Augmented Lagrangian Method (ALM) with quadratic penalty function. Constraints from equation (9), (10), and (11) for CASE A or (12) for CASE B are rewritten as a linear function.

$$c(X) = AX - B \tag{13}$$

The ALM optimization problem is defined by a augmented lagrangian function

$$\begin{aligned} L_A(X, \lambda) &= J(X) + \lambda^T c(X) + \rho c(X)^T c(X) \\ L_A(X, \lambda) &= \frac{1}{2} X^T R X + \lambda^T (AX - B) + \rho [AX - B]^T [AX - B] \end{aligned} \tag{14}$$

where $J(X) = \frac{1}{2} X^T R X = \frac{1}{2} \sum_{k=0}^{N-1} u_k^2$ and $\lambda$ is a lagrange multiplier calculated iteratively by $\lambda_{k+1} = \lambda_k + 2\rho c(\bar{X})$.

## Result

For known $\lambda_k$, I find the minimum value of $L_A(X, \lambda_k)$ w.r.t X using Quasi-Newton BFGS method and fibonnaci line search algorithm with search interval $\epsilon = 10$ and Fibonacci number parameter $k = 15$.

## CASE A

For this case, we applied the constraint on equation (11), where the final position of the missile need to be zero no matter of the velocity. The penalty parameter value $\rho$ is chosen as $\rho = 0.1$. The number of parameters use is $N = 50$. The results are shown in figure #.

## CASE B

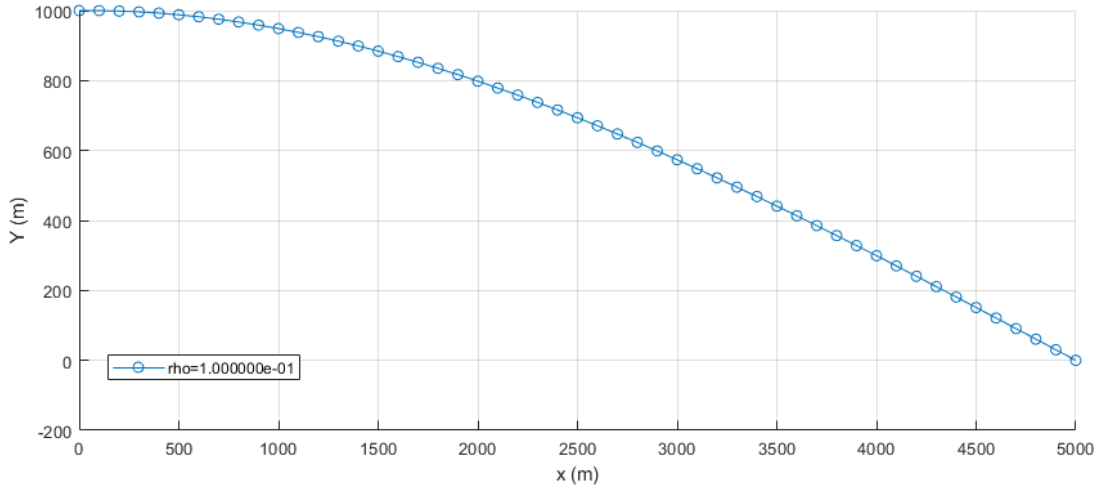
For this case, we need to consider both the vertical position and vertical speed at final position, which is defined as equation (12). The result is shown in figure #.

## Discussion

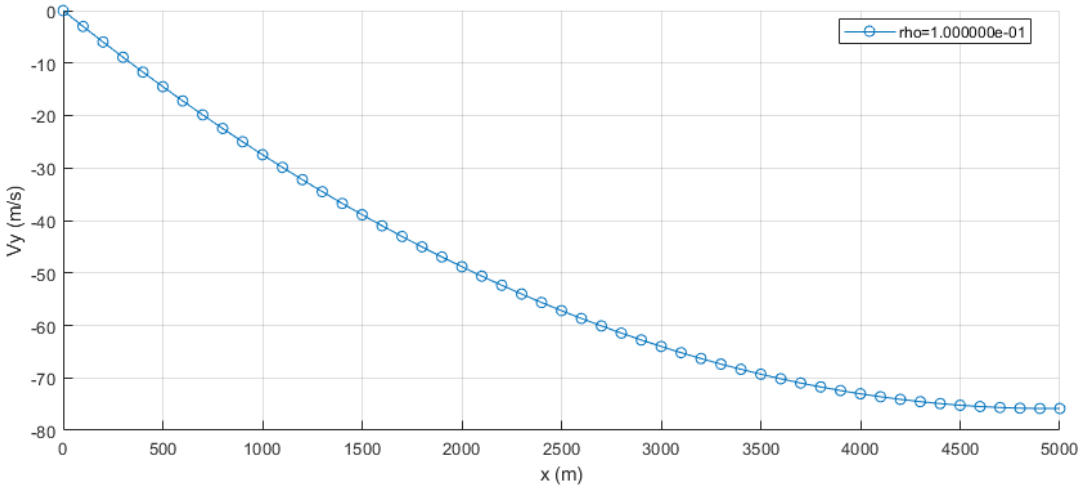
Comparing to previous homework, which is unconstrained problem, constrained problem took a significant amount of time to compute. One of the reason is because we need to maximize the objective function on equation (14) by searching two variables, parameter vector $X$ (equation (8)) and Lagrangian multiplier $\lambda$. The speed of convergence in this problem depends on the penalty value $\rho$. I tried to tune the variable to several values, but it is hard to analytically find the optimum value of $\rho$ that optimize the speed of convergence. However, selection of $\rho = 0.1$ for both cases give an acceptable convergence speed.

If we look at figure 2(a), we can see that the vertical position of the missile is slowly decreasing and finally it reaches $y \approx 0$ at final. This is the result of implementation on $y_N = 0$ constraint on the optimization problem. However, the vertical velocity of the missile doesn't reach 0 at final position. The algorithm only tries to minimize the input given to the missile. As shown in figure 2(c), the minimum input is found to be a linear function w.r.t time with lowest slope as possible.

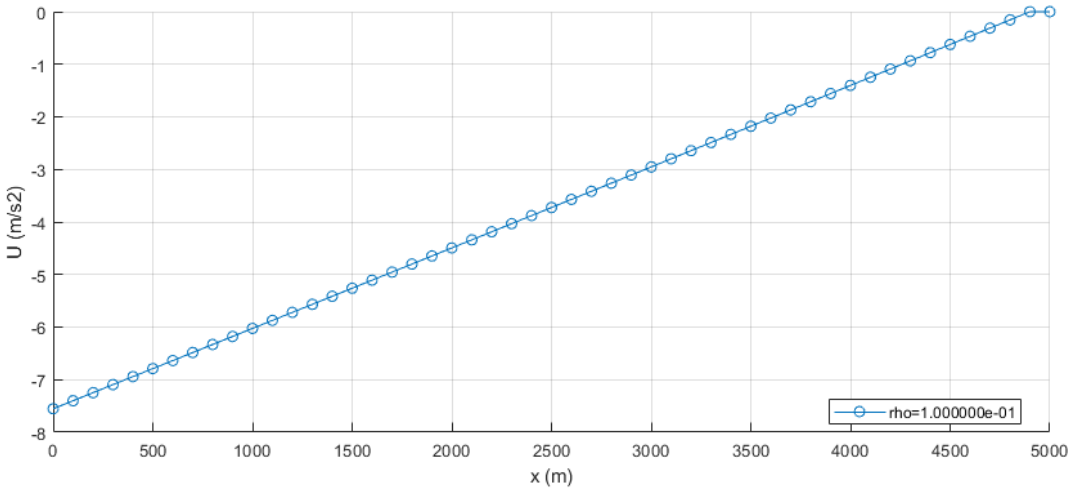
Contrary, for the case B, both vertical position and velocity (figure 3(a) and (b) respectively) at final is approximately zero. It happened because we put two constraints both for vertical velocity and speed (equation (12)). The speed curve is a quadratic function and vertical input 3(c) is a linear function with $u_{N/2} = 0$.




(a) Vertical position $y_k$

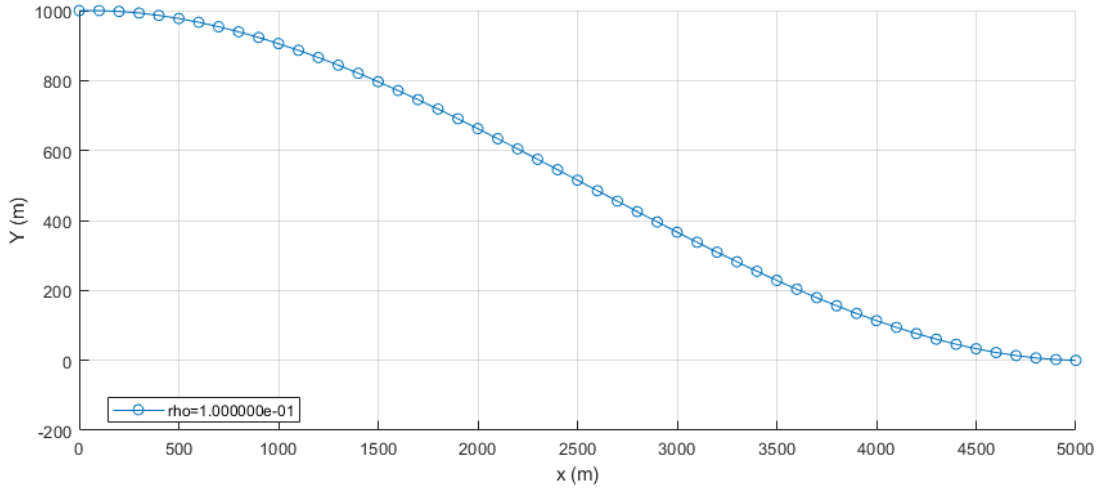



(b) Vertical speed $v_{y_k}$

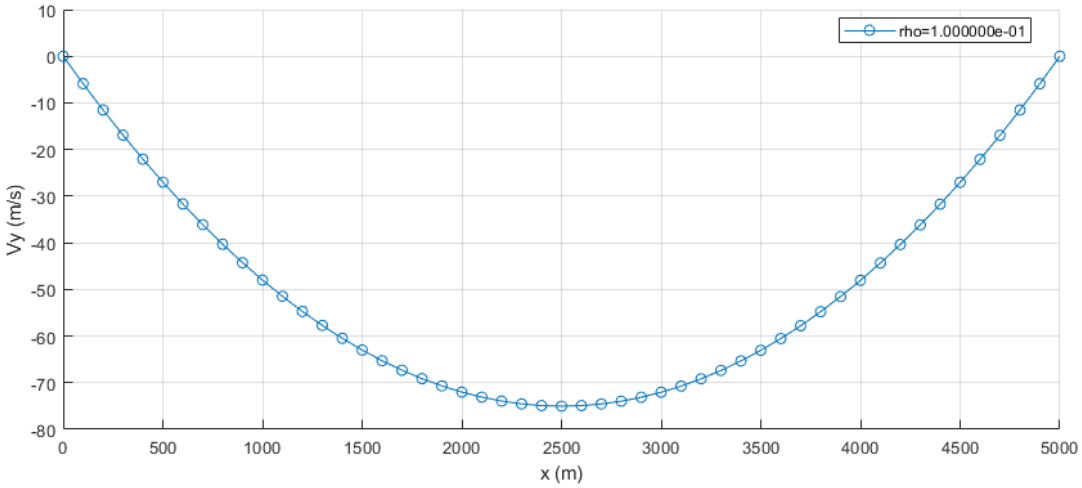



(c) Vertical input $u_k$

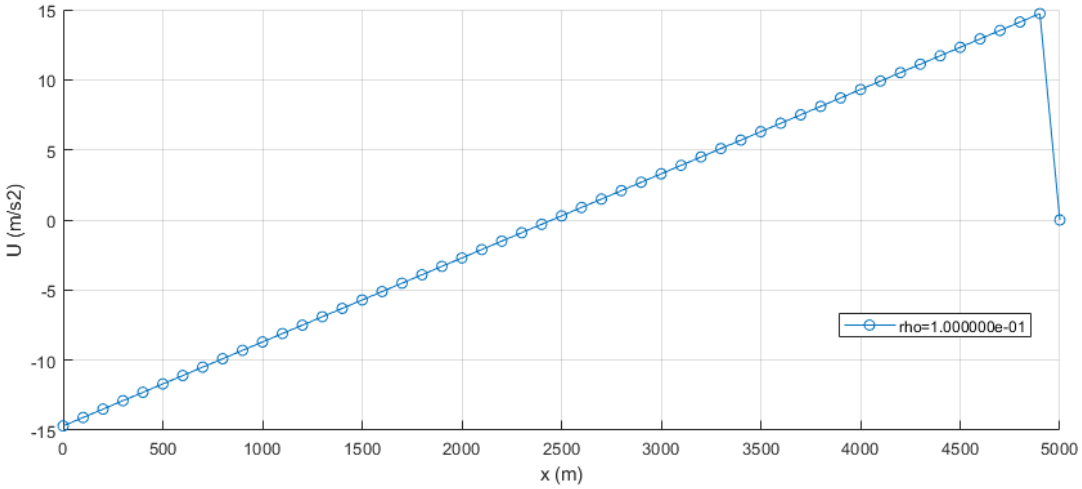
Figure 2: Simulation result for Case A with $N = 50$




(a) Vertical position $y_k$




(b) Vertical speed $v_{y_k}$




(c) Vertical input $u_k$

Figure 3: Simulation result for Case B with $N = 50$

## Appendix

Listing 1: Main code for homework 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HOMEWORK #5
% Joshua Julian Damanik (20194701)
% AE551 - Introduction to Optimal Control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear, clc, close all;
addpath('lib');

X0 = [1000, 0]';
N = 5;


eps = 10;
k = 15;

marker_list = {'o-', '+-', '^-', 'v-',  'x-', '.-'};
legends = {};

plot_data = zeros(N+1,3,1);

rho = 0.1;
delta_rho = 2;

iter = 0;


X = zeros(3*N,1);
X(1:N) = X0(1);
X(N+1:2*N) = X0(2);

[A, B, R] = cost_function_param(N, X0);
c = @(X) (A*X-B);
lambda = zeros(length(B),1);

for j = 1:5
    iter = iter + 1;

    La = @(X, lambda) (0.5*X'*R*X + lambda'*(A*X-B)+rho.*((A*X-B))'*(A*X-B));

    last_X_star = X;

    for ii = 1:10000


        f = @(X) La(X,lambda);
        X_star = minimize(X, eps, k, f);

        quasi = quasi_newton_class(length(X));
        for n = 1:1000
            p = quasi.bgfs(X, eps, f);
            [Xa, Xb] = unimodal_interval(X, eps, p, f);
            X_star = fibonacci_search(Xa, Xb, k, f);

            err = norm(X_star - X);
            X = X_star;
            if (err < eps * 0.01)
                break;
            end
        end
```

```
%           X_star = -(2*R+2*rho*(A'*A)) \ (lambda'*A-2*(B'*A))';
%           X = X_star;

        L_star = lambda + 2*rho*c(X_star);
        err = norm(L_star - lambda);
        fprintf('j=%d,ii=%d,err=%.4f\n',j,ii,err);
        if (err < 2*rho*0.01)
            break;
        end
        X = X_star;
        lambda = L_star;
    end

    plot_data(1,1,j) = X0(1);
    plot_data(1,2,j) = X0(2);
    plot_data(2:N+1,1,j) = X_star(1:N);
    plot_data(2:N+1,2,j) = X_star(N+1:2*N);

    plot_data(:,3,j) = [X_star(2*N+1:3*N); 0];

    legends = [legends, {sprintf('rho=%d', rho)}];

    if (A*X-B)'*(A*X-B) <= 0.01
        break;
    end

    rho = rho*delta_rho;

end

title = {'Vertical Position', 'Vertical Speed', 'Vertical Input'};
ylabels = {'Y (m)', 'Vy (m/s)', 'U (m/s2)'};
legend_position = {'NorthEast', 'SouthEast', 'SouthEast'};
for i = 1:3
    figure('Name', title{i}), hold on;
    set(gcf, 'Position', [100*i, 100*i, 1000, 400]);
    for j = 1:iter
        figure(i),plot(5000 / N * (0:N), plot_data(:,i,j), marker_list{mod(j-1,length(marke
    end
    grid on;
    xlabel('x (m)');
    ylabel(ylabels{i});
    legend(legends, 'Location', 'Best');
end
```

Listing 2: Cost function definition

```
function [A, B, R] = cost_function_param(N, X0)
    dt = 20 / N;

    A11 = eye(N) - diag(ones(N-1,1),-1);
    A12 = -dt*diag(ones(N-1,1),-1);
    A13 = zeros(N);
    A1 = [A11, A12, A13];

    A21 = A13;
    A22 = A11;
    A23 = -dt*eye(N);
    A2 = [A21, A22, A23];

    A3 = [zeros(1, N-1), 1, zeros(1, 2*N)];
    A4 = [zeros(1, 2*N-1), 1, zeros(1, N)];
```

```
    A = [A1; A2; A3];%; A4];

    B1 = [X0(1)-X0(2)*dt; zeros(N-1,1)];
    B2 = [X0(2); zeros(N-1,1)];
    B3 = 0;
    B4 = 0;

    B = [B1; B2; B3];%; B4];


    R = blkdiag(zeros(2*N),eye(N));
end
```