AE 551: Introduction to Optimal Control **(Due: 2020/04/17)**

Homework #4 Submission

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

Problem

Equation of Motion

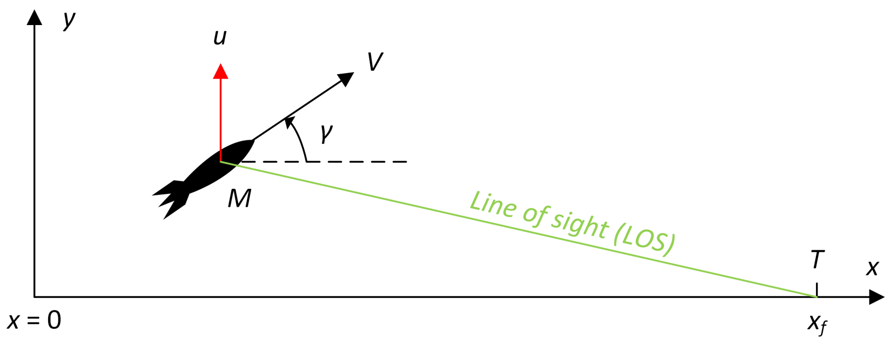


Figure 1: Problem definition

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

Discretization of the Equations of Motion

In this homework, you use the simplest method for discretization, where

$$\dot{X}_{k-1} = f(X_{k-1}, U_{k-1}) \approx \frac{X_k - X_{k-1}}{\Delta t} \tag{7}$$

$$X \triangleq [x, y, \dot{x}, \dot{y}]^T$$

$$x_k = X_{k-1} + \dot{v}_{x,k-1}\Delta t \tag{8}$$

$$y_k = X_{k-1} + \dot{v}_{y,k-1}\Delta t \tag{9}$$

$$v_{x,k} = v_{x,k-1} \tag{10}$$

$$v_{y,k} = v_{y,k-1} + \dot{u}_{k-1}\Delta t \tag{11}$$

Intial conditions (given)

$$x(0) = 0, \quad y(0) = 1000\ m$$
$$v_x(0) = 250\ m/s, \quad v_y(0) = 0$$

Terminal conditions (constraints)

$$x(t_f) = 5000\ m, \quad t_f = 20\ sec$$

There is no *hard* constraint on $y(t_f)$ and $v_y(t_f)$. But hte requirement for target intercept is treated by using the *soft* constraint $\frac{1}{2}[y(t_f)]^2$.

Problem Formulation I: Object parameters control inputs

By discretizing equation (6), we get

$$\min J = \frac{c}{2}y_N^2 + \frac{1}{2}\sum_{k=0}^{N-1}\left(ay_{k+1}^2 + u_k^2\right)dt \tag{12}$$

Parameter vector

$$U = [u_0, u_1, \ldots, u_{N-1}]^T \tag{13}$$

This is an unconstrained problem. The terminal constraint $x(t_f) = 5000\ m$ is imposed by choosing $t_f = 20\ sec$. $y_1, \ldots, y_N$ are obtained from the difference equations of motion.

Choose $N >= 5$. We need to do trade-offs with the values of a and c. Use you 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-4`

By looking at the cost function definition from equation (6), we could see that we need to minimize three components:

- Final height at destination $y(t_f)$
- Height at every time $y(t)$
- Control input given to the missile $u(t)$

The equations of motion used are in the order of 2. Thus we need to define a state vector X consists of position and velocity of missile and use equation (13) as input vector. However the horizontal velocity of the missile v_x is constant all the time, so we can remove horizontal components from the state vector to simplify the calculation. We define the state vector as

$$X_k = \begin{bmatrix} y_k \\ v_{y_k} \end{bmatrix} \tag{14}$$

We can now rewrite the equations (9) and (11) as

$$\begin{aligned} X_k &= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} X_{k-1} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} U_{k-1} \\ X_k &= AX_{k-1} + BU_{k-1} \end{aligned} \tag{15}$$

If we replace the X_{k-1} again and again with equation (15), we can get

$$\begin{aligned} X_k &= AAX_{k-2} + ABU_{k-2} + BU_{k-1} \\ X_k &= AAAX_{k-3} + AABU_{k-3} + ABU_{k-2} + BU_{k-1} \\ &\vdots \\ X_k &= A^k X_0 + A^{k-1}BU_0 + \cdots + ABU_{k-2} + BU_{k-1} \end{aligned} \tag{16}$$

We can then represent X_k, for every $k = (1, 2, \dots, N)$ of the equation (16) in a vector notation as

$$\begin{aligned} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} &= \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix} X_0 + \begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix} U \\ X &= A_N X_0 + B_N U \end{aligned} \tag{17}$$

From now on we can represent X_k for every $k = (1, 2, \dots, N)$ with only knowing initial state X_0 and input vector U. Then, we can also rewrite the cost function of equation (12) with

$$\begin{aligned} \min J &= X^T \begin{bmatrix} A_p & 0 & \dots & 0 & 0 \\ 0 & A_p & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & A_p & 0 \\ 0 & 0 & \dots & 0 & A_p + C_p \end{bmatrix} X + U^T U \\ \min J &= X^T P X + \frac{1}{2} U^T U \end{aligned} \tag{18}$$

where

$$A_p = \begin{bmatrix} \frac{a}{2} & 0 \\ 0 & 0 \end{bmatrix}, \quad C_p = \begin{bmatrix} \frac{c}{2} & 0 \\ 0 & 0 \end{bmatrix}$$

Inserting equation (17) into (18) results to

$$\begin{aligned}
\min J &= X_0^T A_N^T P A_N X_0 + U^T B_N^T P B_N U + \frac{1}{2} U^T U + 2 X_0^T A_k^T P B_k U \\
\min J &= X_0^T (A_N^T P A_N) X_0 + U^T (B_N^T P B_N + \frac{1}{2} I_N) U + 2 X_0^T (A_k^T P B_k) U \\
\min J &= X_0^T Q X_0 + U^T R U + 2 X_0^T S U \qquad (19)
\end{aligned}$$

where

$$\begin{aligned}
Q &= A_N^T P A_N \\
R &= B_N^T P B_N + \frac{1}{2} I_N \\
S &= A_k^T P B_k
\end{aligned}$$

and I_N is an N-by-N identity matrix.

Then, we can find the minimum value of cost function analytically by deriving equation (19) w.r.t U

$$\begin{aligned}
\frac{\partial J}{\partial U} &= 2RU + 2X_0^T S = 0 \\
U &= -R^{-1} S^T X_0 \qquad (20)
\end{aligned}$$

Result

The optimization program used in this homework uses the similar code from Homework #3 assignment. The full code used for this homework assignment is attached at appendix and can also be seen at online repository `https://github.com/joshuadamanik/Homework-4`.

The program uses equation (19) as the optimization objective. The program uses Fibonacci line search algorithm with Fibonacci parameter $k = 15$ and *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) algorithm for determining search direction with search step $\epsilon = 0.01$. The iteration number used is $N = 50$, it means that the time period of control $T = 4$ *sec*.

The program was run with six different variations of cost function parameters a and c. For the first three results, we activated both parameters. And the last three results, we deactivated parameter a to see the response of the result. The result is represented in figure 2.

The optimization program shows acceptably accurate results, as shown in table 1, with average error percentage of 2.8528×10^{-05}%. The analytical results are computed with equation (20).

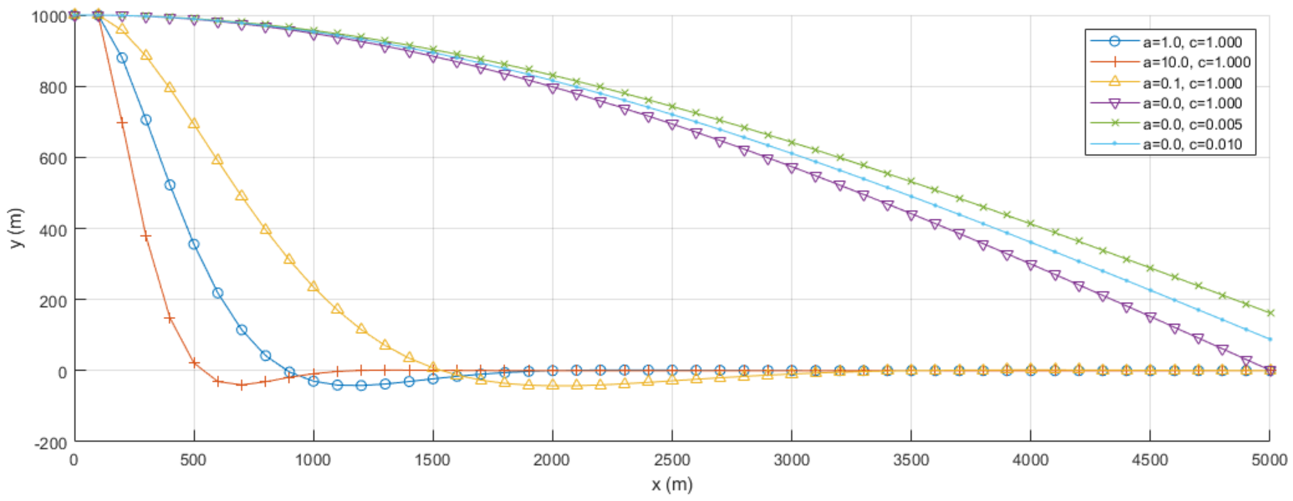
Table 1: Result accuracy comparing to analytical solution

Simulation	$\|\|U^*_{program}\|\|_2$	$\|\|U^*_{analytic}\|\|_2$	% Error
1	930.4266	930.4266	4.2011e-05
2	2151.6758	2151.6758	7.6062e-05
3	395.1907	395.1907	1.0194e-05
4	31.0553	31.0553	2.1561e-05
5	26.0507	26.0508	1.3031e-05
6	28.3462	28.3462	8.3095e-06

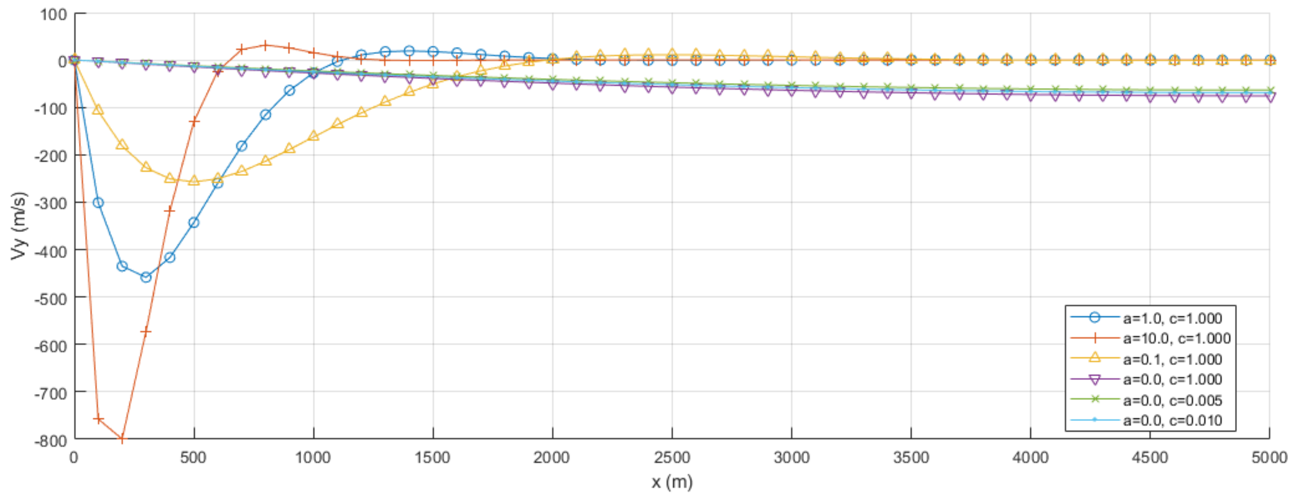
Discussion

For the first result, parameter value is $a = 1$ and $c = 1$. By looking at the dark blue line, the vertical position of the missile is quickly going low to 0. We can see also there is an overshoot where the $y < 0$. However, we can acknowledge this phenomenon by looking at the cost function at equation (6). The second term of the equation tries to minimize vertical position at every time $y_{k+1}, k = 0, 1, \ldots, N-1$. It means that the vertical velocity will get to zero quickly, but the term require us to use as little energy as possible. In result, there is an unavoidable overshoot happened in the result.

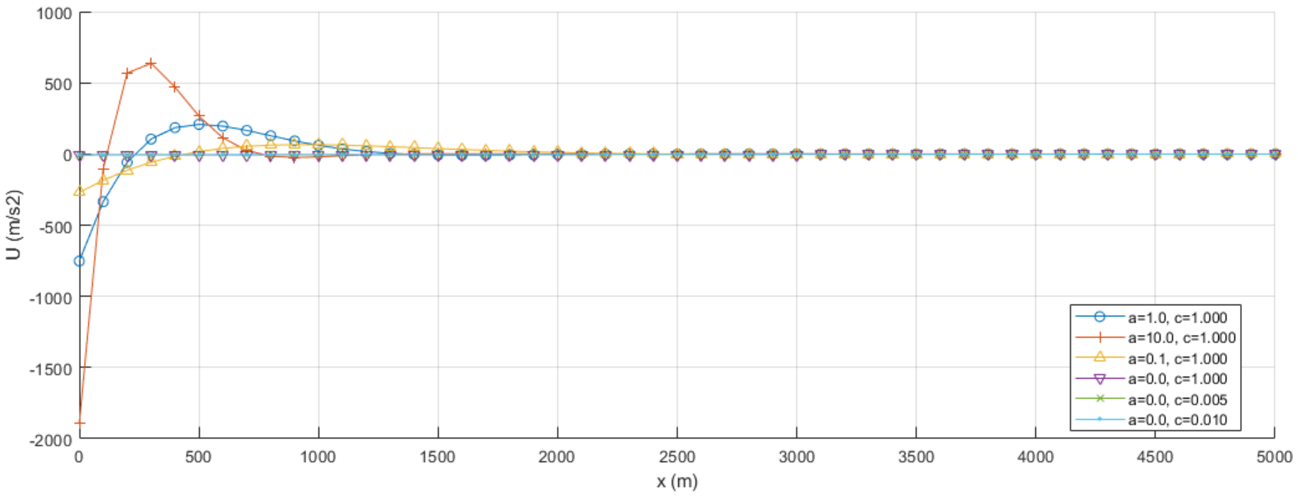
Then we can try to increase the parameter of a to 10. By looking at orange line, we can see that the missile reach $y = 0$ quicker than the previous result. In contrast, by reducing the a to 0.1, we can notice that it took longer for the missile to reach $y = 0$ (yellow line).



(a) Vertical position y_k



(b) Vertical velocity ${v_y}_k$



(c) Vertical input u_k

Figure 2: Simulation result

After that, we can try to set the parameter of a to 0. By this time, with respect to vertical position, only the first term of cost function that will minimize the final position of the missile. By also considering the minimal input given to the missile, third term of cost function, the missile will try to lower its vertical position y slowly until it reach the final position at $y = 0$. We can see this at purple line.

However if we try to reduce the parameter c to 0.01 and 0.001, we can see that the missile fails to reach $y = 0$ (light blue line and green line respectively). So without the optimization of second term, we need to make sure the parameter c is high enough to make sure the missile reach the specified final position.

By comparing the result of optimization with and without optimization of second term, we can see that all of result with second term optimization ($a > 0$) shows faster response than the ones without ($a = 0$) but with considerable amount of overshoot. In practical, we need to avoid this overshoot by introducing some constraints. Looking at the energy required (Figure 2(c)), results with second term optimization also shows a significant increase in the magnitude of input given to the missile at the beginning phase of flight.

Finally, we can conclude that parameter a will make the missile quickly reach the line of sight ($y = 0$) in cost of more energy required and parameter c will determine the accuracy of final location of missile.

Appendix

Listing 1: Main code for homework 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% HOMEWORK #3
% Joshua Julian Damanik (20194701)
% AE551 − Introduction to Optimal Control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

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

%% Test functions

f1 = @(x1,x2) 0.5.*(x1-1).^2 + 10.*(x2-1).^2;
f2 = @(x1,x2) (1-x1).^2 + 100.*(x2-x1.^2).^2;
f3 = @(x1,x2) x1.^2 + 0.5.*x2.^2 + 3;

min_x(:,1) = [1, 1]';
min_x(:,2) = [1, 1]';
min_x(:,3) = [0, 0]';

f = f1;

%% Initialization

x1 = -2;
x2 = -2;

%eps_list = logspace(0,−5,10);
eps_list = 0.1*ones(1,10);

%k_list = 7*ones(1,10);
k_list = 5:14;

method_list = {'newton', 'gradient', 'rank1', 'dfp', 'bgfs'};

if isequal(f, f1)
    fname = 'f1';
elseif isequal(f, f2)
    fname = 'f2';
elseif isequal(f, f3)
    fname = 'f3';
end



for l = 1:length(method_list)
    method = method_list{l};

    fprintf('%s\t%s\n', fname, method);
    fprintf('eps\tk\titer\tX\tY\tError\n');

    for m = 1:min(length(eps_list), length(k_list))

        X = [x1, x2]';
        Xline = X;

        eps = eps_list(m);
        k = k_list(m);
        quasi = quasi_newton_class(eye(2));

        for n = 1:100
```

```
            if strcmp(method, 'gradient')
                %% Steepest Decent
                p = steepest_decent(X, eps, f);
            elseif strcmp(method, 'newton')
                %% Newton's method
                p = newtons_method(X, eps, f);
            elseif strcmp(method, 'rank1')
                p = quasi.rankone(X, eps, f);
            elseif strcmp(method, 'dfp')
                p = quasi.dfp(X, eps, f);
            elseif strcmp(method, 'bgfs')
                p = quasi.bgfs(X, eps, f);
            end

            %% Fibonacci search (Line search)
            [Xa, Xb] = unimodal_interval(X, eps, p, f);
            X_star = fibonacci_search(Xa, Xb, k, f);
            Xline(:,n+1) = X_star;

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

            X = X_star;
        end

        %fprintf('Iteration #%d: (%.4f, %.4f)\n', n, X_star);


        %% Plotting data

        if isequal(f, f1)
            X_star_anal = min_x(:,1);
        elseif isequal(f, f2)
            X_star_anal = min_x(:,2);
        elseif isequal(f, f3)
            X_star_anal = min_x(:,3);
        end

        erms = norm(X_star_anal - X_star);

        fprintf('%e\t%d\t%d\t%.4f\t%.4f\t%e\n', eps, k, n, X_star, erms);
    end
end

function fib = Fib(k)
    list = [0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181
    fib = list(k+1);
end

function X_star = fibonacci_search(Xa, Xb, k0, f)
    fib0 = Fib(k0);

    k = k0 - 2;
    fib_min = 0;
    fib_max = fib0;

    while k > 0
        del_fib = Fib(k);
        fib_range = [fib_min, fib_min + del_fib, fib_max - del_fib, fib_max];
        X_range = Xa + (Xb-Xa) * fib_range / fib0;
        Z_range = f(X_range(1,:), X_range(2,:));
        [~, imin] = min(Z_range);
        X_star = X_range(:,imin);

        if (imin < 3)
            fib_max = fib_range(3);
        else
            fib_min = fib_range(2);
        end
        k = k - 1;
    end

    X = X_star;
end

function g = grad_central_diff(X, eps, f)
    gx = (f(X(1) + eps, X(2)) - f(X(1) - eps, X(2))) / (2*eps);
    gy = (f(X(1), X(2) + eps) - f(X(1), X(2) - eps)) / (2*eps);
    g = [gx, gy]';
end

function G = hess_central_diff(X, eps, f)
    gk = grad_central_diff(X, eps, f);
    gkh_xp = grad_central_diff(X + [eps; 0], eps, f);
    gkh_yp = grad_central_diff(X + [0; eps], eps, f);

    gkh_xn = grad_central_diff(X - [eps; 0], eps, f);
    gkh_yn = grad_central_diff(X - [0; eps], eps, f);
    Y = 1/eps * [gkh_xp-gkh_xn, gkh_yp-gkh_yn];
    G = 0.5*[Y+Y'];
end

function p = newtons_method(X, eps, f)
    g = grad_central_diff(X, eps, f);
    G = hess_central_diff(X, eps, f);
    p = -G\g;
end

function p = steepest_decent(X, eps, f)
    g = grad_central_diff(X, eps, f);
    p = -g;
end

function [Xa, Xb] = unimodal_interval(X, eps, p, f)
    Xa = X;
    Xb = X;
    p = p / norm(p);

    while true
        lastX = X;
        X = X + eps * p;
        Xb = X;

        eps = 1.5*eps;

        if (f(X(1), X(2)) < f(lastX(1), lastX(2)))
            Xa = lastX;
        else
            break;
        end
    end
```

```
end
```

Listing 2: Quasi newton class code

```
classdef quasi_newton_class < handle
    properties
        H
        X_last
        g_last
    end

    methods
        function obj = quasi_newton_class(H0)
            obj.H = H0;
        end

        function p = rankone(obj, X, eps, f)
            g = grad_central_diff(X, eps, f);
            if ~isempty(obj.X_last)
                del_x = X - obj.X_last;
                del_g = g - obj.g_last;

                num = del_x - obj.H * del_g;
                den = del_g'*(del_x-obj.H*del_g);

                obj.H = obj.H + (num * num'./ den);
            end

            p = -obj.H*g;
            obj.X_last = X;
            obj.g_last = g;
        end

        function p = dfp(obj, X, eps, f)
            g = grad_central_diff(X, eps, f);
            if ~isempty(obj.X_last)
                del_x = X - obj.X_last;
                del_g = g - obj.g_last;

                num1 = del_x*del_x';
                den1 = del_x'*del_g;

                num2 = obj.H*del_g;
                den2 = del_g'*obj.H*del_g;

                obj.H = obj.H + num1/den1 - num2*num2'/den2;
            end

            p = -obj.H*g;
            obj.X_last = X;
            obj.g_last = g;
        end

        function p = bgfs(obj, X, eps, f)
            g = grad_central_diff(X, eps, f);
            if ~isempty(obj.X_last)
                del_x = X - obj.X_last;
                del_g = g - obj.g_last;

                obj.H = obj.H + (1 + (del_g'*obj.H*del_g)/(del_g'*del_x))*(del_x*del_x')/(d
                              - (obj.H*del_g*del_x' + (obj.H*del_g*del_x')')/(del_g'*del_x)
            end
            p = -obj.H*g;
            obj.X_last = X;
            obj.g_last = g;
        end

        function Xk = test(obj)
            Xk = isempty(obj.X_last);
        end
    end
end
```