AE 551: Introduction to Optimal Control **(Due: 2020/04/10)**

Homework #3 Submission

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

Problem (a)

Apply one of the quasi-Newton methods to solve the two problems of Homework #2, and compare the results with the gradient method and Newton's method.
You may use the inverse Hessian form of the quasi-Newton methods.

(Solution) Full code used for this homework is attached at appendix and also can be accessed from https://github.com/joshuadamanik/Homework-3

The code used for this homework assignment reused the same code assigned for Homework #2. The line search is done using fibonacci method. But there are some additions for calculating search direction p using Quasi-Newton method. There are 3 Quasi-Newton algorithms used: *Symmetric Rank-One* Update, *Davidon-Fletcher-Powel* (DFP) update, and *Broyden-Fletcher-Goldfarb-Shanno* (BGFS) update, which approximate inverse Hessian matrix by iteration defined by equation (1), (2), and (3) respectively. The search direction is then calculated as $p_k = -H_k g^{(k)}$.

$$H_{k+1}^{Rank1} = H_k + \frac{(\Delta x^{(k)} - H_k \Delta g^{(k)})(\Delta x^{(k)} - H_k \Delta g^{(k)})}{\Delta g^{(k)T}(\Delta x^{(x)} - H_k \Delta g^{(k)}} \tag{1}$$

$$H_{k+1}^{DFP} = H_k + \frac{\Delta x^{(k)} \Delta x^{(k)T}}{\Delta x^{(k)T} \Delta g^{(k)}} - \frac{[H_k \Delta g^{(k)}][H_k \Delta g^{(k)}]^T}{\Delta g^{(k)T} H_k \Delta g^{(k)}} \tag{2}$$

$$H_{k+1}^{BGFS} = H_k + \left(1 + \frac{\Delta g^{(k)T} H_k \Delta g^{(k)}}{\Delta g^{(k)T} \Delta x^{(k)}}\right) \frac{\Delta x^{(k)} \Delta x^{(k)T}}{\Delta x^{(k)T} \Delta g^{(k)}} - \frac{H_k \Delta g^{(k)} \Delta x^{(k)T} + [H_k \Delta g^{(k)} \Delta x^{(k)T}]^T}{\Delta g^{(k)T} \Delta x^{(k)}} \tag{3}$$

Where $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$ and $\Delta g^{(k)} = g^{(k+1)} - g^{(k)}$, $k = 1, 2, 3, ...$
Take note that we can define the initial inverse hessian matrix as Identity matrix of dimension 2×2 ($H_1 = I_2$)

Result

Figure 1 and 2 show the comparison of five different search algorithm using test function f_1 and f_2 respectively. The test functions are defined as:

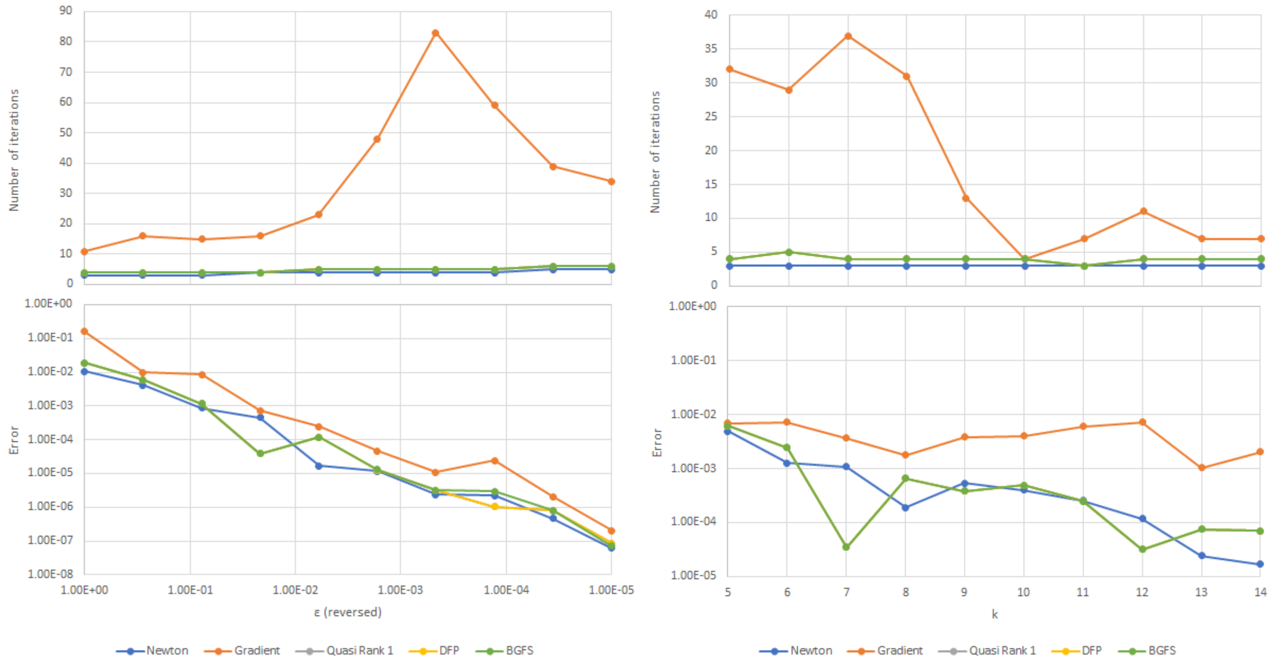
$$f_1(x_1, x_2) = \frac{1}{2}(x_1 - 1)^2 + 10(x_2 - 1)^2 \tag{4}$$

$$f_2(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \tag{5}$$

The search was done by adjusting two search parameters: search step ϵ, and Fibonacci parameter k.
However there are some notes. In figure 1, the Quasi-Newton Rank 1 and DFP shows almost similar result, thus the lines overlap each other. In figure 2, iterations number for Gradient and DFP algorithm seems to saturate at 100 because we limit the number of iterations to 100 in order to prevent long computation time and to present more representative data.

Discussion

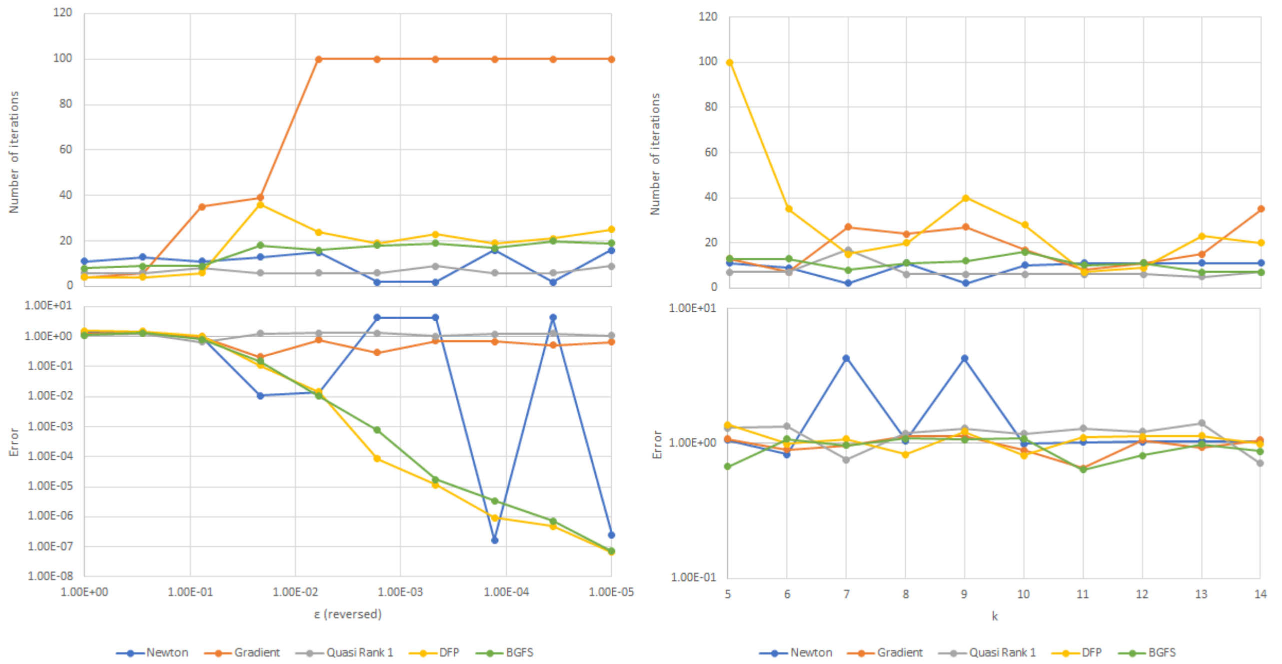
For a simple quadratic function f_1, as seen in figure 1, Newton's method and Quasi-Newton's method outperforms Gradient method most of the time. The Gradient method requires huge number of iteration steps



(a) Variations to ϵ

(b) Variations to k

Figure 1: Simulation results of test function f_1



(a) Variations to ϵ

(b) Variations to k

Figure 2: Simulation results of test function f_2

and produce higher error comparing to other methods. This can also be seen on more complex function f_2. Thus, we can say that knowledge of second derivative can increase the performance of search.

Quasi-Newton's method does not calculate Hessian matrix directly like Newton's method, but both methods can perform the search with similar performance as Newton's method. It is because Quasi-Newton's method can approximate inverse hessian matrix.

On more complex function f_2, as seen in figure 2, some algorithms including Gradient, Newton's method, and Quasi-Newton's method Rank 1 fails to reach solution accurately at small search step ϵ. Even tough DFP method requires high number of iterations for low fibonacci number, both DFP and BGFS methods, which is Quasi-Newton's method rank 2, shows outperform other methods, seen as lower error. From here we can show that even though Quasi-Newton's method rank 1 can achieve acceptable result for simple function like f_1, it fails to perform an accurate result, unlike Quasi-Newton's method rank 2, in more complex functions like f_2.

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
```