Numerical Optimization, 2023 Fall
Homework 4

Name: Zhou Shouchen
Student ID: 2021533042

Due 23:59 (CST), Jan. 4, 2024

Problem 1. f is a positive definite quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x}, \quad \boldsymbol{A} \in \mathbb{S}^n_{++}, \boldsymbol{b} \in \mathbb{R}^n,$$

$\boldsymbol{x}^k$ is the current iteration point, $\boldsymbol{d}^k$ is the descent direction. Derive the step size of exact linear search [20pts]

$$\alpha^k = \arg\min_{\alpha>0} f(\boldsymbol{x}^k + \alpha\boldsymbol{d}^k).$$

Let $g(\boldsymbol{x}) = f(\boldsymbol{x}^k + \alpha\boldsymbol{d}^k) = \frac{1}{2}(\boldsymbol{x}^k + \alpha\boldsymbol{d}^k)^T\boldsymbol{A}(\boldsymbol{x}^k + \alpha\boldsymbol{d}^k) + \boldsymbol{b}^T(\boldsymbol{x}^k + \alpha\boldsymbol{d}^k)$.

Problem 2. Prove that $f : \mathbb{R}^n \to \mathbb{R}$ is affine if and only if f is both convex and concave. [20pts]

Problem 3. Solve the optimal solution of the Rosenbrock function

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2,$$

using MATLAB programming to implement three algorithms (each 20pts): gradient descent (GD) method, Newton method, and Quasi-Newton methods (either rank-1, DFP or BFGS). You are required to print iteration information of last 10 steps: including objective, step size, residual of gradient. Technical implementation: explain how to choose the step size, how to set the termination criteria, how to choose the initial point, the value of the required parameters, converge or not and convergence rate. (paste the code in the pdf to submit it, no need to submit the source code) [60pts]

Rosenbrock.m

```
% f(x, y) = (1-x)^2+100(y-x^2)^2
function [val, nabla] = Rosenbrock(x,y)
    val = (1 - x) ^ 2 + 100 * (y - x ^ 2) ^ 2;
    nabla = [-2 * (1 - x) - 400 * x * (y - x ^ 2); 200 * (y - x ^ 2)];
end
```

print_info.m

```
function print_info(obj, step_size, grad_res)
    for i = max(1 , length(obj) - 10):length(obj)
        fprintf('Step %d: obj = %f, step size = %f, gradient residual = %f\n', i, obj(i), step_size(i),
         grad_res(i));
    end
end
```

plot_trace.m

```
function plot_trace(points, id)
    % plot the contour of of Rosenbrock function
    % then plot the trace of the points
    % points is a matrix of size (n, 2)
    % each row is a point
    % the first column is x, the second column is y
    % the last point is marked with a red star

    % plot the contour of Rosenbrock function
    x = linspace(-2, 2, 100);
    y = linspace(-1, 3, 100);
    [X, Y] = meshgrid(x, y);
    Z = 100 * (Y - X .^ 2) .^ 2 + (1 - X) .^ 2;
    figure(id);
    contour(X, Y, Z, 20);
    hold on;

    % plot the trace of the points
    plot(points(:, 1), points(:, 2), 'b-o');

    % mark the last point with a red star
    plot(points(end, 1), points(end, 2), 'r*');

    % set the axis
    axis([-2, 2, -1, 3]);
end
```

gradient_descent.m

```
function gradient_descent(init_point)
    disp('
         ====================================================================
         ')
    disp('============== start gradient descent method ==============')

    % [x, y] = init_point;
    x = init_point(1);
    y = init_point(2);

    points = [[x, y]];
    obj = [];
    step_size = [];
    grad_res = [];
    while (1)
        % update the value of x and y
        [val, nabla] = Rosenbrock(x, y);
        % [x, y] = [x, y] - 0.001 * nabla;
        x = x - 0.001 * nabla(1);
        y = y - 0.001 * nabla(2);

        points = [points; [x, y]];
        obj = [obj, val];
        step_size = [step_size, 0.001];
        grad_res = [grad_res, norm(nabla)];

        [newval, new_nabla] = Rosenbrock(x, y);
        if (abs(newval - val) < 0.0001)
            break;
        end
    end

    print_info(obj, step_size, grad_res);
    plot_trace(points, 1)

    disp('============= finished gradient descent method =============')
    disp('
        ==============================================================
        ')
end
```

Newton_method.m

```
function Newton_method(init_point)
    disp('
        ==============================================================
        ')
    disp('=================== start Newton method ===================')

    % [x, y] = init_point;
    x = init_point(1);
    y = init_point(2);

    points = [[x, y]];
    obj = [];
    step_size = [];
    grad_res = [];


    print_info(obj, step_size, grad_res);
    plot_trace(points, 2)

    disp('================== finished Newton method ==================')
    disp('
        ==============================================================
        ')
end
```

Quasi_Newton_method.m

```
function Quasi_Newton_method(init_point)
    disp('
        ===========================================================
        ')
    disp('================ start Quasi-Newton method ================')

    % [x, y] = init_point;
    x = init_point(1);
    y = init_point(2);

    points = [[x, y]];
    obj = [];
    step_size = [];
    grad_res = [];


    print_info(obj, step_size, grad_res);
    plot_trace(points, 3)

    disp('=============== finished Quasi-Newton method ===============')
    disp('
        ============================================================
        ')
end
```

output

```
asfafsafafaas
```