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

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

Since $\boldsymbol{A} \in \mathbb{S}^n_{++}$, i.e. $\boldsymbol{A}$ is symmetric, so

$$(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{x}^k = (\boldsymbol{x}^k)^T\boldsymbol{A}\boldsymbol{d}^k$$

So

$$\frac{\partial g}{\partial \alpha} = \alpha(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k + (\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{b}^T\boldsymbol{d}^k$$

And

$$\frac{\partial^2 g}{\partial \alpha^2} = (\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k$$

Since $\boldsymbol{A} \in \mathbb{S}^n_{++}$, which means that $\boldsymbol{A}$ is a positive defined matrix, i.e.

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} > 0$$

So

$$\forall \boldsymbol{d}^k, \frac{\partial^2 g}{\partial \alpha^2} = (\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k > 0$$

So $g(\alpha)$ is a convex function. In order to find the minimum point $\alpha^k = \arg\min_{\alpha>0} g(\alpha)$, we just need to let the gradient to be 0. i.e.

$$\frac{\partial g}{\partial \alpha}(\alpha^k) = 0$$

So

$$\alpha^k(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k + (\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{b}^T\boldsymbol{d}^k = 0$$

So

$$\alpha^k = -\frac{((\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{b}^T\boldsymbol{d}^k)}{(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k}$$

Since we have known that $(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k > 0$, so the α^k is a legal solution.
So abovel all, the step size of exact linear search is

$$\alpha^k = -\frac{((\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{b}^T\boldsymbol{d}^k)}{(\boldsymbol{d}^k)^T\boldsymbol{A}\boldsymbol{d}^k}$$

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

1. sufficiency:

If f is affine, then we can write f as $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, where $\boldsymbol{w} \in \mathbb{R}^n, b \in \mathbb{R}$.
Then we have

$$\nabla f(\boldsymbol{x}) = \boldsymbol{w}$$

and

$$\nabla^2 f(\boldsymbol{x}) = \mathbf{0}$$

So

$$\nabla^2 f(\boldsymbol{x}) \succeq \mathbf{0}, \nabla^2 f(\boldsymbol{x}) \preceq \mathbf{0}$$

So f is both convex and concave.

2. necessity:

If f is convex and concave, then we have:
since f is concave, so

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2, \theta \in [0,1], f(\theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2) \geq \theta f(\boldsymbol{x}_1) + (1-\theta) f(\boldsymbol{x}_2)$$

since f is convex, so

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2, \theta \in [0,1], f(\theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2) \leq \theta f(\boldsymbol{x}_1) + (1-\theta) f(\boldsymbol{x}_2)$$

So we have

$$\forall \boldsymbol{x}_1, \boldsymbol{x}_2, \theta \in [0,1], f(\theta \boldsymbol{x}_1 + (1-\theta)\boldsymbol{x}_2) = \theta f(\boldsymbol{x}_1) + (1-\theta) f(\boldsymbol{x}_2)$$

Which is exactly the definition of affine function.
So f is affine.

So above all, we have proved that $f : \mathbb{R}^n \to \mathbb{R}$ is affine if and only if f is both convex and concave.

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]

1. Gradient descent method:

set step size to be constant: $\alpha^k = 0.002$.
direction: $\boldsymbol{d}^k = -\nabla f(\boldsymbol{x}^k)$.
termination criteria: if the current point's gradiant's norm $\left\|\nabla f(\boldsymbol{x}^k)\right\|_2 < 10^{-4}$, then we regard it converges.
initial point: $\boldsymbol{x}^0 = (x, y)$, where x and y are randomly uniformed generated in the range $[-1, 1]$.
There is no additional parameters.
The method converged, and from the variation of output, we can see that the convergence rate is linear convergence.

2. Newton method:

step size: $\alpha^k = 1$.
direction: $\boldsymbol{d}^k = -\nabla^2 f(\boldsymbol{x}^k)^{-1}\nabla f(\boldsymbol{x}^k)$.
termination criteria: if the current point's gradiant's norm $\left\|\nabla f(\boldsymbol{x}^k)\right\|_2 < 10^{-4}$, then we regard it converges.
initial point: $\boldsymbol{x}^0 = (x, y)$, where x and y are randomly uniformed generated in the range $[-1, 1]$.
There is no additional parameters.
The method converged, and from the variation of output, we can see that the convergence rate is quadratic convergence.

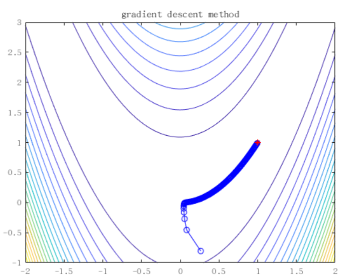
3. Quasi-Newton method:

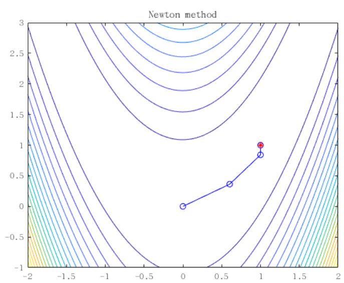
step size: we can use Armijo linear seach to find a suitable α^k: setting $\alpha^k = 1$, and $\gamma = 0.5$, and let α^k constantly times γ until the Armijo condition: $f(\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k) \le f(\boldsymbol{x}^k) + \alpha^k \gamma \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k$ is satisfied.
direction: $\boldsymbol{d}^k = -H_k^{-1}\nabla f(\boldsymbol{x}^k)$, where H_k is generated by BFGS method.
termination criteria: if the current point's gradiant's norm $\left\|\nabla f(\boldsymbol{x}^k)\right\|_2 < 10^{-4}$, then we regard it converges.
initial point: $\boldsymbol{x}^0 = (x, y)$, where x and y are randomly uniformed generated in the range $[-1, 1]$.
The original H_0^{-1} is set to be I_2.
As for the update of H_k^{-1}, we set $\boldsymbol{s}_k = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k$ and $\boldsymbol{y}^k = \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k)$, and we use the BFGS method to update H_k^{-1}.
i.e. if $\boldsymbol{s}_k{}^\top \boldsymbol{y}_k > 0$, then we update $H_{k+1}^{-1} = (I_2 - \frac{\boldsymbol{s}_k \boldsymbol{y}_k{}^\top}{\boldsymbol{y}_k{}^\top \boldsymbol{s}_k}) H_k^{-1} (I_2 - \frac{\boldsymbol{y}_k \boldsymbol{s}_k{}^\top}{\boldsymbol{y}_k{}^\top \boldsymbol{s}_k}) + \frac{\boldsymbol{s}_k \boldsymbol{s}_k{}^\top}{\boldsymbol{y}_k{}^\top \boldsymbol{s}_k}$.
Ohterwise it remains: $H_{k+1}^{-1} = H_k^{-1}$.
The method converged, and from the variation of output, we can see that the convergence rate is superlinear convergence.

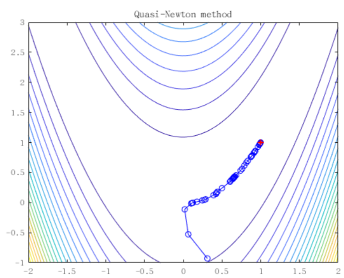
The output information and the trace of the three methods are as follows(it seems that the format of output is not aligned in latex, but it worked well in matlab file...):

output

```
============================================================
=============== start gradient descent method ===============
 k       x_k       y_k    f(x_k, y_k) step size    ||d||
==== ========== ========== ========== ========== ==========
9873 9.9989e-01 9.9977e-01 1.2709e-08 2.0000e-03 1.0068e-04
9874 9.9989e-01 9.9977e-01 1.2689e-08 2.0000e-03 1.0060e-04
9875 9.9989e-01 9.9977e-01 1.2668e-08 2.0000e-03 1.0052e-04
9876 9.9989e-01 9.9977e-01 1.2648e-08 2.0000e-03 1.0044e-04
```








```
9877 9.9989e−01 9.9978e−01 1.2628e−08 2.0000e−03 1.0036e−04
9878 9.9989e−01 9.9978e−01 1.2608e−08 2.0000e−03 1.0028e−04
9879 9.9989e−01 9.9978e−01 1.2588e−08 2.0000e−03 1.0020e−04
9880 9.9989e−01 9.9978e−01 1.2567e−08 2.0000e−03 1.0012e−04
9881 9.9989e−01 9.9978e−01 1.2547e−08 2.0000e−03 1.0004e−04
9882 9.9989e−01 9.9978e−01 1.2527e−08 2.0000e−03 9.9958e−05
9883 9.9989e−01 9.9978e−01 1.2507e−08     −−−−        −−−−
============= finished gradient descent method =============
==========================================================
=========================================================
=================== start Newton method ===================
k     x_k         y_k     f(x_k, y_k) step size    ||d||
= ========== ========== ========== ========== ==========
1 0.0000e+00 0.0000e+00 0.0000e+00 1.0000e+00 3.3601e+02
2 6.0241e−01 3.6289e−01 1.1613e+02 1.0000e+00 7.9437e−01
3 9.9973e−01 8.4159e−01 1.5808e−01 1.0000e+00 7.0584e+01
4 9.9974e−01 9.9948e−01 2.4921e+00 1.0000e+00 5.2493e−04
5 1.0000e+00 1.0000e+00 6.8896e−08 1.0000e+00 3.0811e−05
6 1.0000e+00 1.0000e+00 4.7467e−13     −−−−        −−−−
================== finished Newton method ==================
==========================================================
=========================================================
================ start Quasi−Newton method ================
 k     x_k         y_k     f(x_k, y_k) step size    ||d||
== ========== ========== ========== ========== ==========
40 9.8100e−01 9.6149e−01 1.7181e−03 1.0000e+00 3.5169e−01
41 9.8457e−01 9.6871e−01 4.3737e−04 1.0000e+00 2.6875e−01
42 9.8695e−01 9.7348e−01 2.8300e−04 1.0000e+00 2.4288e−01
43 9.9422e−01 9.8801e−01 2.0630e−04 2.5000e−01 1.9886e−01
44 9.9809e−01 9.9578e−01 5.5454e−05 1.0000e+00 1.7670e−01
45 9.9890e−01 9.9761e−01 1.9908e−05 2.5000e−01 8.3814e−02
46 9.9897e−01 9.9790e−01 4.8890e−06 1.0000e+00 2.1574e−02
47 9.9914e−01 9.9826e−01 1.3252e−06 1.0000e+00 7.9361e−03
48 9.9954e−01 9.9908e−01 7.8646e−07 5.0000e−01 3.1274e−03
49 9.9998e−01 9.9995e−01 2.1516e−07 5.0000e−01 3.7022e−04
50 9.9999e−01 9.9998e−01 6.2918e−10     −−−−        −−−−
=============== finished Quasi−Newton method ===============
==========================================================
```

And the codes are as follows:

Rosenbrock.m

```
% f(x, y) = (1−x)^2+100(y−x^2)^2
function [val, nabla, Hessian] = 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)];
    Hessian = [2 − 400 ∗ y + 1200 ∗ x ^ 2, −400 ∗ x; −400 ∗ x, 200];
end
```

print_info.m

```
function print_info(max_iter, obj, points, step_size, grad_res)
    upper = max_iter - 1;
    for i = max(1 , max_iter - 10):upper
        fprintf('%d %.4e %.4e %.4e %.4e %.4e\n', i, points(i, 1), points(i, 2), obj(i), step_size(i), grad_res(
        i));
    end
    fprintf('%d %.4e %.4e %.4e    ----       ----    \n', max_iter, points(max_iter, 1), points(max_iter, 2),
        obj(max_iter));
end
```

plot_trace.m

```
function plot_trace(max_iter, points, id, name)
    points = points(1 : max_iter - 1, :);

    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]);

    % set name
    title(name);

end
```

gradient_descent.m

```
function gradient_descent()
    disp('
        ==============================================================
        ')
    disp('============== start gradient descent method ==============')
    disp(' k        x_k        y_k    f(x_k, y_k) step size    ||d||   ')
    disp('==== ========== ========== ========== ==========
        ==========')

    %  (x, y) are random initial points
    x = rand() * 2 - 1;
    y = rand() * 2 - 1;

    grad_res = zeros(1, 10000);
    obj = zeros(1, 10000);
    step_size = zeros(1, 10000);
    points = zeros(10000, 2);

    [val, grad, Hessian] = Rosenbrock(x, y);
    obj(1) = val;
    grad_res(1) = norm(grad);
    points(1, :) = [x, y];

    for iter = 2:10000
        % update the value of x and y
        [val, grad, Hessian] = Rosenbrock(x, y);
```

```
        % [x, y] = [x, y] - 0.002 * nabla;

        x = x - 0.002 * grad(1);
        y = y - 0.002 * grad(2);

        points(iter, :) = [x, y];
        obj(iter) = val;
        step_size(iter - 1) = 0.002;
        grad_res(iter - 1) = norm(grad);

        if (norm(grad) < 1e-4)
            break;
        end
    end

    print_info(iter, obj, points, step_size, grad_res);
    plot_trace(iter, points, 1, 'gradient descent method');

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

Newton_method.m

```
function Newton_method()
    disp('
        =================================================================
        ')
    disp('=================== start Newton method ===================')
    disp('k     x_k         y_k     f(x_k, y_k) step size    ||d||   ')
    disp('= ========== ========== ========== ========== ==========')


    % (x, y) are random initial points
    x = rand() * 2 - 1;
    y = rand() * 2 - 1;

    grad_res = zeros(1, 100);
    obj = zeros(1, 100);
    step_size = zeros(1, 100);
    points = zeros(100, 2);

    for iter = 2:100
        [val, grad, Hessian] = Rosenbrock(x, y);

        % if Hessian is not invertible, then output error
        if (abs(det(Hessian)) < 1e-7)
            disp(' Error : Hessian matrix is invertible ');
            break;
        end

        d = -inv(Hessian) * grad;
        x = x + d(1);
        y = y + d(2);

        points(iter, :) = [x, y];
        obj(iter) = val;
        step_size(iter - 1) = 1;
        grad_res(iter - 1) = norm(grad);

        if (norm(d) < 1e-4)
            break;
        end
    end

```

```

    print_info(iter, obj, points, step_size, grad_res);
    plot_trace(iter, points, 2, 'Newton method');

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

Quasi_Newton_method.m

```
function Quasi_Newton_method()
    disp('
        ======================================================================
        ')
    disp('================= start Quasi-Newton method =================')
    disp(' k    x_k        y_k     f(x_k, y_k) step size    ||d||    ')
    disp('== ========== ========== ========== ========== ==========')

    % (x, y) are random initial points
    x = rand() * 2 - 1;
    y = rand() * 2 - 1;

    grad_res = zeros(100, 1);
    grad_record = zeros(100, 2);
    obj = zeros(100, 1);
    step_size = zeros(100, 1);
    points = zeros(100, 2);

    [val, grad, hess] = Rosenbrock(x, y);
    obj(1) = val;
    grad_res(1) = norm(grad);
    grad_record(1, :) = [0, 0];
    points(1, :) = [x, y];

    H_inverse = eye(2);
    for iter = 2:100
        [val, grad, hess] = Rosenbrock(x, y);

        d = -H_inverse * grad;

        % use Armijo rule to determine the step size
        alpha = 1;
        gamma = 0.5;
        while (Rosenbrock(x + alpha * d(1), y + alpha * d(2)) > val + gamma * alpha * grad' * d)
            alpha = alpha * gamma;
        end

        x = x + alpha * d(1);
        y = y + alpha * d(2);

        s_k = [x, y] - points(iter - 1, :);
        s_k = s_k';

        y_k = (grad' - grad_record(iter - 1, :))';

        % the condition of update H_k:
        if s_k' * y_k > 0
            H_inverse = (eye(2) - s_k * y_k' / (y_k' * s_k)) * H_inverse * (eye(2) - y_k * s_k' / (y_k'
      * s_k)) + s_k * s_k' / (y_k' * s_k);
        end

        points(iter, :) = [x, y];
        obj(iter) = val;
        step_size(iter - 1) = alpha;
        grad_res(iter - 1) = norm(grad);
```

```
        grad_record(iter, : ) = grad;

        if (norm(d) < 1e-4)
            break;
        end
    end

    print_info(iter, obj, points, step_size, grad_res);
    plot_trace(iter, points, 3, 'Quasi-Newton method');

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

implement.m

```
gradient_descent()
Newton_method()
Quasi_Newton_method()
```