```
%%%%%%%%%%%%%% Main Entrance %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Instruction: Please read through the code and fill in blanks
% (marked by ***). Note that you need to do so for every involved
% function, i.e., m files.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Optional overhead
clear; % Clear the workspace
% Note: for debugging purpose, do not use "clear all"
close all; % Close all windows
clc; %Clear screen

%% Optimization settings
% Here we specify the objective function by giving the function handle to a
% variable, for example:
f = @(x)distance(x); % replace rosenbrock with your objective function
% In the same way, we also provide the gradient and the Hessian of the
% objective:
g = @(x)distanceg(x); % replace accordingly
H = @(x)distanceH(x); % replace accordingly
% Note that explicit gradient and Hessian information is only optional.
% However, providing these information to the search algorithm will save
% computational cost from finite difference calculations for them.

% Specify algorithm
opt.alg = 'gradient';
%opt.alg = 'newton';

% Turn on or off line search. You could turn on line search once other
% parts of the program are debugged.
opt.linesearch = true; % or true

% Set the tolerance to be used as a termination criterion:
opt.eps = 1e-4; % this should be a small number like 1e-3

% Set the initial guess:
x0 = [2; 3]; % this should be a p-dim vector where p is the size of the
% problem

%% Run optimization
% Run your implementation of the gradient descent and Newton's method. See
% gradient.m and newton.m.
if strcmp(opt.alg,'gradient')
    solution = gradient(f,g,H,x0,opt);
elseif strcmp(opt.alg,'newton')
    solution = newton(f,g,H,x0,opt);
end

%% Report
% Implement report.m to generate a report.
report(solution,f);
```

```
function y = distance(x)
    y = (2-2*x(1)-3*x(2))^2 + x(1)^2 + (x(2)-1)^2;
```

```
function g = distanceg(x)
    g = [-4*(2-2*x(1)-3*x(2)) + 2*x(1); -6*(2-2*x(1)-3*x(2)) + 2*(x(2)-1)];
```

```
function H = distanceH(x)
    H = [10 12;12 20];
```

```
%%%%%%%%%%%%%% Gradient Descent Implementation %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function solution = gradient(f,g,H,x0,opt)
    % Set initial conditions
     x = x0; % Set current solution to the initial guess
     iter = 0; % Set iteration counter to 0

    % Initialize a structure to record search process
     solution = [];

    % Calculate the norm of the gradient
     gnorm = norm(g(x),2); % this needs to be a scalar

    % Set the termination criterion:
    while gnorm>opt.eps % if not terminated
        iter = iter + 1

        % save current step
        solution.x([1,2],iter) = x;
        % solution.x is an array of solutions, i.e., a matrix
        % opt.linesearch switches line search on or off.
        % You can first set the variable "a" to different constant values and see how↙
it
        % affects the convergence.
        if opt.linesearch
            a = lineSearch1(f,g,H,x,opt);
        else
            a = 0.001;
        end


        % Gradient descent:
        d = -1*g(x);
        x = x + a*d; % update x based on gradient info
        disp(x);
        % Update termination criterion:
        gnorm = norm(g(x),2); % update the norm of gradient
    end
```

```
%%%%%%%%%%%%%% Newton's Method Implementation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function solution = newton(f,g,H,x0,opt)
    % Set initial conditions
     x = x0; % Set current solution to the initial guess
     iter = 0; % Set iteration counter to 0
     solution.x([1,2],1) = x;
    % Calculate the norm of the gradient
     gnorm = norm(g(x),2);

    while gnorm>opt.eps % if not terminated
        iter = iter + 1;

        % opt.linesearch switches line search on or off.
        % You can first set the variable "a" to different constant values and see how
it
        % affects the convergence.
        if opt.linesearch
            a = lineSearch1(f,g,H,x,opt);
        else
            a = 0.001;
        end

        % Newton's method:
         x = x - a*inv(H(x))*g(x);

        % save current step
         solution.x([1,2],iter+1) = x;

        % Update termination criterion:
         gnorm = norm(g(x),2);
    end
    disp(x);
    disp(iter);
```

```
% Armijo line search
function a = lineSearch1(f,g,H,x,opt)
    t = 0.1; % scale factor on current gradient: [0.01, 0.3]
    b = 0.55; % scale factor on backtracking: [0.1, 0.8]
    a = 1; % maximum step length
    G = feval(g,x);

    % Calculate the descent direction D for gradient or newton
    if strcmp(opt.alg,'gradient')
         D = -1*G;
    elseif strcmp(opt.alg,'newton')
         D = -1*inv(H(x))*G;
    end

    % terminate if line search takes too long
    count = 0;
    while f(x+a*D) > f(x)+t*a*G'*D
        % stop if condition satisfied
        % implement Armijo's criterion here
        % perform backtracking
        a = b*a
        count = count + 1;
    end
    disp(a);
    disp(count);
end
```

```
%%%%%%%%%%%%%% Generate Report  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function report(solution,f)
    figure; % Open an empty figure window
    hold on; % Hold on to the current figure

    % Draw a 2D contour plot for the objective function
    % You can edit drawing parameters within the file: drawContour.m
    drawContour(f);

    % Plot the search path
    x = solution.x;
    iter = size(x,2);
    plot(x(1,1),x(2,1),'.y','markerSize',20);
    str1 = [num2str(x(1,1)),',',num2str(x(2,1))];
    text(x(1,1),x(2,1),str1);
    for i = 2:iter
        % Draw lines. Type "help line" to see more drawing options.
        line([x(1,i-1),x(1,i)],[x(2,i-1),x(2,i)],'Color','y');
        plot(x(1,i),x(2,i),'.y','markerSize',20);
    end
    title('Contour plot');
    xlabel('x2');
    ylabel('x3');
    str2 = [num2str(x(1,i)),',',num2str(x(2,i))];
    text(x(1,i),x(2,i),str2);
    % Plot the convergence
    F = zeros(iter,1);
    for i = 1:iter
        F(i) = feval(f,x(:,i));
    end
    figure;
    plot(1:iter, log(F-F(end)+eps),'k','lineWidth',3);
    title('Convergence Plot');
    xlabel('Iteration');
    ylabel('Log Convergence');
```