```
%%%%%%%%%%%%%%% Sequential Quadratic Programming Implementation %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function solution = mysqp(f, df, g, dg, x0, opt)
    % Set initial conditions

     x = x0; % Set current solution to the initial guess

    % Initialize a structure to record search process
    solution = struct('x',[]);
    solution.x = [solution.x, x]; % save current solution to solution.x

    % Initialization of the Hessian matrix
     W = eye(length(x0));             % Start with an identity Hessian matrix

    % Initialization of the Lagrange multipliers
     mu_old = [0, 0];    % Start with zero Lagrange multiplier estimates

    % Initialization of the weights in merit function
     w = abs(mu_old);         % Start with zero weights

    % Set the termination criterion
     gnorm = norm(df(x) + mu_old*dg(x), 2); % norm of Largangian gradient

    while gnorm>opt.eps % if not terminated

        % Implement QP problem and solve
        if strcmp(opt.alg, 'myqp')
            % Solve the QP subproblem to find s and mu (using your own method)
            [s, mu_new] = solveqp(x, W, df, g, dg);
        else
            % Solve the QP subproblem to find s and mu (using MATLAB's solver)
            qpalg = optimset('Algorithm', 'active-set', 'Display', 'off');
            [s,~,~,~,lambda] = quadprog(W,[df(x)]',dg(x),-g(x),[], [], [], [], [],↙
qpalg);
            mu_new = lambda.ineqlin;
        end

        % 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, w] = lineSearch(f, df, g, dg, x, s, mu_old, w);
        else
            a = 0.0001;
        end

        % Update the current solution using the step
         dx = s;               % Step for x
         %x = x + dx;             % Update x using the step

        % Update Hessian using BFGS. Use equations (7.36), (7.73) and (7.74)
        % Compute y_k
         y_k = df(x+dx) + mu_new*dg(x+dx) - ...
```

```
            df(x) - mu_new*dg(x);

        % Compute theta

         if dx'*y_k' >= 0.2*dx'*W*dx
             theta = 1;
         else
             theta = 0.8*dx'*W*dx/(dx'*W*dx - dx'*y_k');
         end

        % Compute  dg_k using y_k, theta, W and dx
         dg_k = theta*y_k' + (1-theta)*W*dx;

        % Compute new Hessian using BFGS update formula
         W = W + ((dg_k*dg_k')/(dg_k'*dx)) - (((W*dx)*(W*dx)')/(dx'*W*dx));

         x = x + dx;
        % Update termination criterion:
         gnorm = norm(df(x) + mu_new*dg(x),2); % norm of Largangian gradient

        mu_old = mu_new;  % Update mu_old by setting it to mu_new

        % save current solution to solution.x
        solution.x = [solution.x, x];
    end
```