DAO MAE 598 HW5.

Table of Contents

.... 1
HOUSEKEEPING 1
Define our functions. 1
Optimization settings 1
SOLUTION. 2
Funciton Library 2

Template code provided by Professor Yi Ren.

HOUSEKEEPING

```
clc; clear;
close all; format long
```

Define our functions.

```
f = @(x) x(1)^2+(x(2)-3)^2;

df = @(x)[2*x(1) , 2*(x(2)-3)]; % ROW. Each new column is a new
 variable, each new row is a new function.

g = @(x)[x(2)^2-2*x(1); (x(2)-1)^2+5*x(1)-15];  % Equations for all
 gs!
dg = @(x)[-2 2*x(2); 5 2*(x(2)-1)] ;  % Equations for all dgs!
```

Optimization settings

```
opt.alg = 'myqp'; % 'myqp' or 'matlabqp'. NOTE opt is a STRUCTURE as
 well.


opt.linesearch = true; % false or true

% TOLERANCE
opt.eps = 1e-3;

% Initial Guess
x0 = [1;1];

% Feasibility check for the initial point.
if max(g(x0)>0)
    errordlg('Infeasible intial point! You need to start from a
 feasible one!');
```

```
    return
end
```

SOLUTION.

```
solution = mysqp(f, df, g, dg, x0, opt);
optimal_x = solution.x(:,end)
g = g(solution.x(:,end))
f= f(solution.x(:,end))
```

Note that answer printout appears at the end of this document.

Funciton Library

SQP

```
function solution = mysqp(f, df, g, dg, x0, opt)
    % STEP 1 -- Set initial conditions
    x = x0; % Set current solution to the initial guess
    % STORE x.
    solution = struct('x',[]); % Create an entry called 'x' w/ empty
 field
    solution.x = [solution.x, x]; % CONCATENATE. save current solution
 to solution.x

    % HESSIAN
    W = eye(numel(x));    % Initialize as I, with dimensions
 corresponding to x
    % LAGRANGE MULTIPLIERS
    mu_old = zeros(size(g(x)));    % Start with ZERO Lagrange
 multiplier estimates
    % Initialization of the weights in merit function
    w = zeros(size(g(x)));         % Start with zero weights

    % STEP 2 -- Calculate the termination criterion
    gnorm = norm(df(x) + mu_old'*dg(x)); % norm of Lagrangian
 gradient.

    % STEP 3 -- RUN THE LOOP
    while gnorm > opt.eps % if not terminated...

        % STEP 3.1 -- 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

        % STEP 3.2 -- Based on the linesearch opt.
        if opt.linesearch  % IF TRUE.
            [a, w] = lineSearch(f, df, g, dg, x, s, mu_old, w);
        else
            a = 0.1;
        end

        % STEP 3.3 -- Update the current solution using the step
        dx = a*s;                % Step for x
        x = x + dx;              % Update x using the step

        % STEP 3.4 -- BFGS UPDATE

            % Compute y_k
            y_k = [df(x) + mu_new'*dg(x) - df(x-dx) - mu_new'*dg(x-
dx)]';
            % 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
            dg_k = theta*y_k + (1-theta)*W*dx;
            % Compute new Hessian
            W = W + (dg_k*dg_k')/(dg_k'*dx) - ((W*dx)*(W*dx)')/
(dx'*W*dx);

        % STEP 3.5 -- Reevalute termination criterial.
        gnorm = norm(df(x) + mu_new'*dg(x)); % norm of Largangian
 gradient, with mu_new and new x guess.
        mu_old = mu_new;  % UPDATE Multipliers

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

% Armijo line search
function [a, w] = lineSearch(f, df, g, dg, x, s, mu_old, w_old)
    t = 0.1; % scale factor on current gradient: [0.01, 0.3]
    b = 0.8; % scale factor on backtracking: [0.1, 0.8]
    a = 1; % maximum step length

    D = s; % direction for x. This is the output of the QP.

    % Calculate WEIGHTS.
    w = max(abs(mu_old), 0.5*(w_old+abs(mu_old))); % Update these at
 each SQP iteration (they depend on updated mu)

    % terminate if line search takes too long.
```

```
    count = 0;
    while count<100
        % Calculate phi(alpha) using merit function in (7.76)
        phi_a = f(x + a*D) + w'*abs(min(0, -g(x+a*D)));

        % Caluclate psi(alpha) in the line search using phi(alpha)...
        phi0 = f(x) + w'*abs(min(0, -g(x)));        % phi(0)   ....
 f(x) + W^T * max(0, abs(g(x)))
        dphi0 = df(x)*D + w'*((dg(x)*D).*(g(x)>0)); % phi'(0)  . NOTE
 df(x) is a ROW, d is a COLUMN.
        % And dg(x)*D .* [0 1 0 0 1] --> you only keep the ones
        % corresponding to g_i >0 (the violations)
        psi_a = phi0 +  t*a*dphi0; % THE LINEAR FUNCTION OF a.
        % stop if condition satisfied
        if phi_a<psi_a
            break;
        else
            % backtracking
            a = a*b;
            count = count + 1;
        end
    end
end


% QP FUNCTION
function [s, mu0] = solveqp(x, W, df, g, dg)  % x is commonly columns.
    % Implement an Active-Set strategy to solve the QP problem given
 by
    % min     (1/2)*s'*W*s + c'*s
    % s.t.    A*s-b <= 0
    %
    % where As-b is the linearized active contraint set

    % Strategy should be as follows:
    % 1-) Start with empty working-set (A is empty)
    % 2-) Solve the problem using the working-set
    % 3-) Check the constraints and Lagrange multipliers (does the
 solution
    % satisfy constraints?)
    % 4-) If all constraints are staisfied and Lagrange multipliers
 are positive, terminate!
    % 5-) If some Lagrange multipliers are negative or zero, find the
 most negative one
    %     and remove it from the active set
    % 6-) If some constraints are violated, add the most violated one
 to the working set
    % 7-) Go to step 2


    % Compute c in the QP problem formulation
    c = [df(x)]';   % NOTE c is just f'.

    % Compute A in the QP problem formulation
```

```
   A0 = dg(x);     % Though we want an empty set, we'll clear these
out later.

   % Compute b in the QP problem formulation
   b0 = -g(x);     % Same here.

   % Initialize variables for active-set strategy
   stop = 0;           % Start with stop = 0
   % Start with empty working-set
   A = [];         % A for empty working-set
   b = [];         % b for empty working-set
   % Indices of the constraints in the working-set
   active = [];    % Indices for empty-working set

   while ~stop  % Continue until stop = 1
       % Initialize all mu as zero and update the mu in the working
set
       mu0 = zeros(size(g(x)));

       % Extact A corresponding to the working-set
       A = A0(active,:);  % We pull whichever are active.
       % Extract b corresponding to the working-set
       b = b0(active);

       % Solve the QP problem given A and b
       [s, mu] = solve_activeset(x, W, c, A, b); % Separate function
for QP solve.
       % Round mu to prevent numerical errors (Keep this)
       mu = round(mu*1e12)/1e12;  % Sometimes it is 0, but MATLAB's
precision messes up. C'mon!

       % Update mu values for the working-set using the solved mu
values
       mu0(active) = mu; % Now we add now-active mu's into the
original array

       % Calculate the constraint values using the solved s values
       gcheck = A0*s-b0;

       % Round constraint values to prevent numerical errors (Keep
this)
       gcheck = round(gcheck*1e12)/1e12;  % Again, to prevent errors.

       % Variable to check if all mu values make sense.
       mucheck = 0;        % Initially set to 0

       % Indices of the constraints to be added to the working set
       Iadd = [];              % Initialize as empty vector
       % Indices of the constraints to be added to the working set
       Iremove = [];           % Initialize as empty vector

       % Check mu values and set mucheck to 1 when they make sense
       if (numel(mu) == 0)
           % When there no mu values in the set
```

```
            mucheck = 1;         % OK
        elseif min(mu) > 0
            % When all mu values in the set positive
            mucheck = 1;         % OK
        else
            % When some of the mu are negative
            % Find the most negaitve mu and remove it from acitve set
            [~,Iremove] = min(mu);  % Use Iremove to remove the
 constraint
        end

        % Check if constraints are satisfied
        if max(gcheck) <= 0
            % If all constraints are satisfied
            if mucheck == 1
                % If all mu values are OK, terminate by setting stop =
 1
                stop = 1;
            end
        else
            % If some constraints are violated
            % Find the most violated one and add it to the working set
            [~,Iadd] = max(gcheck); % Use Iadd to add the constraint
        end
        % Remove the index Iremove from the working-set
        active = setdiff(active, active(Iremove));
        % Add the index Iadd to the working-set
        active = [active, Iadd];

        % Make sure there are no duplications in the working-set (Keep
 this)
        active = unique(active);
    end
end

% ACTIVE SET SOLVER.
function [s, mu] = solve_activeset(x, W, c, A, b)
    % Given an active set, solve QP

    % Create the linear set of equations given in equation (7.79)
    M = [W, A'; A, zeros(size(A,1))];
    U = [-c; b];
    sol = M\U;          % Solve for s and mu

    s = sol(1:numel(x));                % Extract s from the solution
    mu = sol(numel(x)+1:numel(sol));    % Extract mu from the solution

end
```

```
optimal_x =
   1.060416903353997
   1.456335638945722
g =
   0.000079686555450
```

```
  -9.489673267858013
f =
   3.507383668507993
```

Published with MATLAB® R2020a