ME598/494 Homework 5

Aditya Vipradas
ASU ID: 1209435588

April 11, 2016

SQP algorithm is implemented to solve the given optimization problem with two inequality constraints. The starting point is considered to be (1, 1). This algorithm is accompanied with the BFGS method to approximate the Hessian of the Lagrangian and the Armijo line search with the corresponding merit function. The active-set strategy is incorporated to solve the QP subproblem in each iteration. As observed from the output, the problem is solved in 4 iterations. Only the first inequality constraint is found to be active and the minimum is obtained at (1.0604, 1.4563). The SQP path and convergence plots are shown below followed by the MATLAB codes.

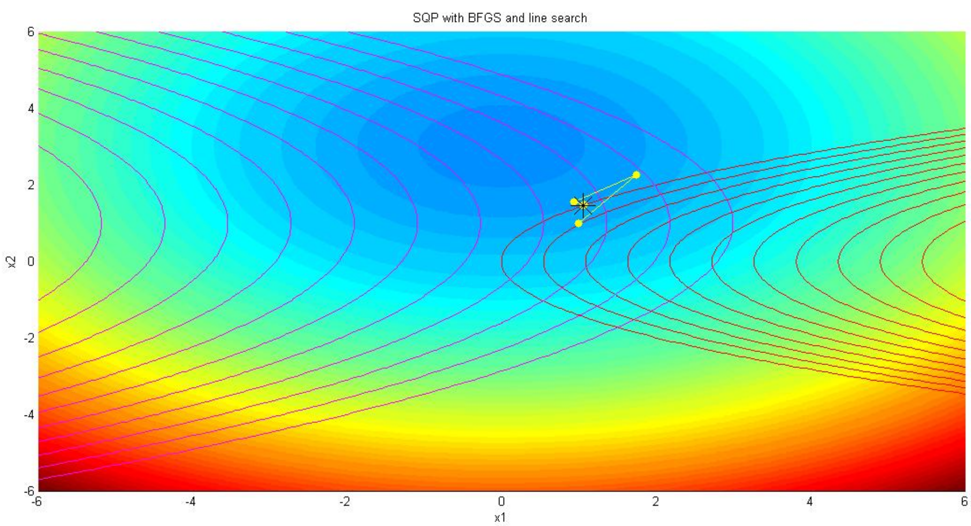

Figure 1: SQP (with BFGS and line search) path

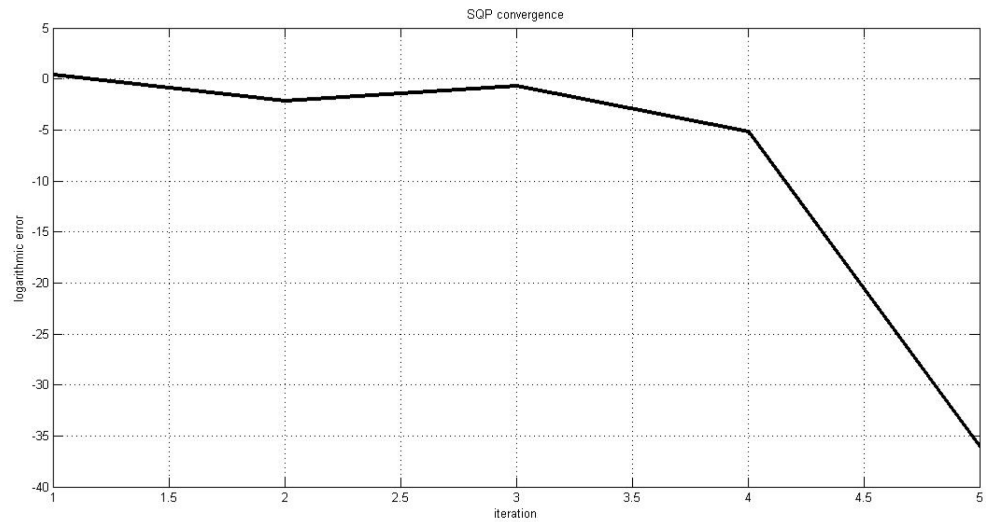

Figure 2: Function error plot

```matlab
%%%%%%%%%%%%%% 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
close all; % Close all windows
clc;

%% Optimization settings
% Here we specify the objective function by giving the function handle to a
% variable, for example:
f = @(x)objective(x); % replace with your objective function
% In the same way, we also provide the gradient of the
% objective:
df = @(x)objectiveg(x); % replace accordingly

g = @(x)constraint(x);
dg = @(x)constraintg(x);

% 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 QP solution algorithm
% When set to 'matlabqp' MATLAB's QP solver is used.
% When set to 'myqp' your own QP solver is used.

opt.alg = 'myqp'; % 'myqp' or 'matlabqp'

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

% Set the tolerance to be used as a termination criterion:
opt.eps = 1e-3;

% Set the initial guess: (column vector, i.e. x0 = [x1; x2] )
x0 = [1; 1];
disp(g(x0));
% 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
%% Run optimization
% Run your implementation of SQP algorithm. See mysqp.m
solution = mysqp(f, df, g, dg, x0, opt);

%% Report
report(solution,f,g);
```

```
%%%%%%%%%%%%%% 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
```

```
function [s, mu0] = solveqp(x, W, df, g, dg)
    % 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
    % 2-) Solve the problem using the working-set
    % 3-) Check the constraints and Lagrange multipliers
    % 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)';

    % Compute A in the QP problem formulation using all constraints
     A0 = dg(x);

    % Compute b in the QP problem formulation using all constraints
     b0 = -1*g(x);

    % 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 = [0, 0];
        % Extact A corresponding to the working-set from A0
         A = A0(active, :);
        % Extract b corresponding to the working-set from b0
         b = b0(active);

        % Solve the QP problem given A and b
        [s, mu] = solve_activeset(x, W, c, A, b)
        % Round mu to prevent numerical errors (Keep this)
        mu = round(mu*1e12)/1e12
        %mu = round(mu);
        % Update mu values for the working-set using the solved mu values
         mu0(active) = mu;

        % 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
%         gcheck = round(gcheck)
        % 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 active set

             Iremove = find(mu==min(mu))  % Use Iremove to remove the constraint
            % Remove the index Iremove from the working-set
             active(Iremove) = [];
        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 = find(gcheck == max(gcheck)) % Use Iadd to add the constraint
            % Add the index Iadd to the working-set
             active(end+1) = Iadd
        end
        % Make sure there are no duplications in the working-set (Keep this)
        active = unique(active)
    end
end

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)
    row = size(A);
    row = row(1);
     M = [W, A'; A, zeros(row)];
     U = [-1*c; b];
```

```
    sol = M\U;          % Solve for s and mu

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

    % Calculate weights in the merit function using eaution (7.77)
     w = max(abs(mu_old), 0.5*(w_old + abs(mu_old)));
    % 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,-1*g(x+a*D)));

        % Caluclate psi(alpha) in the line search using phi(alpha)
         phi0 = f(x) + w*abs(min(0,-1*g(x)));% phi(0)
         dphi0 = df(x) + w*abs(min(0,-1*dg(x)));% phi'(0)
         psi_a = phi0 + t*a*dphi0;% psi(alpha) = phi(0)+t*alpha*phi'(0)

        % stop if condition satisfied
        if phi_a < psi_a
            stop = 1
            if stop;
                break;
            else
                % backtracking
                a = a*b;
                count = count + 1;
            end
        end
    end
end
```

```
function f = objective(x)
    %%% Calculate the objective function f(x)


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

```
function df = objectiveg(x)
    %%% Calculate the gradient of the objective (row vector)
    %%% df(x)/dx = [df/dx1, df/dx2, ..., df/xn]

    df = [2*x(1), 2*(x(2) - 3)];

end
```

```
function g = constraint(x)
    %%% Calculate the constraints (column vector)
    %%% g(x) = [g1(x); g2(x); ... ; gm(x)]

    g = [x(2)^2 - 2*x(1); (x(2) - 1)^2 + 5*x(1) - 15];
end
```

```
function dg = constraintg(x)
    %%% Calculate the gradient of the constraints
    %%% dg(x)/dx = [dg1/dx1, dg1/dx2, ... , dg1/dxn;
    %%%             dg2/dx1, dg2/dx2, ... , dg2/dxn;
    %%%             ...
    %%%             dgm/dx1, dgm/dx2, ... , dgm/dxn]


    dg = [-2, 2*x(2); 5, 2*(x(2)-1)];

end
```