MAE 598: Design Optimization
HW5 (SQP Program)

Table of Contents

Aishwarya Ledalla

Optional overhead

```
clear; close all; clc;
```

Optimization settings

```
f = @(x) x(1)^2 + (x(2) - 3)^2; % objective function
df = @(x) [2*x(1) 2*(x(2) - 3)]; % gradient of objective

g = @(x) [x(2)^2 - 2*x(1); (x(2) - 1)^2 + 5*x(1) - 15]; % constraint
dg = @(x) [-2, 2*x(2); 5, 2*(x(2) - 1)]; % gradient of contraint
```

Specify algorithm

```
opt.alg = 'myqp';

% 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:
x0 = [1;1];

% Feasibility check for the initial point.
if max(g(x0)>0)
    errordlg('Mission Failure!! Try again with feasible intial point!');
    return
end
```

Run optimization

Run your implementation of SQP algorithm. See mysqp.m

```
solution = mysqp(f, df, g, dg, x0, opt);
```

Report

```
x = solution.x; i = 1:length(x); Optimal = [i;x(1,:);x(2,:)];
for j = 1:length(x)
    f_o(j,:) = f(x(:,j));
    g_o(j,:) = g(x(:,j));
end

table(Optimal,'RowNames',{'iteration','X1','X2'})
sprintf('f(x) = %0.3f',f_o(5))
sprintf('g1(x) = %0.3f <= 0',g_o(5,1))
sprintf('g2(x) = %0.3f <= 0',g_o(5,2))
```

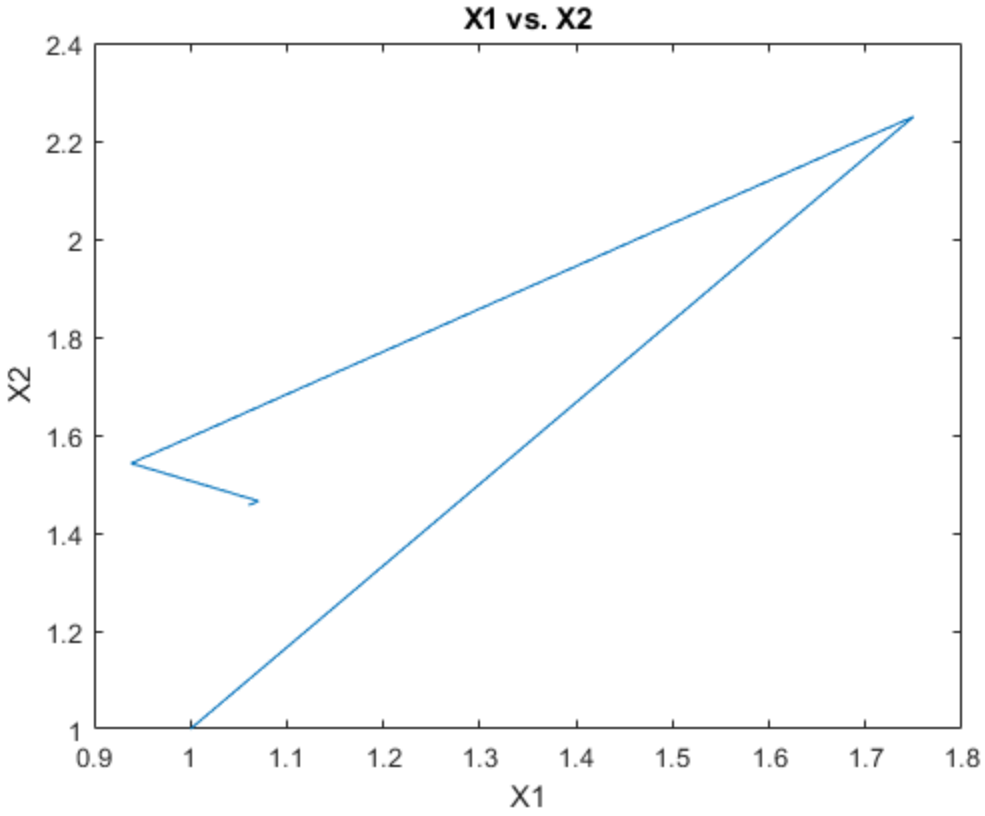
```
ans =

  3×1 table
```

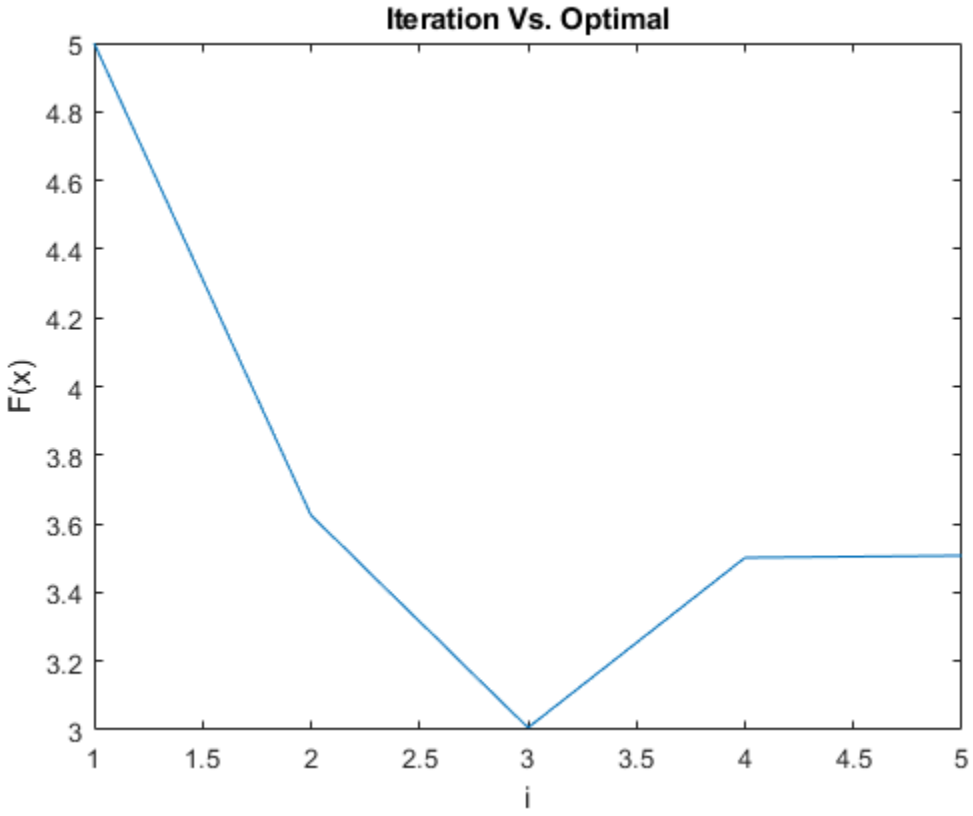
	Optimal				
iteration	1	2	3	4	5
X1	1	1.75	0.93852	1.0705	1.0604
X2	1	2.25	1.5421	1.4653	1.4563

```
ans =

    'f(x) = 3.507'


ans =

    'g1(x) = 0.000 <= 0'


ans =

    'g2(x) = -9.490 <= 0'
```

Plots:

```
figure(1)
plot(x(1,:),x(2,:)); title('X1 vs. X2'); xlabel('X1'); ylabel('X2');
figure(2)
plot(i,f_o); title('Iteration Vs. Optimal'); xlabel('i'); ylabel('F(x)');
```


X1 vs. X2
X2
X1
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.9
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8


Iteration Vs. Optimal
F(x)
i
5
4.8
4.6
4.4
4.2
4
3.8
3.6
3.4
3.2
3
1
1.5
2
2.5
3
3.5
4
4.5
5

Sequential Quadratic Programming Implementation with BFGS

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(numel(x));            % Start with an identity Hessian matrix
% Initialization of the Lagrange multipliers
mu_old = zeros(size(g(x)));   % Start w/ 0 LagrangemultiplierEstimates
% Initialization of the weights in merit function
w = zeros(size(g(x)));        % Start with 0 weights

% Set the termination criterion
gnorm = norm(df(x) + mu_old'*dg(x)); % 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
        [s, mu_new] = solveqp(x, W, df, g, dg);
    else
        % Solve 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 = 0.1;
    end

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

    % Update Hessian using BFGS. Use eqs (7.36), (7.73) and (7.74)
    % 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);

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

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

Line Search on the Merit Function

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, -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)
        dphi0 = df(x)*D + w'*((dg(x)*D).*(g(x)>0)); % phi'(0)
        psi_a = phi0 +  t*a*dphi0;                  % psi(alpha)
        % stop if condition satisfied
        if phi_a<psi_a;
            break;
        else
            % backtracking
            a = a*b;
            count = count + 1;
        end
    end
    end
```

QP Subproblem using Active Set Strategy

```
function [s, mu0] = solveqp(x, W, df, g, dg)
% Compute c in the QP problem formulation
c = [df(x)]';

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

% Compute b in the QP problem formulation
b0 = -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 = zeros(size(g(x)));

    % Extact A corresponding to the working-set
    A = A0(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);
    % Round mu to prevent numerical errors (Keep this)
    mu = round(mu*1e12)/1e12;

    % 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;

    % 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
        active = unique(active);
    end
    end
```

Active Set

```
    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

end
```

Published with MATLAB® R2021b