Final Submission-HW_5

Contents


```
%HOME_WORK_5 BY K PADMA RAO
%ID 1223239758
%%%%%%%%%%%%%% Main Entrance %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% By Max Yi Ren and Emrah Bayrak %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Optional overhead

```
clear; % Clear the workspace
close all; % Close all windows
```

Optimization settings

Here we specify the objective function by giving the function handle to a variable

```
%for example:
f = @(x) x(1)^2+ (x(2)-3)^2;                      % Given objective function
                                                  % In the same way, we also provide the gradient of the objective:
df =@(x) [2*x(1), 2*x(2)-6];                      % obtained gradient of the objective
g =@(x) [x(2)^2-2*x(1); (x(2)-1)^2+5*x(1)-15];    %given contraints
dg =@(x) [-2 2*x(2); 5 2*x(2)-2 ];               %gradient for the given contraints


% 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.

                                                  %Specifying the algorithm
opt.alg = 'myqp';                                 %Used'myqp'

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

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

x0 = [1;1];                                       % Initial guessed vales :

                                                  % Feasibility checking 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

Code to Run the implementation of SQP algorithm.

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

Report

```
                                                  %report(solution(),f,g);
for i = 1:length(solution.x)                      %defining the length
    sol(i) = f(solution.x(:, i));                 %Storing the solution value
    con = g(solution.x(:, i));                    %Storing the gradient values of x1 and x2
    constraint_one(i) = con(1);                   %Storing the g1 constraint as contraint_one
    constraint_two(i) = con(2);                   %Storing the g2 constraint as contraint_two
end

count = 1:length(solution.x);                     % Iteratiing of each x1 and x2
tiledlayout('flow')

nexttile                                          %Tile 1
plot(count, sol,'b','LineWidth',2)
grid on
title('f(x1, x2) vs. Iterations')
xlabel('Iterations')
ylabel('f(x1, x2)')

nexttile                                          %Tile 2
hold on
plot(count, sol,'b','LineWidth',2)
plot(count, constraint_one,'LineWidth',1.5)
plot(count, constraint_two,'LineWidth',1.5)
grid on
legend('f(x) value', 'con_one(x)', 'con_two(x)')
```

```
title('Objective fnct& Constraint Functions vs. Iterations')
xlabel('Iterations')
ylabel('Objective fnct & Constraint Functions')
hold off

nexttile                                                    %Tile 3
plot(solution.x(1, :), solution.x(2, :),'b','LineWidth',2)
grid on
title('Values of x2 vs. x1')
xlabel('x1')
ylabel('x2')

disp("x1 & x2 are as follows  = ");                     %Resulting the values of x1 and x2
disp(solution.x(:, end));                               %Optimumed values of x1 and x2

disp("F(x1, x2) = ");                                   %Resulting the objective funtion values at x1 and x2
disp(sol(end));                                         %Objective function at x1 and x2

disp("con_one(x1, x2) = ");                                  %Resulting constraint_one at x1 and x2
disp(constraint_one(end));                              %Constraint_one function at x1 and x2

disp("con_two(x1, x2) = ");                                  %Resulting constraint_two at x1 and x2
disp(constraint_two(end));                              %Constraint_two function at x1 and x2

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

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

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

                                                        %Search process
    solution = struct('x',[]);
    solution.x = [solution.x, x];                       %Storing the Current solution to solution.x

                                                        %Initializing the Hessian matrix
    W = eye(numel(x));                                  %Taking an identity Hessian matrix
                                                        %Initializing the Lagrange multipliers
    mu_old = zeros(size(g(x)));                         %Starting with the  zero Lagrange multiplier estimates
                                                        %Initializing the weights in merit function
    w = zeros(size(g(x)));                              %Starting with the zero weights

                                                        %Setting the termination criterion
    gnorm = norm(df(x) + mu_old'*dg(x));                %Norm of the Largangian gradient

    while gnorm>opt.eps                                 %if not terminated

                                                        %Implementing the QP problem and solving it
        if strcmp(opt.alg, 'myqp')
                                                        %Solving the QP
                                                        %Subproblem to find s and mu

            [s, mu_new] = solveqp(x, W, df, g, dg);
        else
                                                        %Solvin 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), [], [], [], [], x, qpalg);
            mu_new = lambda.ineqlin;
        end




        if opt.linesearch                               %Here, opt.linesearch switches the line search on or off.
            [a, w] = lineSearch(f, df, g, dg, x, s, mu_old, w);
        else
            a = 0.1;                                    %Setting the variable "a" to constant value 0.1 and checking the affect of convergence
        end

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

                                                        %Updating Hessian using BFGS(HINT: Use equations (7.36), (7.73) and (7.74) to Compute y_k)
        y_k = [df(x) + mu_new'*dg(x) - df(x-dx) - mu_new'*dg(x-dx)]';
                                                        %Computing the theta value
        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
                                                        %Computing  dg_k
        dg_k = theta*y_k + (1-theta)*W*dx;
                                                        %Here,Computing the new Hessian
        W = W + (dg_k*dg_k')/(dg_k'*dx) - ((W*dx)*(W*dx)')/(dx'*W*dx);

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

      solution.x = [solution.x, x];                     %Storing the current solution to solution.x
```

```
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The following code performs 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;                                    %'t' is the scale factor on current gradient: [0.01, 0.3]
    b = 0.8;                                    %'b' is the scale factor on backtracking: [0.1, 0.8]
    a = 1;                                      %Assigning the maximum step length as 1

    D = s;                                      %Putting direction for x

                                                %Calculating the weights in the merit function using eqaution (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
                                                %Calculating the phi(alpha) using merit function in (7.76)
        phi_a = f(x + a*D) + w'*abs(min(0, -g(x+a*D)));

                                                %Caluclate the 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


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%The following code solves the QP subproblem using active set strategy

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [s, mu0] = solveqp(x, W, df, g, dg)
                                                %Steps to follow
                                                %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 act
                                                % 6-) If some constraints are violated, add the most violated one to the working set
                                                % 7-) Go to step 2


                                                %Computing c in the QP problem formulation
    c = [df(x)]';

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

                                                %Computing b in the QP problem formulation
    b0 = -g(x);

                                                %Initializing variables for active-set strategy
    stop = 0;                                   %Starting this,by with stop = 0
                                                %Starting with the 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                                 %here we Continue until stop = 1
                                                %Initializing all mu as zero and update the mu in the working set
        mu0 = zeros(size(g(x)));

                                                %Extacting A corresponding to the working-set
        A = A0(active,:);
                                                %Extracting b corresponding to the working-set
        b = b0(active);

                                                %Solving the QP problem given A and b
```

```
        [s, mu] = solve_activeset(x, W, c, A, b);
                                                     %Rounding the values of mu so that we can prevent numerical errors
        mu = round(mu*1e12)/1e12;

                                                     %Updating mu values for the working-set using the solved mu values
        mu0(active) = mu;

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

                                                     %Rounding constraint values so that we can prevent numerical errors
        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 = [];                                   %Initializing as empty vector
                                                     %Indices of the constraints to be added to the working set
        Iremove = [];                                %Initializing as empty vector

                                                     %Checking mu values and setting mucheck to 1 when they make sense
        if (numel(mu) == 0)
                                                     %When there no mu values in the set
            mucheck = 1;                             %then OK
        elseif min(mu) > 0
                                                     %When all mu values in the set positive
            mucheck = 1;                             %then OK
        else
                                                     %When some of the mu are negative
                                                     %Finding the most negaitve mu and remove it from acitve set
            [~,Iremove] = min(mu);                   %Using Iremove to remove the constraint
        end


                                                     %Checking that the
                                                     %constraints are satisfied or not
        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);                  %Using Iadd to add the constraint
        end
                                                     %Removing the index Iremove from the working-set
        active = setdiff(active, active(Iremove));
                                                     %Adding 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

function [s, mu] = solve_activeset(x, W, c, A, b)
                                                     %Given an active set, and by solving QP

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

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

end
```

```
x1 & x2 are as follows  =
    1.0604
    1.4563

F(x1, x2) =
    3.5074

con_one(x1, x2) =
   7.9687e-05

con_two(x1, x2) =
   -9.4897
```

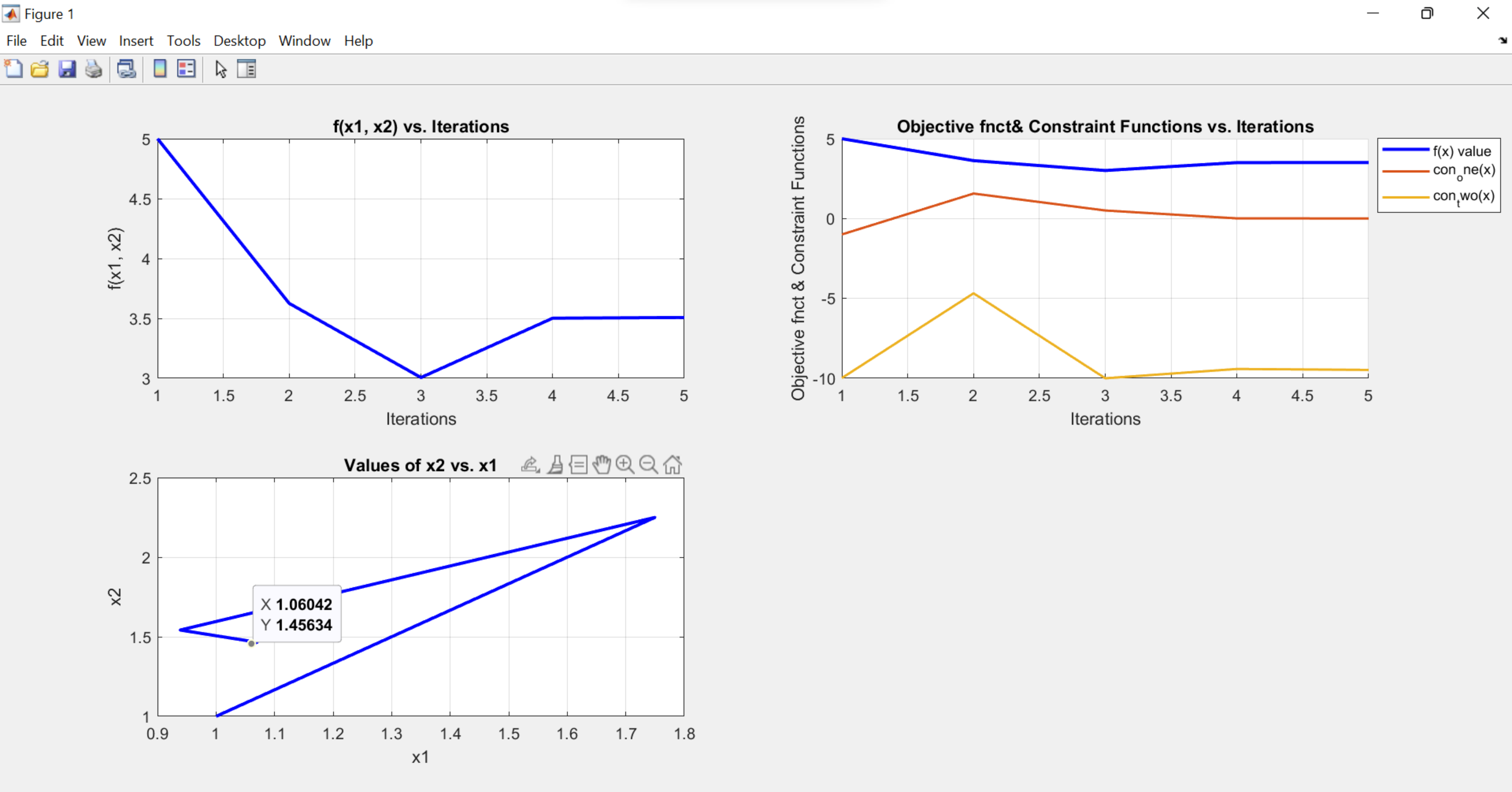
Figure 1
File Edit View Insert Tools Desktop Window Help
f(x1, x2) vs. Iterations
f(x1, x2)
Iterations
Objective fnct& Constraint Functions vs. Iterations
Objective fnct & Constraint Functions
Iterations
f(x) value
con_one(x)
con_two(x)
Values of x2 vs. x1
x2
x1
X 1.06042
Y 1.45634