MATLAB CODE FOR IPM METHOD

```
%%Note:
%% The algorithm implemented below is the IPM method. We have used BFGS
update instead of Hessain of Lagrangian and also used
%% merit function and second order correction to ensure global convergence.

%% Section -1 : Defining variables and constraints

f =  @(x1,x2) 0.01*x1^2 + x2^2 - 100;
g1 = @(x1,x2) x2 -10*x1 +10;
g2 = @(x1,x2) 2 - x1;
g3 = @(x1,x2) x1 - 50;
g4 = @(x1,x2) -50 - x2;
g5 = @(x1,x2) x2 - 50;
h1 = @(x1,x2) 0;
h2 = @(x1,x2) 0;
h3 = @(x1,x2) 0;
h4 = @(x1,x2) 0;
h5 = @(x1,x2) 0;

s1 = 19; s2 = 1; s3 = 47; s4 = 51; s5 = 49;

num_iconstraints = 5; %Defining the number of Inequality constraints
num_econstraint = 0; %Defining the number of Equality Constraints
num_variables = 2; %Defining the number of decision variables

s = zeros(num_iconstraints,1); %Vector of slack variables

    s(1,1) = 19;
    s(2,1) = 1;
    s(3,1) = 47;
    s(4,1) = 51;
    s(5,1) = 49;


S = diag(s);%Matrix of Slack Variables


u = 20; % Perturbation Factor

z = zeros(num_iconstraints,1);
for i = 1:num_iconstraints
    z(i,1) = u/s(i,1); %Initializing the values of Lagrange Multipliers
end

Z = diag(z); %Matrix of Lagrange Multipliers

equality = 0; %Indicator variable for presence of equality constraints
```

```
inequality = 1; %Indicator variable for presence of Inequality constraints
```

%% Section 2: Gradients Hessians and Lagrangians

```
x1 = 3; x2 = 2;
x = [x1;x2];
[g_c1,g_c2,g_c3,g_c4,g_c5] = grad_constraints(x); %Function
grad_constraints() calculates gradients of all 5 constraints

Ai = [g_c1 g_c2 g_c3 g_c4 g_c5]; % Vector of Gradient of Inequality
Constraints

L = @(x1,x2) f +
z1*(g1(x1,x2)+s1)+z2*(g2(x1,x2)+s2)+z3*(g3(x1,x2)+s3)+z4*(g4(x1,x2)+s4)+z5*(g
5(x1,x2)+s5); %Lagrangian

L_gradient = @(x1,x2)(x1/50);
L_gradient1 = @(x1,x2)(2*x2);

v = 0.2;
merit_func = @(x1,x2,s1,s2,s3,s4,s5,u) f(x1,x2) - u*
(log(s1)+log(s2)+log(s3)+log(s4)+log(s5)) + v*norm([g1(x1,x2)+s1 g2(x1,x2)+s2
g3(x1,x2)+s3 g4(x1,x2)+s4 g5(x1,x2)+s5],1); %Merit Function
```

%% Section 3: Creating matrices A and b to solve linear system of equations and the entire IPM algorithm with Second Order Correction

```
x1 = 3; x2 = 2; %Initial Points for the algorithm
Ai_new = zeros(num_iconstraints,num_iconstraints);
Ai_new(1:num_variables,1:num_iconstraints) = Ai;

t = 0.995;

Z1 = diag(Z);

B_knew = eye(num_variables); %Initial BFGS Update as Identity Matrix

B1 = [L_gradient(x1,x2);L_gradient1(x1,x2)];
B2 = [z - u* (S\ones(5,1))];
B3 =
[(g1(x1,x2)+s(1));(g2(x1,x2)+s(2));(g3(x1,x2)+s(3));(g4(x1,x2)+s(4));(g5(x1,x
2)+s(5))];

warning('off','all');
datasave =[];
```

```
iter = 0;
fprintf('  Iteration       X1         X2   F(X1,X2)     Error\n');
datasave = [0   x1    x2    f(x1,x2) max([norm(B1),norm(B2),norm(B3)])];

while max([norm(B1),norm(B2),norm(B3)])> 0.041 %Stopping Criteria

    if(equality == 0 && inequality == 1) %Case where there are only
inequality constraints
    alpha_d =0.01;

    %The below steps are for creating the matrix for the primal dual
    %method. The Jacobian is represented by the matrix A and the right hand
    %side by B. The vector d contains directions 4 components dx, ds, dy
    %and dz corresponding to primal variables, slack variables and the
    %corresponding lagrange multipliers.

    M1 = [B_knew zeros(num_variables,num_iconstraints)  Ai];
    M2 = [zeros(num_iconstraints,num_variables) inv(S)*Z
eye(num_iconstraints)];
    M3 = [Ai' eye(num_iconstraints)
zeros(num_iconstraints,num_iconstraints)];
    A = [M1;M2;M3];
    B = -[[L_gradient(x1,x2);L_gradient1(x1,x2)] ; (z - u*
(S\ones(num_iconstraints,1))) ;
[(g1(x1,x2)+s(1));(g2(x1,x2)+s(2));(g3(x1,x2)+s(3));(g4(x1,x2)+s(4));(g5(x1,x
2)+s(5))]];
    B1 = [L_gradient(x1,x2);L_gradient1(x1,x2)];
    B2 = [z - u* (S\ones(5,1))];
    B3 =
[(g1(x1,x2)+s(1));(g2(x1,x2)+s(2));(g3(x1,x2)+s(3));(g4(x1,x2)+s(4));(g5(x1,x
2)+s(5))];
    b = double(B);
    d = linsolve(A,B); % Solving Linear System of Equations to get search
directions
    dX = d(1:num_variables);
    ds = d(num_variables+1:num_variables+num_iconstraints);
    dz =
d(num_variables+num_iconstraints+1:num_variables+num_iconstraints*2);

    elseif equality == 1 && inequality == 0 %Case with only equality
constraints
    alpha_d =0.01;
    M1 = [B_knew zeros(num_variables,num_econstraints)  Ae];
    M2 = [zeros(num_econstraints,num_variables) inv(S)*Z
zeros(num_econstraints,num_econstraints)];
    M3 = [Ae' eye(num_econstraints) zeros(num_econstraints,constraints)];
    A = [M1;M2;M3];
    B = -[[L_gradient(x1,x2);L_gradient1(x1,x2)] ; (z - u*
(S\ones(num_econstraints,1))) ;
[(h1(x1,x2));(h2(x1,x2));(h3(x1,x2));(h4(x1,x2));(h5(x1,x2))]];
    b = double(B);
    d = linsolve(A,B);
    dX = d(1:num_variables);
    dy = d(num_variables+1:num_variables+num_econstraints);
    dz =
d(num_variables+num_econstraints+1:num_variables+num_econstraints*2);
```

```
    else %General case for both inequality and equality constraints
    alpha_d =0.01;
    M1 = [B_knew zeros(num_variables,num_constraints)  Ae  Ai];
    M2 = [zeros(num_constraints,num_variables) inv(S)*Z  zeros(5,5)
eye(num_iconstraints)];
    M3 = [Ae' zeros(num_iconstraints,num_iconstraints)
zeros(num_econstraints,num_econstraints)
zeros(num_econstraints,num_iconstraints)];
    M4 = [Ai' eye(num_constraints) zeros(num_econstraints,num_econstraints)
zero(num_iconstraints,num_iconstraints)];
    A = [M1;M2;M3;M4];
    B = -[[L_gradient(x1,x2);L_gradient1(x1,x2)] ; (z - u*
(S\ones(num_iconstraints,1))) ;
[(h1(x1,x2));(h2(x1,x2));(h3(x1,x2));(h4(x1,x2));(h5(x1,x2))]
;[(g1(x1,x2)+s(1));(g2(x1,x2)+s(2));(g3(x1,x2)+s(3));(g4(x1,x2)+s(4));(g5(x1,
x2)+s(5))]];
    b = double(B);
    d = linsolve(A,B);
    dX = d(1:num_variables);
    ds = d(num_variables+1:num_variables+num_iconstraints);
    dy =
d(num_variables+num_iconstraints:num_variables+num_iconstraints+num_econstrai
nts);
    dz =
d(num_variables+num_iconstraints+num_econstraints:num_variables+num_iconstrai
nts+num_econstraints+num_iconstraints);

    end

    x1_old = x1 ; %Storing old values for BFGS Update
    x2_old = x2 ; %Storing old values for BFGS Update
    dw = [dX;ds];

    % The below section is for updating the step size of dual variable
    beta_d = alpha_d;
    for k = 1:99

        beta_d = beta_d + 0.01;
        if (z + beta_d*dz >= (1-t)*z)

            alpha_d = beta_d;

        else
            break;
        end
    end

    S1_old = S(1,1);
    S2_old = S(2,2);
    S3_old = S(3,3);
    S4_old = S(4,4);
    S5_old = S(5,5);
```

```
    s1 = s(1);
    s2 = s(2);
    s3 = s(3);
    s4 = s(4);
    s5 = s(5);

    if equality==0 && inequality == 1 %Case where there are only inequality
constraints

    %Directional Derivative of Merit Function

    direc_merit_func = @(x1,x2,s1,s2,s3,s4,s5,u)
grad_merit(x1,x2,s1,s2,s3,s4,s5,u)' * dw +
v*norm([g1(x1,x2)+s1,g2(x1,x2)+s2,g3(x1,x2)+s3,g4(x1,x2)+s4,g5(x1,x2)+s5],1);
    elseif equality == 1 && inequality == 0 %Case with equality constraints
    direc_merit_func = @(x1,x2,s1,s2,s3,s4,s5,u)
grad_merit(x1,x2,s1,s2,s3,s4,s5,u)' * dw +
v*norm([h1(x1,x2),h2(x1,x2),h3(x1,x2),h4(x1,x2),h5(x1,x2)],1);
    else %General Case
    direc_merit_func = @(x1,x2,s1,s2,s3,s4,s5,u)
grad_merit(x1,x2,s1,s2,s3,s4,s5,u)' * dw +
v*norm([g1(x1,x2)+s1,g2(x1,x2)+s2,g3(x1,x2)+s3,g4(x1,x2)+s4,g5(x1,x2)+s5],1)+
v*norm(h1(x1,x2),h2(x1,x2),h3(x1,x2),h4(x1,x2),h5(x1,x2),1);
    end

    n = 0.4;
    newpoint = 0;
    tau1 = 0.5;
    tau2 = 0.8;
    tau = 0.01;
    alpha_p2 = 1;

    %We are performing the second order correction below to take into
    %Maretos effect caused by merit function. We ensure global convergence
    %as well as reasonable step size. We use second order order
    %correctionto determine the step size for primal variables.

    while newpoint == 0
        merit_reduction_new =
merit_func((x1+alpha_p2*dX(1)),(x2+alpha_p2*dX(2)),s1+alpha_p2*ds(1),s2+alpha
_p2*ds(2),s3+alpha_p2*ds(3),s4+alpha_p2*ds(4),s5+alpha_p2*ds(5),u);
        if merit_reduction_new <= merit_func(x1,x2,s1,s2,s3,s4,s5,u)+
n*alpha_p2*direc_merit_func(x1,x2,s1,s2,s3,s4,s5,u)
            x1 = x1 + alpha_p2*dX(1);
            x2 = x2 + alpha_p2*dX(2);
            newpoint = 1;
        elseif alpha_p2 == 1

            %Calculating new search direction by second order correction
            dk_new = -
Ai*pinv(Ai'*Ai)*[g1(x1+dX(1),x2+dX(2))+s1;g2(x1+dX(1),x2+dX(2))+s2;g3(x1+dX(1
),x2+dX(2))+s3;g4(x1+dX(1),x2+dX(2))+s4;g5(x1+dX(1),x2+dX(2))+s5];
            merit_reduction_new =
merit_func((x1+alpha_p2*dX(1)+dk_new(1)),(x2+alpha_p2*dX(2)+dk_new(2)),s1,s2,
s3,s4,s5,u);
```

```
            if merit_reduction_new <= merit_func(x1,x2,s1,s2,s3,s4,s5,u)+
n*alpha_p2*direc_merit_func(x1,x2,s1,s2,s3,s4,s5,u)
                %Updating the values of primal decision variables x
                x1 = x1 + dX(1)+dk_new(1);
                x2 = x2 + dX(2)+dk_new(2);
                newpoint = 1;
            else
                alpha_p2 = 0.99*alpha_p2;

            end
        else
                alpha_p2 = 0.99*alpha_p2;

        end
    end
    alpha_p = alpha_p2;

    %Updating parameter s and z
    s = s + alpha_p*ds;
    z = z + alpha_d*dz;

    %Creating Matrices of values of Lagrange Multipliers to be used  in
    %Jacobian A above.

    S = diag(s);
    Z = diag(z);

    %BFGS Update instead of using Hessian of Lagrangian
    f1 = x1-x1_old;
    f2 = x2-x2_old;

    f_k = [f1;f2];

    y_new = L_gradient(x1,x2);
    y_old = L_gradient(x1_old,x2_old);

    y_new1 = L_gradient1(x1,x2);
    y_old1 = L_gradient1(x1_old,x2_old);

    y_k = [double(y_new - y_old);double(y_new1-y_old1)];

    B_k = A(1:2,1:2);

    %We are using Damped BFGS update below

    if f_k'*y_k >= 0.2* (f_k' * B_k * f_k)
        theta_k = ones(1,2);
    else
        theta_k = (0.8*f_k'*y_k)/(f_k' * B_k * f_k)-(f_k'*y_k);
    end
    r_k = theta_k'*f_k' + (ones(1,2)-theta_k)' *(B_k*f_k)';
    B_knew = B_k - ((B_k * f_k * f_k' * B_k)/(f_k' * B_k *f_k))+
bsxfun(@rdivide ,(r_k * r_k'),(f_k'* r_k'));
```

```
    %Updating the perturbation factor

    u = 0.2 *(s'* z)/5;
    A(1:2,1:2) = B_knew;
    iter = round(iter+1);
    datasave = [datasave;   round(iter)    x1      x2      f(x1,x2)
max([norm(B1),norm(B2),norm(B3)])];

end

disp(datasave);
```

%% Using fmincon to evaluate problem

```
fun = @(x)0.01*x(1)^2 + x(2)^2 - 100;
x0 = [-1,-1];
A = [-10,1];
lb = [2,-50];
ub = [50,50];
b = 10;
Aeq =[];
beq =[];
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub);
```