PRACTICE 3

NON-LINEAR OPTIMIZATION WITH CONSTRAINTS

Andrés Herencia and Antonio Fernández

MUTECI 2023-2024

Table of Contents

Exercise 1

For the following nonlinear problem

$$\begin{aligned} \min_{(x_1,x_2)\in\mathbb{R}^2} \quad & \{2x_1 - x_2\} \\ \text{such to} \quad & -x_1^2 + x_2 \leq 0 \\ & (x_1-1)^2 + x_2 - 5 \leq 0 \\ & x_2 \geq 0 \end{aligned}$$

It is requested

(a) Determine the KKT points.

(b) Identify the optimum from the previous points.

(c) Use MATLAB (or any other software) to identify the optimum starting from various points, for example, (-0.5, -0.5); (0.5, 0.5); (2, 1) and verify if the same solution is reached.

(d) Perform two iterations with the Zoutendijk algorithm starting from the point (2, 1).

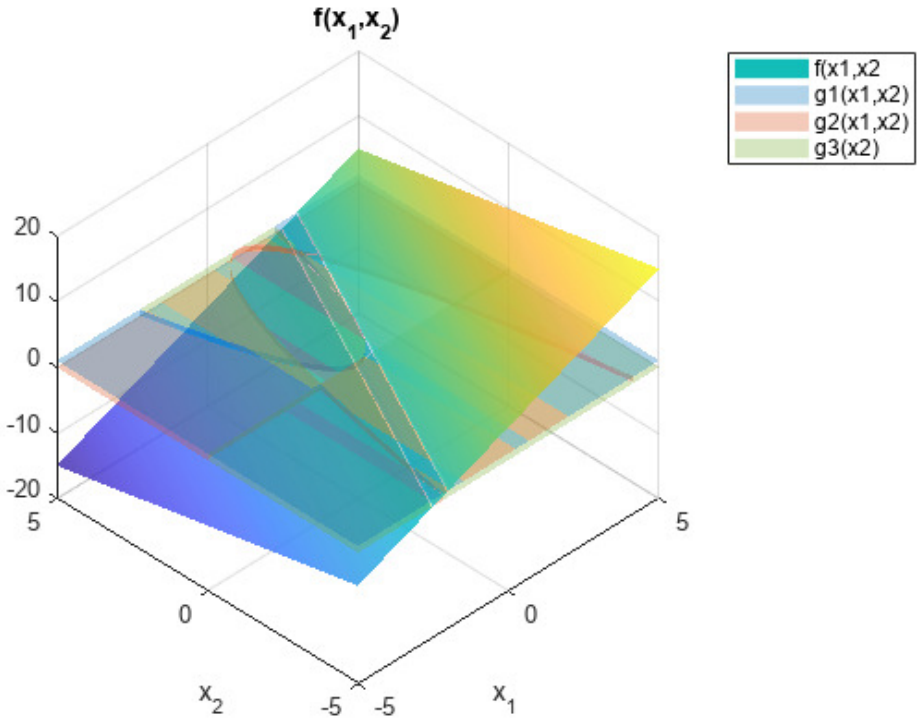
Solution

Before starting to compute, we can represent the function to make some interpretations about the solution.

```
f = @(x1,x2) 2.*x1 - x2;
g1 = @(x1,x2) -x1.^2 + x2 <=0;
g2 = @(x1,x2) x1.^2 - 2.*x1 + 1 + x2 - 5 <=0;
g3 = @(x1,x2) -x2 <= 0;

[Xgrid,Ygrid]=meshgrid(-5:0.01:5,-5:0.01:5);
z = f(Xgrid,Ygrid); z1 = g1(Xgrid,Ygrid); z2 = g2(Xgrid,Ygrid);
z3 = g3(Xgrid,Ygrid);

fig1 = figure(1);
surface(Xgrid,Ygrid,z,'EdgeColor','none','FaceAlpha',0.8)
hold on
s1 = surface(Xgrid,Ygrid,z1,'FaceColor','[0 0.4470 0.7410]', ...
    'FaceAlpha',0.3, 'EdgeColor','none');
s2 = surface(Xgrid,Ygrid,z2,'FaceColor','[0.8500 0.3250 0.0980]', ...
    'FaceAlpha',0.3, 'EdgeColor','none');
s3 = surface(Xgrid,Ygrid,z3,'FaceColor','[0.4660 0.6740 0.1880]', ...
    'FaceAlpha',0.3, 'EdgeColor','none');
hold off
grid on
title('f(x_1,x_2)')
xlabel('x_1')
ylabel('x_2')
view([-45 -45])
legend('f(x1,x2','g1(x1,x2)','g2(x1,x2)','g3(x2)')
view([-45 45])
```



a) Determine the KKT points.

We will compute te KKT points.

```
syms x1 x2 u1 u2 u3;

obj = 2.*x1 -x2;
ineq1 = -x1.^2 + x2;
ineq2 = (x1 - 1).^2 + x2 - 5;
ineq3 = -x2;

L = obj + u1*ineq1 +u2*ineq2 +u3*ineq3;
dL_dx1 = diff(L, x1);
dL_dx2 = diff(L, x2);

eq1 = dL_dx1 == 0;
eq2 = dL_dx2 == 0;
eq3 = u1*ineq1 == 0;
eq4 = u2*ineq2 == 0;
eq5 = u3*ineq3 == 0;

sol = solve([eq1, eq2, eq3, eq4, eq5], [x1, x2, u1, u2, u3]);

KKTpoints = [];

for i = 1:length(sol.x1)

    x1_sol = sol.x1(i);
    x2_sol = sol.x2(i);
```

```
    u1_sol = sol.u1(i);
    u2_sol = sol.u2(i);
    u3_sol = sol.u3(i);

    if subs(ineq1, [x1, x2], [x1_sol,x2_sol])<=0 && ...
        subs(ineq2, [x1, x2], [x1_sol,x2_sol])<=0 && ...
        subs(ineq3, [x1, x2], [x1_sol,x2_sol])<=0
        if u1_sol >= 0 && u2_sol >= 0 && u3_sol >= 0
            KKTpoints = [KKTpoints; [x1_sol, x2_sol]];
        end
    end
end

disp(KKTpoints)
```

$$\begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 2 & 4 \end{pmatrix}$$

b) Identify the optimum from the previous points.

We evaluate the function at each point and check when it achieves the lowest value.

```
subs(obj, [x1, x2], [KKTpoints(1,1),KKTpoints(1,2)])
```

```
ans =
  1
```

```
subs(obj, [x1, x2], [KKTpoints(2,1),KKTpoints(2,2)]) % minimum value
```

```
ans =
  -3
```

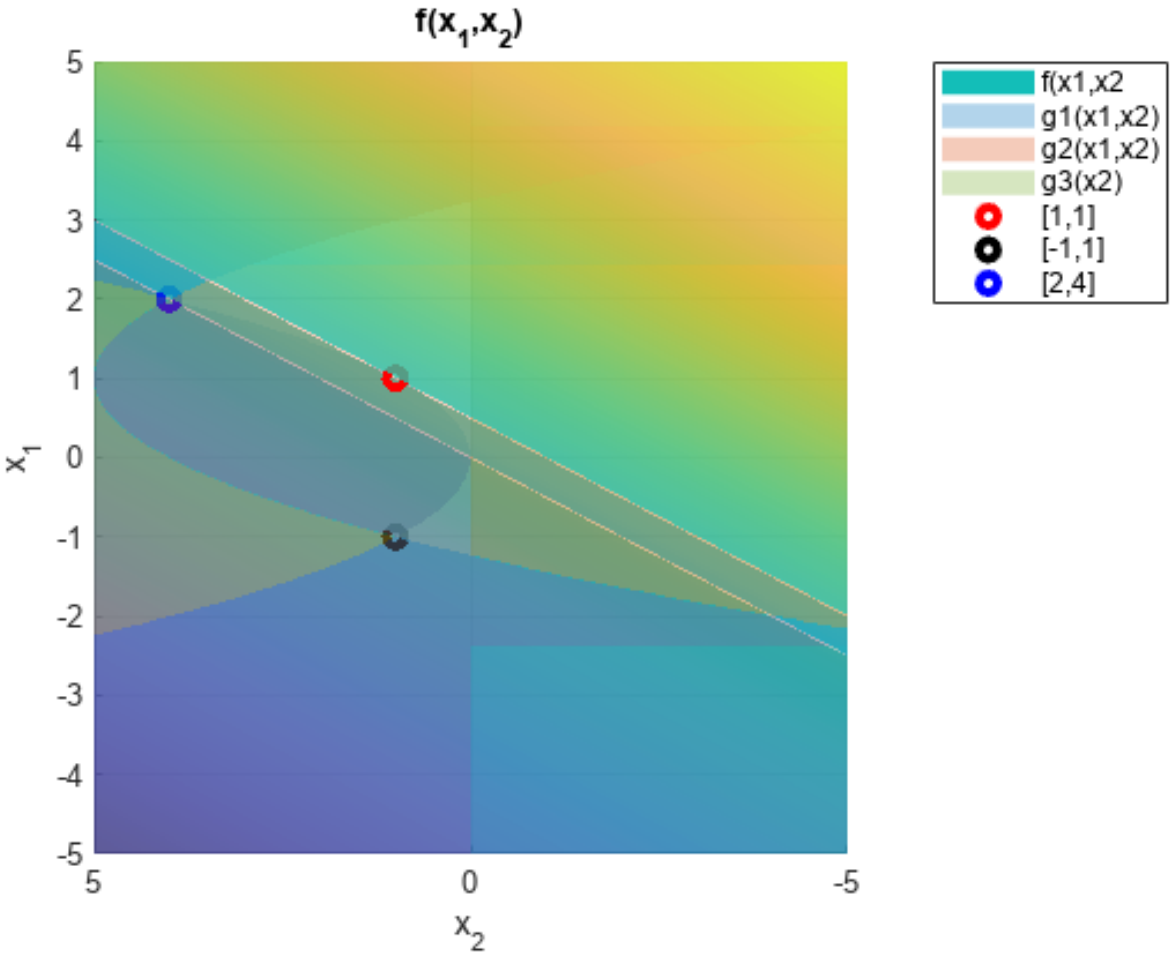
```
subs(obj, [x1, x2], [KKTpoints(3,1),KKTpoints(3,2)])
```

```
ans =
  0
```

The optimum from the previous points is $(-1, 1)$ with value -3.

We can represent the points in a figure

```
fig1;
hold on
plot3(1, 1, f(1,1),'ro','LineWidth',3);
plot3(-1, 1, f(-1,1),'ko','LineWidth',3);
plot3(2, 4, f(2,4),'bo','LineWidth',3);
legend('f(x1,x2','g1(x1,x2)','g2(x1,x2)','g3(x2)','[1,1]','[-1,1]','[2,4]')
view([-90 90])
```



c) Use MATLAB (or any other software) to identify the optimum starting from various points, for example, (-0.5, -0.5); (0.5, 0.5); (2, 1) and verify if the same solution is reached.

```
x0 = [-0.5 ; -0.5];
lb = [-inf, 0]
```

```
lb = 1x2
  -Inf     0
```

```
[x,fun,exitflag, output]=fmincon(@top_ejem_pnl_obj,x0,[],[],[],[],lb,[], ...
@top_ejem_pnl_res);
```

```
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
x % optimum achieved
```

```
x = 2x1
           -1
            1
```

```
x0 = [0.5 ; 0.5];
[x,fun,exitflag, output]=fmincon(@top_ejem_pnl_obj,x0,[],[],[],[],lb,[], ...
@top_ejem_pnl_res);
```

```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the value of the step size tolerance and constraints are
satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
x % nearly [0,0], not a feasible point
```

```
x = 2x1
   0.00060024
   2.4034e-07
```

This is almost [0,0].

```
x0 = [2 ; 1];
[x,fun,exitflag, output]=fmincon(@top_ejem_pnl_obj,x0,[],[],[],[],lb,[], ...
@top_ejem_pnl_res);
```

```
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
x % a kkt point
```

```
x = 2x1
            2
            4
```

Depending on the starting point, we get different solutions. This is due to the set of constraints of **our problem is not convex**, which means that the function can have more than one minimum, so we can not guarantee that we will achieve the global minimum when a local minimum is found. In these types of problems, the algorithm is extremely sensitive to the initial point. Additionally, as the convergence of the problem is difficult, we generally require specific algorithms for its resolution. For these type of problems, we could use the Global Optimization Toolbox or another external solver such as Knitro, but it is not the aim of this practice.

NOTE: all the `top_ejem_pnl` functions can be found on annexes.

d) Perform two iterations with the Zoutendijk algorithm starting from the point `[2, 1]`.

All the own-made function can be found on annexes.

First iteration

```
x0 = [2, 1]
```

```
x0 = 1x2
     2     1
```

```
%Active restrictions
restrictions = {ineq1, ineq2, ineq3};

for i = 1:numel(restrictions)
    result = round(subs(restrictions{i}, [x1, x2], x0),4);
    if result == 0
        disp(['The constraint ', num2str(i), ' is equal to zero.']);
    else
        disp(['The constraint ', num2str(i), ' is not equal to zero.']);
    end
end
```

```
The constraint 1 is not equal to zero.
The constraint 2 is not equal to zero.
The constraint 3 is not equal to zero.
```

```
grad = gradient(obj, [x1, x2])
```

```
grad =
  ⎛  2 ⎞
  ⎝ -1 ⎠
```

```
A = [-1,2,-1; 0,1,0; 0,0,1; 0,-1,0; 0,0,-1];
c = [1;0;0];
b = [0;1;1;1;1];

lb(1:3)=-Inf;
```

```
x=linprog(c,A,b,[],[],lb);
```

```
Optimal solution found.
```

```
display(x)
```

```
x = 3x1
    -3
    -1
     1
```

As $z = -3 < 0$ then $(-1, 1)$ is a descent direction.

```
xp = 0;
lb = 0;
[l,fun,flag]=fmincon(@top_ejem_pnl_obj2,xp,[],[],[],[],lb,[],@top_ejem_pnl_res2);
```

```
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
display(l)
```

```
l =
      0.69722
```

```
xk1=x0+l*[x(2),x(3)]
```

```
xk1 = 1x2
       1.3028       1.6972
```

The first iteration give us $x_1 = (1.3028, 1.6972)$.

Second iteration

```
for i = 1:numel(restrictions)
    result = round(subs(restrictions{i}, [x1, x2], xk1),4);
    if result == 0
        disp(['The constraint ', num2str(i), ' is equal to zero.']);
    else
        disp(['The constraint ', num2str(i), ' is not equal to zero.']);
    end
```

```
end
```

```
The constraint 1 is equal to zero.
The constraint 2 is not equal to zero.
The constraint 3 is not equal to zero.
```

```
gradienterest = gradient(ineq1, [x1, x2]);
A = [-1,2,-1;-1,-2*xk1(1),1;0,1,0;0,0,1;0,-1,0;0,0,-1];
c = [1;0;0];
b = [0;0;1;1;1;1];

lb(1:3)=-Inf;

x=linprog(c,A,b,[],[],lb);
```

```
Optimal solution found.
```

```
display(x)
```

```
x = 3x1
     -0.13148
      0.43426
            1
```

As $z = -0.1315 < 0$ then $(0.4343, 1)$ is a descent direction.

```
xp = 0;
lb = 0;
[l,fun,flag]=fmincon(@top_ejem_pnl_obj3,xp,[],[],[],[],lb,[],@top_ejem_pnl_res3);
```

```
Local minimum found that satisfies the constraints.

Optimization completed because the objective function is non-decreasing in
feasible directions, to within the value of the optimality tolerance,
and constraints are satisfied to within the value of the constraint tolerance.

<stopping criteria details>
```

```
display(l)
```

```
l =
       1.9655
```

```
xk2=xk1+l*[x(2),x(3)]
```

```
xk2 = 1x2
       2.1563       3.6627
```

Since the local minimum in that region is located at $(2, 4)$, it seems that $(2.1563, 3.6627)$ is a good result.

Exercise 2

For the following linear programming problem:

$$\begin{aligned} \min_{x_1,x_2\in\mathbb{R}^2} \quad & f(x_1,x_2) = -x_1 - x_2 \\ \text{such to} \quad & -x_1 + 2x_2 - 4 \leq 0 \\ & 2x_1 + x_2 - 6 \leq 0 \\ & -x_1, -x_2 \leq 0 \end{aligned}$$

Perform two iterations with the primal-dual interior-point algorithm, either by hand or by programming, whichever is more convenient.

Solution

The `primaldual` function is given on annexes.

```
A=[-1,2;2,1];
b=[4;6];
c=[-1;-1];

sol = primaldual(A, b, c, 0.001, 0.9, 0.1, 2);
display(sol);
```

```
sol = 2x1
        1.5052
        2.6461
```

Annexes

Exercise 1. c)

Objective function

```
function f= top_ejem_pnl_obj(x)
%% Objective function of the exercise 1, practice 3
    f = 2.*x(1) - x(2);
end
```

Constraints

```
function [c , ceq]= top_ejem_pnl_res(x)
%% Constraints associated to the objective function of the exercise 1, practice 3
   ceq=[ ];
   c=[-x(1).^2+x(2); (x(1)-1).^2+x(2)-5; -x(2)];
end
```

Exercise 1. d)

First iteration

Objective function

```
function f= top_ejem_pnl_obj2(l)
%% Objective function for the first iteration of the Zoutendijk algorithm
    f = -3*l;
end
```

Constraints

```
function [c , ceq]= top_ejem_pnl_res2(l)
%% Constraints associated to the objective function in the first iteration
% of the Zoutendijk algorithm
    ceq=[ ];
    c=[-(2-l)^2+1+l;(1-l)^2+l-4;-l-1];
end
```

Second iteration

Objective function

```
function f= top_ejem_pnl_obj3(l)
%% Objective function for the second iteration of the Zoutendijk algorithm
    x=[1.3028, 1.6972];
    d=[0.4343, 1];
    sust=x + l*d;
    f = 2*sust(1) -sust(2);
end
```

Constraints

```
function [c , ceq]= top_ejem_pnl_res3(l)
%% Constraints associated to the objective function in the second iteration
% of the Zoutendijk algorithm
    x=[1.3028,1.6972];
    d=[0.4343,1];
    sust=x+l*d;
    ceq=[];
    c=[-sust(1).^2 + sust(2);(sust(1) - 1).^2 + sust(2) - 5;-sust(2)];
end
```

Exercise 2

Primal-dual function

```
function result = primaldual(A, b, c, eps, rho, sigma, niter)
%% Primal-dual interior-point algorithm
x=A'*inv(A*A')*b;
u=inv(A*A')*A*c;
s=c-A'*u;

minx=min(x);
mins=min(s);
delx=max(-1.5*minx,0);
dels=max(-1.5*mins,0);

x=x+delx;
s=s+dels;
delxx=(1/2)*(x'*s)/sum(s);
delss=(1/2)*(x'*s)/sum(x);

x=x+delxx;
s=s+delss;

step=1;
aux=1;
while(step>eps || aux<=niter)

    n=length(x);
    mu=(sigma*s'*x)/n;
    e=ones(1, n)';
    X=diag(x);
    S=diag(s);
    I=eye(n);

    rows_S = size(S, 1); cols_S = size(S, 2);
    rows_X = size(X, 1); cols_X = size(X, 2);
    rows_A = size(A, 1); cols_A = size(A, 2); cols_A_transpose = size(A', 2);

    result = [S, zeros(rows_S, cols_X), X;
        A, zeros(rows_A, cols_X), zeros(rows_A, cols_A);
        zeros(cols_A_transpose, cols_S), A', I];

    temp = inv(result)*[mu*e-X*S*e;b-A*x;c-A'*u-s];

    dx = temp(1:size(S, 2));
    du = temp(size(S, 2) + 1:size(S, 2) + size(X, 2));
    ds = temp(size(S, 2) + size(X, 2) + 1:end);

    if ds >= 0
        disp('It does not exist a feasible solution');
        return;
    end
```

```
    if dx >= 0
        disp('The solution is not bounded');
        return;
    end

    dxneg = dx(dx < 0);
    for i = 1:numel(dxneg)
        alpha_p(i) = rho * min(-x ./ dxneg);
    end
    alpha_p = min(alpha_p);

    dsneg = ds(ds < 0);
    for i = 1:numel(dsneg)
        alpha_d(i) = rho .* min(-s ./ dsneg);
    end
    alpha_d = min(alpha_d);

    s = s + alpha_d.*ds;
    u = u + alpha_d.*du;
    x = x + alpha_p.*dx;

    step=transpose(c)*x-transpose(b)*u;
    aux = aux+1;
    result = x;
end
end
```