MATLAB solution of Constrained Optimization Problems

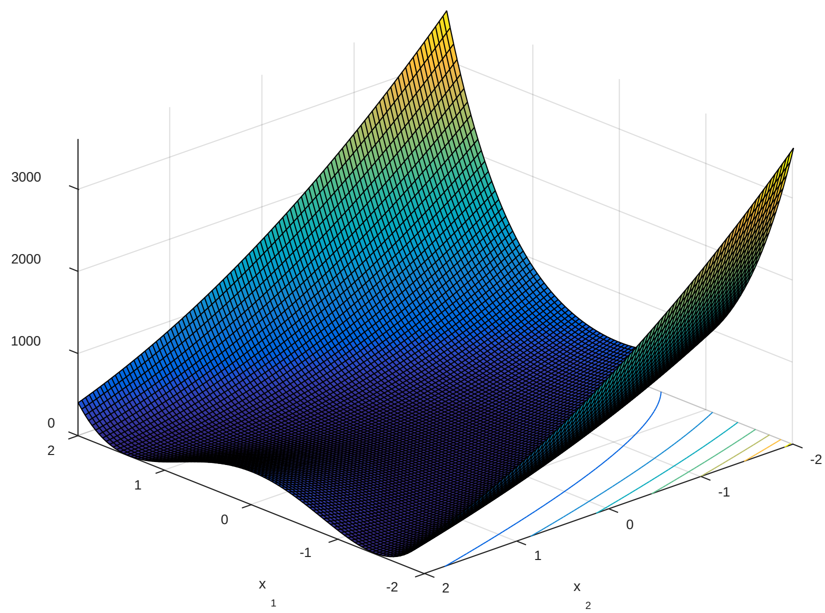
Constrained minimization problems can be solved in MATLAB using *fmincon* functions.

One of the advantages of *fmincon* is the number of algorithms and options it allows the user to implement. Further description can be found at:

https://www.mathworks.com/help/optim/ug/fmincon.html

The following example shows how it works for a constrained *Rosenbrock Valley Function*.

$$f = 100(x_2 - {x_1}^2)^2 + (1 - x_1)^2$$



This function has a global minimum at: $\begin{Bmatrix} 1 \\ 1 \end{Bmatrix}$

The search space is a half unit circle centered at the origin. The space is further constrained by:

$$x_1 + 2x_2 \leq 1$$

The program is listed below.

SQP (Sequential Quadratic Programming) is chosen for the search algorithm.

```
%Example on using fmincon for minimizing
% We use Rosenbrock function
% The function is subject to inequality constraint that limits the search
% to the area within a circle of radius 0.5

fun = @(x)100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
options = optimoptions('fmincon','Display','iter-detailed','Algorithm','sqp');

%Linear constraint x(1)+2x(2)<=1
A = [1,2];
b = 1;

%No equality constraints
Aeq = [];
beq = [];

%Lower and upper bounds of the variables
lb = [-1.0,-1.0];
ub = [1.0,1.0];

%nonlinear constraint
nonlcon = @circlecon;

%Initial guess
x0 = [0,0];
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)


function [c,ceq] = circlecon(x)
c = (x(1)-0)^2 + (x(2)-0)^2 - (1/2)^2;
ceq = [];
```

The results are:

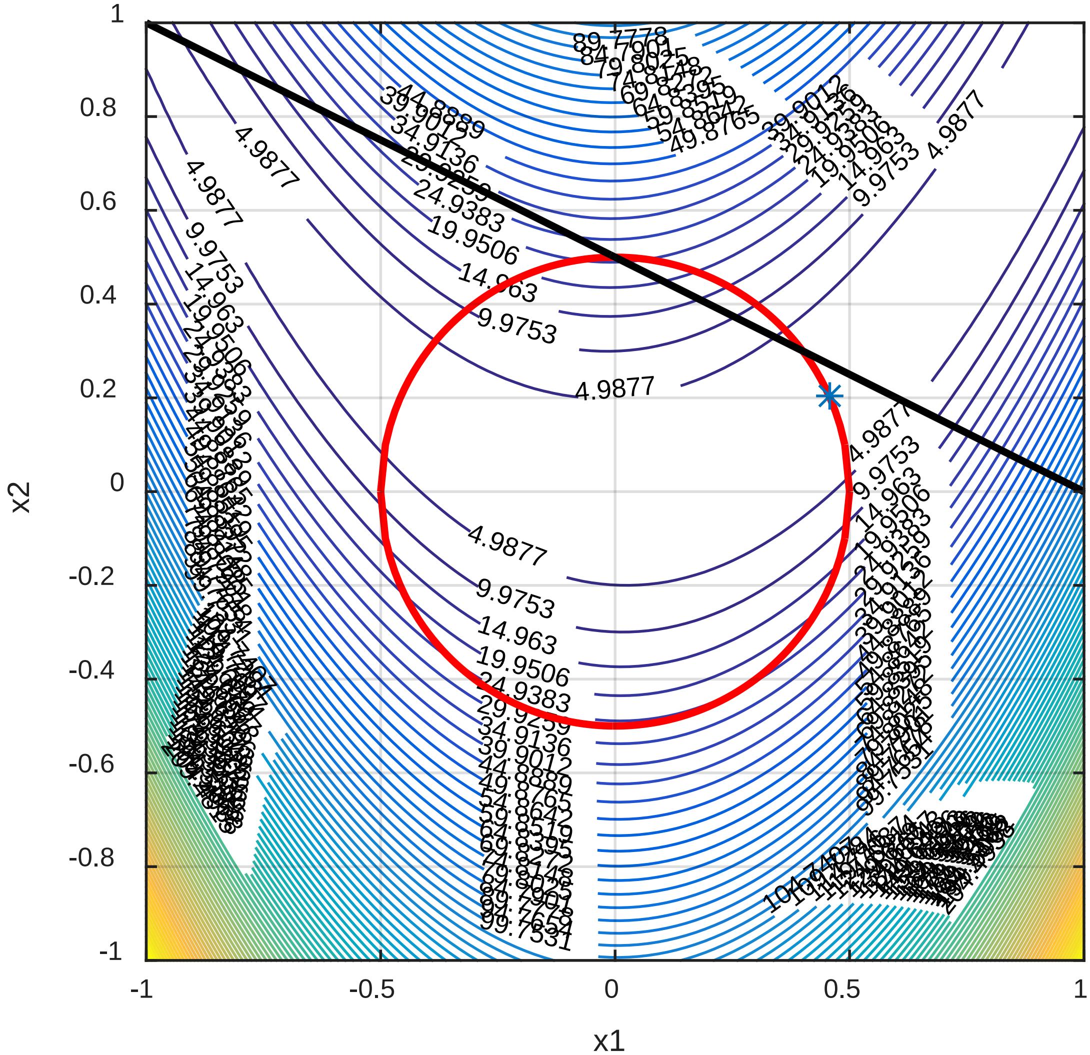
Iter	Func-count	Fval	Feasibility	Step Length	Norm of step	First-order optimality
0	**3**	**1.000000e+00**	**0.000e+00**	**1.000e+00**	**0.000e+00**	**2.000e+00**
1	**10**	**9.097794e-01**	**0.000e+00**	**2.401e-01**	**2.401e-01**	**1.153e+01**
2	**13**	**2.683976e-01**	**7.121e-02**	**1.000e+00**	**3.651e-01**	**3.462e+00**
3	**27**	**2.747601e-01**	**3.924e-02**	**1.977e-02**	**3.026e-02**	**3.336e+00**
4	**30**	**2.955718e-01**	**1.613e-03**	**1.000e+00**	**4.016e-02**	**5.338e-01**
5	**33**	**2.966303e-01**	**5.153e-06**	**1.000e+00**	**2.270e-03**	**8.607e-02**
6	**36**	**2.966215e-01**	**7.645e-08**	**1.000e+00**	**2.765e-04**	**2.590e-03**
7	**39**	**2.966216e-01**	**3.458e-11**	**1.000e+00**	**5.877e-06**	**3.115e-07**

Optimization completed: The relative first-order optimality measure, 3.114989e-07, is less than options. OptimalityTolerance = 1.000000e-06, and the relative maximum constraint violation, 3.458245e-11, is less than options. ConstraintTolerance = 1.000000e-06.

Optimization Metric	Options
relative first-order optimality = 3.11e-07	**OptimalityTolerance = 1e-06 (default)**
relative max(constraint violation) = 3.46e-11	**ConstraintTolerance = 1e-06 (default)**

x =

0.4556 0.2059


x1
x2
-1
-0.5
0
0.5
1
0.8
0.6
0.4
0.2
-0.2
-0.4
-0.6
-0.8
4.9877
9.9753
14.963
19.9506
24.9383
29.9259
34.9136
39.9012
44.8889
49.8765
54.8642
59.8519
64.8395
69.8272
74.8148
79.8025
84.7901
89.7778
99.7531

Alternative Approach: Combining Unconstrained Search (*fminsearch*) with Penalty Functions

Alternatively, we can use *fminsearch* with penalty function to solve the same problem as follows. Penalties are expressed using the bracket operators. The same penalty parameter, *R*, is used for both constraints.

```
close all
clear all
%initial guess
x0=[0 0];

options=optimset('LargeScale','off','Display','iter-detailed');
[x, fval,exitflag,output]=fminsearch(@Rosenbrock_Constrained,x0,options)

Rosenbrock(x)


function P=Rosenbrock_Constrained(x)

R=1000;

%Original function
f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;

%Constraints
g(1)=(x(1)-0)^2 + (x(2)-0)^2 - (1/2)^2;
g(2)=x(1)+2*x(2)-1;

if g(1)<=0
   gg(1)=0;
else
   gg(1)=R*g(1)^2;
end;

if g(2)<=0
   gg(2)=0;
else
   gg(2)=R*g(2)^2;
end;

P=f+gg(1)+gg(2);


function f=Rosenbrock(x)

f=100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

The table below shows the progression of solution as a function of the penalty parameter *R*, which is used for both constraints.

R	***P***	***F***	x_1	x_2	***No. of Function Evaluations***
1	0.2408	0.2202	0.5314	0.2799	105
10	0.2812	0.2678	0.4828	0.2314	95
100	0.2949	0.2931	0.4589	0.2088	94
1000	0.2964	0.2963	0.4560	0.2061	102

The table shows that the search approaches the minimum from outside the feasible range.

It is useful to remember that *fmincon* stopped at $(0.4556, 0.2059)^T$ using *39* function evaluations.

Use *fmincon* to solve Problems 7.31 and 7.34. Compare your earlier solutions with what you have done earlier.

Use *fmincon* to solve the three-truss problem (Section 7.22.1), pp. 467. Compare your solution to the results of this section. Please discuss your answer.

Note: in all these problems, use **'Display','iter-detailed'** in *optimoptions.*

.