```
from numpy import *
import pandas as pd
import matplotlib.pyplot as plt
```

## Question 1

$$\begin{aligned} min \quad & 100(x_1-1)^2+x_2 \\ s.t \quad & x_1+6x_2=36 \end{aligned}$$

```
Q=array([[200,0],[0,0]])
c=array([-200,1])
x0=array([0,6])   # we found the intitial feasible point

# we need to find x1=x0+d
# In order to fing d we will solve

# d=[d1,d2,mu]

J=array([[200,0,1],[0,0,6],[1,6,0]])
dQ=array([200,-1,0])   # [ -(gradient of fx at x0),0]
d=dot(linalg.inv(J),dQ)
print(d)

# Optimal point will bw  x1=x0+d             # if objective functionm is quadratic  and equality constraints are linear then method converge in one iteration

print("\n\nOptimal point will be",x0+d[:2])
```

```
[ 1.00083333 -0.16680556 -0.16666667]


Optimal point will be [1.00083333 5.83319444]
```

## Question 2

```
def g1(x):
  return -1*(8 - x[0]**2 - x[1]**2 - x[2]**2 - x[3]**2 - x[0] + x[1]- x[2] + x[3])

def g2(x):
  return -1*(10 - x[0]**2 - 2*x[1]**2 - x[2]**2 - 2*x[3]**2 + x[0] + x[3])


def gradf(func,x):
    f0,n,h1=func(x),len(x),pow(10,-5)
    g=zeros((n,1),dtype=float)
    for i in range(0,n):
        x1=x.copy()
```

```
        x1[i]=x1[i]+h1
        g[i]=(func(x1)-f0)/h1
    return g
```

```
sigma=10
def func(x):
  return x[4] - (1/(sigma))*(log(x[4] - g1(x)) + log(x[4]- g2(x)))
```

```
# t=max(gi(x))+1
x = array([1,1,1,1,-3])
```

```
g1(array([1,1,1,1]))
```

```
-4
```

```
g2(array([1,1,1,1]))
```

```
-6
```

```
gradf(func,x)
```

```
array([[    0.        ],
       [    0.        ],
       [    0.        ],
       [    0.        ],
       [90191.70746988]])
```

```
from numpy import *
#from decimal import *
#getcontext().prec =50
def gradf(fun,x):
    n,h1=len(x),pow(10,-7)
    g=zeros((n,1),dtype=float)
    for i in range(0,n):
        x1,x2=x.copy(),x.copy()
        x1[i],x2[i]=x1[i]+h1,x2[i]-h1
       # print(x,x1)
        g[i]=(fun(x1)-fun(x))/(h1)
    return g
def quasi_newton(fun,con,x0):
    beta1,beta2,r,eps,iter1,n=pow(10,-4),0.9,0.5,pow(10,-5),0,len(x0)
    B0=identity(n,dtype=float)
    f0,g0=fun(x0),gradf(fun,x0)
    alpha=1
    while linalg.norm(g0)>eps and iter1<20000 and alpha>pow(10,-5):
        d0,alpha=-dot(linalg.inv(B0),g0),1
        #print(g0.T@d0)
        while max(con(x0+alpha*d0))>-0.000001:
            alpha=alpha*r
        #print(alpha)
        x1=x0+alpha*d0
```

```
            f1,g1=fun(x1),gradf(fun,x1)
            #print(f1-f0-alpha*beta1*g0.T@d0,dot(g1.T,d0)-beta2*dot(g0.T,d0))#or dot(g1.T,d0)<beta2*dot(g0.T,d0)
            while (f1>f0+alpha*beta1*g0.T@d0) and alpha>pow(10,-5):
                alpha=alpha*r
                x1 = x0 + alpha * d0
                f1, g1 = fun(x1),gradf(fun,x1)
            #print('f1=',f1)
            dt1,s1=x1-x0,g1-g0
            #print('xx=',dt1.T@s1,'alpha=',alpha)
            if dt1.T@s1>pow(10,-3):
                B0=B0+1/(dt1.T@s1)*s1@s1.T-1/(s1.T@B0@s1)*B0@s1@s1.T@B0
            #print(B0)
            # print(alpha,fun(x0+alpha*d0)-fun(x0))
            x0,g0,iter1=x1,g1,iter1+1
        if iter1>=20000:
            print('maximum iteration attains')
        #else:
        #    print('gg=',linalg.norm(g0))
        return x0
    def interior_point_solver(obj_fun,con_fun,x0):
        print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
        sigma,opt_cond,iter1=10.0,1.0,0.0
        if max(con_fun(x0)) > -pow(10,-5):
            print('initial point is not strictly feasible. So starting phase 1')
            n = len(x0)
            y0 = zeros((n + 1, 1), dtype=float)
            y0[0:n], y0[n], sigma = x0, max(con_fun(x0) + 1), 10.0
            print(y0)
            def con_fun_phase_1(x):
                n = len(x)
                return con_fun(x[0:n]) - x[-1]
            while max(con_fun(y0[0:n])) > -0.001:
                def barr_phase_1(x):
                    return x[-1] - 1 / sigma * sum(log(-con_fun_phase_1(x)))

                #y0 = optimize.fmin_cg(con_phase_1, y0,fprime= lambda x: gradf(con_fun_phase_1,x))
                y0 = quasi_newton(barr_phase_1, con_fun_phase_1, y0)
                #print(y0)
                sigma, iter1 = sigma * 10, iter1 + 1
            x0 = y0[0:n]
            print('Phase I complete')
            print('interior point=', x0,'\n','constraint_value=', con_fun(x0))
        else:
            print('initial approximation is an interior point so starting phase II directly')
        sigma=10
        opt_cond=1
        while len(con_fun(x0))/sigma > 0.00000001 and opt_cond >pow(10,-5):  # and max(con_fun(x0))<-0.00001:
            def barr_fun(x):
                return obj_fun(x) - 1 / sigma * sum(log(-con_fun(x)))
            x0 = quasi_newton(barr_fun, con_fun, x0)
            #print(obj_fun(x0), con_fun(x0))
            opt_cond = linalg.norm(gradf(barr_fun, x0))
            print(opt_cond)
            iter1+=1
            sigma=sigma*5
```

```
    print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
    print(sigma)
    if len(con_fun(x0))/sigma <=0.00000001:
        print('maximum iterations attends')
    else:
        print('optimal solution found as norm KKT=',opt_cond,'<10^-7')
    return x0,obj_fun(x0), con_fun(x0),iter1, -10/sigma*1/con_fun(x0)
#########################################################################


from numpy import *
def obj_fun(x):
    return x[0]**2+x[1]**2+2*(x[2]**2)+x[3]**2-5*x[0]-5*x[1]-21*x[2]+7*x[3]
def con_fun(x):
    g=zeros((2,1),dtype=float)
    g[0]=-1*(8 - x[0]**2 - x[1]**2 - x[2]**2 - x[3]**2 - x[0] + x[1]- x[2] + x[3])
    g[1]=-1*(10 - x[0]**2 - 2*x[1]**2 - x[2]**2 - 2*x[3]**2 + x[0] + x[3])

    return g

x0,fval,con_val,iter1,lagrange_mult=interior_point_solver(obj_fun,con_fun,10*ones((4,1),dtype=float))
print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
print('optimalpoint=',x0,'\n')
print('objective value=',fval,'\n')
print('constraint value=',con_val,'\n')
print('no of iterations=',iter1)
print('Lagrange multiplier=',lagrange_mult)
print('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%')
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
initial point is not strictly feasible. So starting phase 1
[[ 10.]
 [ 10.]
 [ 10.]
 [ 10.]
 [571.]]
Phase I complete
interior point= [[-0.30105116]
 [ 0.33406242]
 [-0.40052376]
 [ 0.41703172]]
 constraint_value= [[-8.91610482]
 [-9.29390316]]
5.803442343287987e-06
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50
optimal solution found as norm KKT= 5.803442343287987e-06 <10^-7
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
optimalpoint= [[ 0.37284223]
 [ 1.05397962]
 [ 2.06768455]
 [-0.63900084]]

objective value= [-44.81964552]

constraint value= [[-0.04092613]
 [-2.28112042]]
```

```
no of iterations= 2.0
Lagrange multiplier= [[4.88685339]
 [0.08767621]]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```