```
import numpy as np
import torch

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

def g1(x):
  return x[1]**2 - 2*x[0]

def g2(x):
  return (x[1]-1)**2 + 5*x[0] -15

def dfdx(x):
  return torch.tensor([[2*x[0],2*(x[1]-3)]])

def dfdg1(x):
  return torch.tensor([[-2,2*x[1]]])

def dfdg2(x):
  return torch.tensor([[5,2*(x[1]-1)]])

def ch_weights(mu, weights,k):
  if k > 0:
    weights = torch.max(abs(mu), 0.5 *(weights + abs(mu)))
  else:
    weights = abs(mu)
  return weights

def marit_fun(x,weights,alpha,s):
  G1=max(0, g1(x + alpha*s)) # constrain 1
  G2=max(0, g2(x + alpha*s)) # constrain 2
  return fun(x + alpha*s) +  weights[0,:] * G1 + weights[1,:]* G2

def dFdalpha(x,weights,s):
  if g1(x) <= 0:
       dg1_da = 0
  else:
      dg1_da= torch.matmul(dfdg1(x), s)
  if g2(x) <= 0:
      dg2_da = 0
  else:
    dg2_da= torch.matmul(dfdg2(x), s)
  dF_da = torch.matmul(dfdx(x), s) + (weights[0, :] * dg1_da +
weights[1, :] * dg2_da)
  return dF_da

def lineSearch(x,mu,weights,s,k):
    t = 0.25
    alpha = 1
    weights = ch_weights(mu, weights,k)
    phi = lambda x, weights, alpha, t, dFdalpha: marit_fun(x, weights,
```

```
0, 0) + alpha * t * dFdalpha(x,weights,s)

    while phi(x, weights, alpha, t, dFdalpha) < marit_fun(x, weights,
alpha, s):
        alpha = 0.5 * alpha
    return alpha, weights

def BFGS(x,W, s, mu, alpha):
  lx_k = dfdx(x) + torch.matmul(mu.T, torch.tensor([g1(x),g2(x)]))
  lx_k_1 = dfdx(x + alpha*s) + torch.matmul(mu.T, torch.tensor([g1(x +
alpha*s),g2(x + alpha*s )]))

  delta_l = lx_k_1 -lx_k

  Q = torch.matmul(torch.matmul((alpha*s).T, W), (alpha*s))
  if torch.matmul((alpha*s).T, delta_l.T) >= 0.2 *
torch.matmul(torch.matmul((alpha*s).T, W), (alpha*s)):
        theta = 1
  else:
        theta = 0.8 * Q / (Q - torch.matmul((alpha*s).T, delta_l.T))
  y = theta * delta_l.T + (1 - theta) * torch.matmul(W, (alpha*s))
  W = W + torch.matmul(y, y.T) / torch.matmul(y.T, s) -
torch.matmul(torch.matmul(W, s), torch.matmul(s.T, W)) /
torch.matmul(torch.matmul(s.T, W), s)
  return W

def LangMulticheck(mu,active):
    mu_check = 0
    if len(mu) == 0 or min(mu) > 0:
        mu_check  = 1
    else:
        mu_idx = np.argmin(np.array(mu))
        mu = mu[mu!=min(mu)]
        active.pop(mu_idx)
    return active, mu_check ,mu

def solve_sqp(x, W):
    active = []
    A_initial = torch.cat((dfdg1(x),dfdg2(x)),0)
    B_initial=torch.tensor([[g1(x), g2(x)]]).T
    mu_initial = torch.zeros((B_initial.shape[0], 1))
    mu = []
    while True:
      if len(active) == 0:
            s_mu = torch.matmul(torch.linalg.inv(W), -dfdx(x).T)
            s = s_mu[:2, :]
      if len(active) > 0:
        if len(active) == 1:
                A = A_initial[active[0], :].reshape(1, -1)
                B = B_initial[active[0], :].reshape(1,1)
        if len(active) == 2:
```

```
                A = A_initial
                B = B_initial

          Z = torch.zeros((A.shape[0], A.shape[0]))
          matrix=torch.cat((torch.cat((W,A.T),1),torch.cat((A,Z),1)),0)
          j=torch.cat((-dfdx(x).T,-B),0)
          s_mu = torch.matmul(torch.linalg.inv(matrix), j)
          s = s_mu[:2, :]
          mu = s_mu[2:, :]

        if len(mu) == 1:
            mu_initial[0] = s_mu[2:3, :]
        if len(mu) == 2:
            mu_initial[0] = s_mu[2:3, :]
            mu_initial[1] = s_mu[3:, :]

        sqp_constraint = torch.round((torch.matmul(A_initial,
s.reshape(-1, 1)) + B_initial))
        active, mu_check,mu = LangMulticheck(mu,active)

        if torch.max(sqp_constraint) <= 0 and mu_check == 1:
              return s, mu_initial
        else:
            index = np.argmax(sqp_constraint)
            active.append(index)
            active = np.unique(np.array(active)).tolist()

x = torch.tensor([[1,1.]]).T
x_initial = x
mu = torch.zeros((x.shape[0], 1))
weights = torch.zeros((x.shape[0], 1))+2
W = torch.eye(x.shape[0])
eps = 1e-3
k = 0

delta_L_norm = np.linalg.norm(dfdx(x) + np.matmul(mu.T,
torch.cat((dfdg1(x),dfdg2(x)),0)))

while delta_L_norm > eps:
    s, mu = solve_sqp(x, W)
    a, weights = lineSearch(x,mu,weights,s,k)
    weights_old = weights
    W = BFGS(x,W, s, mu, a)
    x += a*s
    k += 1
    delta_L_norm = np.linalg.norm(dfdx(x) + np.matmul(mu.T,
torch.cat((dfdg1(x),dfdg2(x)),0)))

X_reults = x.T
print("The value of x1 and x2 is:\nx1={}\nx2={}".format(x[0][0],x[1]
```

```
[0]))
print('\n')
print('The minumim value of the function:{}'.format(fun([x[0][0],x[1]
[0]])))
print('\n')
print("The inequality constraints are:\ng1={}\ng2={}".format(g1([x[0]
[0],x[1][0]]),g2([x[0][0],x[1][0]])))
```

```
The value of x1 and x2 is:
x1=1.0601943731307983
x2=1.45615553855896


The minumim value of the function:3.507467746734619


The inequality constraints are:
g1=2.384185791015625e-07
g2=-9.490949630737305
```