**Home Assignment 4**

Daniel Smotritsky, Ilana Pervoi

1. **Equality constrained optimization**

The Lagrangian:

We will find the derivative according to each unknown and set it to 0:

Meaning the critical point is **(1,1,1)**.

1. To show that the critical point is maximum point we will show:

for all such that

Since

and both cannot be 0 at the same time since and then we will obtain in contradiction to what is given, hence:

1. **General constrained optimization**
2. We will start assuming the two inequality constraints are not active:

The Lagrangian:

Notice that the constraint does not obtained when : , hence we need to activate inequality constraint.

When activating :

The Lagrangian:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
| hence KKT conditions are not obtained – it could not be a critical point. | KKT are obtained.  The constraint is obtained. |

Overall, is a critical point for which only one inequality constraint is active and that holds KKT conditions.

1. First, we will find the Hessian:

We can notice that:

We will find the eigen-values by the characteristic polynomial:

The first matrix:

The second matrix:

Overall, we obtain two eigen-values for :

We saw in the previous question that , meaning the eigen-values hold:

1. the penalty function. The minimization problem will look like that:

We will find the gradient:

We obtain: – almost as we got in (b).

**# Objective function**

**def f(m):**  
 def funcToReturn(x):  
 x\_1 = x[0]  
 x\_2 = x[1]  
 return (x\_1 + x\_2) \*\* 2 - 10 \* (x\_1 + x\_2) + m \* (3 \* x\_1 + x\_2 - 6) \*\* 2 + m \* (max(0, (x\_1 \*\* 2) + (  
 x\_2 \*\* 2) - 5)) \*\* 2 + m \* (max(0, -x\_1)) \*\* 2  
  
 return funcToReturn  
  
  
**def Armijo(x, objective\_func, gradient, d, alpha\_0, beta, c):** alpha = alpha\_0  
 fx = objective\_func(x)  
 for j in range(10):  
 if objective\_func(x + alpha \* d) <= fx + c \* alpha \* np.dot(gradient, d):  
 break  
 else:  
 alpha = beta \* alpha  
 return alpha

**# Steepest(or Gradient) Descent (SD)**  
**def SD(maxiter, epsilon):**  
 w\_initial = np.zeros(2)  
 anwserByX = [w\_initial]  
 w\_curr = w\_initial  
  
 grad\_w\_init = np.asarray([-10 + (6 \* 0.01 \* -6), -10 + (2 \* 0.01 \* -6)])  
 for i in range(maxiter):  
 m = 0.01  
 if 0 < i < 20:  
 m = 0.01  
 if 19 < i < 40:  
 m = 0.1  
 if 39 < i < 60:  
 m = 1  
 if 59 < i < 80:  
 m = 10  
 if 79 < i < 100:  
 m = 100  
 w\_prev = w\_curr  
 x\_1 = w\_prev[0]  
 x\_2 = w\_prev[1]  
  
 gradient = np.asarray(  
 [2 \* (x\_1 + x\_2) - 10 + 6 \* m / 2 \* (3 \* x\_1 + x\_2 - 6) + 4 \* x\_1 \* m / 2 \* max(0, (x\_1 \*\* 2) + (  
 x\_2 \*\* 2) - 5) - 2 \* m / 2 \* max(0, -x\_1),  
 2 \* (x\_1 + x\_2) - 10 + 2 \* m / 2 \* (3 \* x\_1 + x\_2 - 6) + 4 \* x\_2 \* m / 2 \* max(0,  
 (x\_1 \*\* 2) + (  
 x\_2 \*\* 2) - 5)])  
  
 alpha = Armijo(w\_prev, f(m), gradient, -gradient, 1, 0.5, 1e-4)  
  
 w\_curr = w\_prev + alpha \* -gradient  
 anwserByX.append(w\_curr) # for plotting  
 if i > 0 and np.linalg.norm(gradient)/

np.linalg.norm(grad\_w\_init) < epsilon:  
 break  
 return w\_curr  
  
  
print(SD(100, 0.01))

1. **Box-constrained optimization**

We will find the derivative of and set it to zero:

Hence, is minimum point if and only if it obtains otherwise,

since is a convex function, we will say the minimum is the bound as followed:

1. Need to show that the minimization for each scalar is given by:

Denote:

In Coordinate Gradient we look at each iteration at different , meaning for all , are constant to this iteration. Hence at iteration those constants would disappear when we look at the derivative according to , meaning there is no need in the most left summation.

Moreover, is SPD, hence for , meaning we could replace the division by 2 by calculating one set such that (only ) and keep the division to the diagonal only:

Now we can notice that is in the format of the previous question, when

and .

We already saw that the minimum point is in .

**def CD(H, g, a, b, x, maxIter):**  
 for k in range(maxIter):  
 for i in range(np.shape(H)[0]):  
 upper\_g = -g[i]  
 for j in range(np.shape(H)[0]):  
 if i != j:  
 upper\_g = upper\_g + x[j] \* H[j][i]  
  
 lower\_h = H[i][i]  
 x[i] = -upper\_g / lower\_h  
 if x[i] < a[i]:  
 x[i] = a[i]  
 if x[i] > b[i]:  
 x[i] = b[i]  
  
 return x

1. The solution obtained by the code above and the given parameters is:
2. **Projected Gradient Descent for LASSO regression**
3. Need to show that the problem is equivalent to:

And the final solution is

We will argue that for all solutions to the problem above, it holds:

For all , or .

In other words, for each entry in the solutions vectors, one must be 0.

We will prove this argument later and assume its correctness for now.

Let us look at vector as described.

Notice that all vectors can be written as , meaning it is correct for any as required.

Proof of the argument:

Let solutions to the minimization problem above.

Assume in contradiction that there is an entry that holds .

Let us look at new vectors built as followed:

For :

For :

If :

If :

If :

If :

In both situations, we obtain

Meaning,

Meaning are better solutions to the min. problem – in contradiction to the optimality of

1. We need to solve with projected SD. As suggested in the assignment, we will define , hence to obtain we will multiply with , block matrix and we will get .

Now the problem will look like this: