CS164 ASSIGNMENT 1: CONSTRAINED OPTIMIZATION
& THE KKT CONDITIONS
November 2020
Akmarzhan Abylay

Problem 1

Write down the KKT conditions for the LP and explain why the optimal solution will in general never be found in the interior of the feasible region and will always be on a vertex or facet (the higher-dimensional equivalent of a face).

Answer: The KKT conditions will be as follows:

1. **Stationarity:** We want to minimize, so we would make sure that the gradients of objective function and constraints (weighted) add up to 0.

$$\nabla_x f(x) + \mu^T \nabla_x g(x) = c + \mu^T A = 0$$

2. **Equality Constraints:** There are no equality constraints, so we would simply need m multipliers.
3. **Inequality constraints (i.e., Complementary Slackness Condition):** ensures that only the active constraints are taken into account and have non-zero multipliers.

$$\mu^T g(x) = 0$$
$$\mu^T (Ax - b) = 0, \; where \; \mu_i \geq 0$$

Both the Linear Program's objective and constraints functions are linear, which means that the problem's feasible set will represent a polyhedron (i.e., the intersection of half-spaces). Only the contour lines (i.e., level sets) that cut through or intersect the feasible set satisfy all of the constraints. Since our feasible set is bounded, and the contour lines are straight and parallel to each other, we cannot get a better solution if we move in the direction orthogonal to the objective function. Thus, as we move in some direction orthogonal to the contour lines, the values will be strictly smaller or larger. The optimum occurs when the value at a contour line intersecting the polyhedron is maximum or minimum for the available segment. This happens on a vertex (or edges) as it is the end of the polyhedron. This is why we can only check the vertices to find the optimal point. This holds if both the objective and constraint functions are linear.

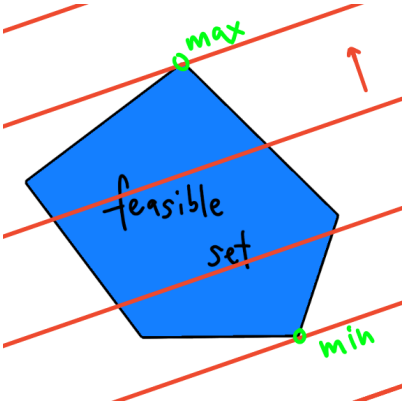


Figure 1: Diagram for Problem 1

For example, consider the above diagram with a simple scenario. The polyhedron represents the feasible set, and the equidistant parallel lines represent the level set for some linear objective function. For this specific diagram, I assumed that the values would increase from right to left diagonally. Out of all intersections

of these lines with the feasible set, the ones in the vertices give us the maximum and minimum values, as the figure shows. This supports the previous proof.

Problem 2

Show how the l_1 and l_∞ regression problems can be expressed as linear programs, by defining slack variables and inequality constraints as needed.

Answer: Let us start from the l_1, which represents the sum of the absolute values. In this case, the regression problem could be simply represented as minimizing the residuals, which is the difference between the predicted and actual values.

$$\min_{\text{x}} Ax - b$$

Here A represents the matrix with weights, while x is the data vector itself. Their matrix multiplication Ax gives us the predicted value, and b is the actual value, so the difference between them gives us the residuals. Since in most cases we can't perfectly predict the data (or we won't since it might be a sign of over-fitting), $Ax \neq b$, which is why we are even calculating the residuals as they are an indicator of how much our prediction deviates from the true value.

Now we want to define an l_1 regression problem, which could be written as:

$$\min_{\text{x}} ||Ax - b||_1,\ A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$$

To transform the problem into an LP, we can equivalently write this using an additional slack variable s:

$$\begin{aligned} &\min_{\text{x, s}} \sum_{i=1}^{m} s_i \\ &s.t. : |a_i^T x - b_i| \leq s_i,\ i = 1, ..., m \end{aligned}$$

Each slack variable s_i here provides an upper bound for the absolute value of corresponding residuals $|a_i^T x - b_i|$, so when we minimize s (i.e., slack variable), we penalize the total absolute value of deviation of all points. This is the same thing as writing:

$$\begin{aligned} &\min_{\text{x, s}} 1^T s \\ &s.t. : a_i^T x - b_i \leq s_i, \quad i = 1, ..., m \\ &s.t. : a_i^T x - b_i \geq -s_i,\ i = 1, ..., m \end{aligned}$$

Now let us focus on the l_∞ regression problem. The problem is defined as:

$$\min_{\text{x}} ||Ax - b||_\infty,\ A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m$$

The infinity-norm here would penalize the maximum deviation over all points. Introducing slack variables, we get:

$$\begin{aligned} &\min_{\text{x, s}} s \\ &s.t. : ||Ax - b||_\infty \leq s \end{aligned}$$

If we define the range for the maximum value, all the other values will also fall within that range, since they are smaller. Analytically:

$$||Ax - b||_\infty \leq s \rightarrow \max_{\text{i=1..m}} |a_i^T x - b_i| \leq s \rightarrow |a_i^T x - b_i| \leq s,\ i = 1, ..., m$$

This means that we can define our linear program to be:

$$\min_{x,\,s} s$$
$$s.t. : a_i^T x - b_i \leq s, \quad i = 1, ..., m$$
$$s.t. : a_i^T x - b_i \geq -s, \; i = 1, ..., m$$

We should note that l_1 requires n slack variables for n data points, while l_∞ only requires 1.

Problem 3

Use the CVXPY package to solve the l_1 and l_∞ optimization problems, and compute the parameters that define the line of best fit. Plot both lines in different colours over the scatter plot of the random datapoints.

Answer: Using the CVXPY library, I defined the needed variables and constraints to get the minimum error of around 4866 for l_1 (i.e., the total absolute value of the residuals) and around 76 for l_∞ (i.e., maximum deviation over all data points). We got a slope of 2.01 and an intercept of 2.63 for l_1, while we obtained a slope of 2.18 and an intercept of -15.5 for l_∞. Their objective functions were different, but if evaluating the common sum of squared residuals, we see that the l_1 is performing better than l_∞ (209424 vs. 187345, which are both terrible).

Figure 2 represents the best fit results and lines (see Appendix for code and SSR calculation). We can see that both of the lines are relative OK in predicting the data. Still, the red line (i.e., norm-infinity) is particularly sensitive to outliers, as it tries to minimize the maximum value within residuals out of all 200 points, which is why the line is a little tilted and doesn't fit most of the data well. Norm-1, on the other hand, is pretty robust, since it gives equal weight to all observations (e.g., unlike least squares, which penalizes large deviations more). However, it can produce multiple results (i.e., lines of best fit) as there can be many lines that have the same sum of absolute residuals.

I also tried playing with regularization for this problem by penalizing the biggest coefficient, but (1) there was no use since the coefficients weren't very big in the first place, and (2) the results weren't different from the case without regularization.

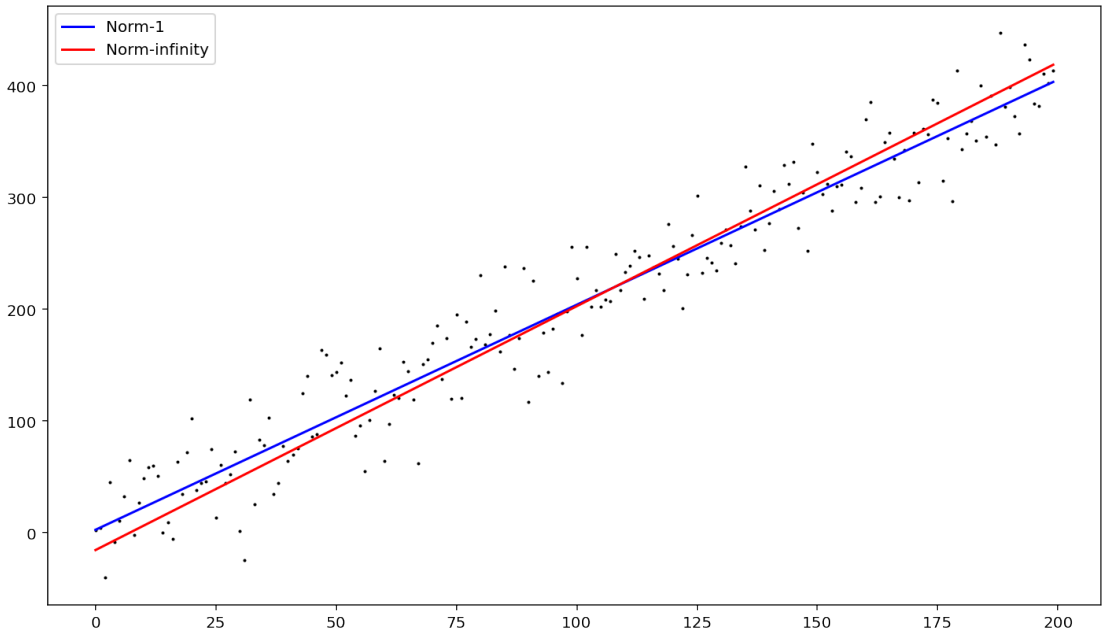


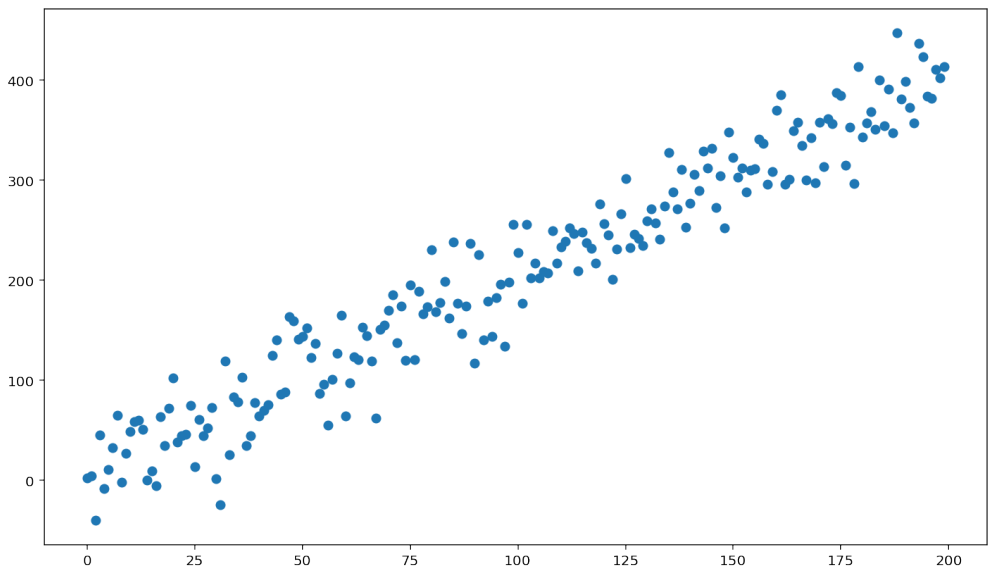
Figure 2: Plot for Problem 3

Appendix

[1]:
```
#loading the needed libraries
import cvxpy as cp
import numpy as np
import pandas as pd
```

[2]:
```
### Below is the code that generates the synthetic data from the assignment.

# l_1 and l_infinity regression using cvxpy
import numpy as np
import cvxpy as cvx
import matplotlib.pyplot as plt

# generate a synthetic dataset

# actual parameter values
theta1_act = 2
theta2_act = 5

# Number of points in dataset
N = 200

# Noise magnitude
mag = 30

# datapoints
x = np.arange(0,N)
y = theta1_act * x + theta2_act *np.ones([1,N]) + np.random.normal(0,mag,N)

plt.figure()
# Scatter plot of data
plt.scatter(x,y)
plt.show()
```

[2]:



[3]:
```
extra = np.ones(N).reshape(N, 1)

#defining the Xs and Ys
X = np.hstack((x.reshape(N, 1), extra))
Y = y.flatten()

#defining the weights
theta_1 = cvx.Variable(2)
theta_inf = cvx.Variable(2)

#slack variables, since we have N for l_1 and 1 for l_infty
slack_1 = cvx.Variable(N)
slack_inf = cvx.Variable()

#objective functions
obj_1 = cvx.Minimize(extra.T@slack_1)
obj_inf = cvx.Minimize(slack_inf)

#constraints
const_1 = [Y-X@theta_1<=slack_1,Y-X@theta_1>=-slack_1]
const_inf = [Y-X@theta_inf<=slack_inf,Y-X@theta_inf>=-slack_inf]

#solving for l_1
prob_1 = cvx.Problem(obj_1, const_1)
prob_1.solve()
print("status:", prob_1.status, "(Norm-1)")
print("optimal value:", prob_1.value)
print("optimal var:", np.round(theta_1.value, 2))
```

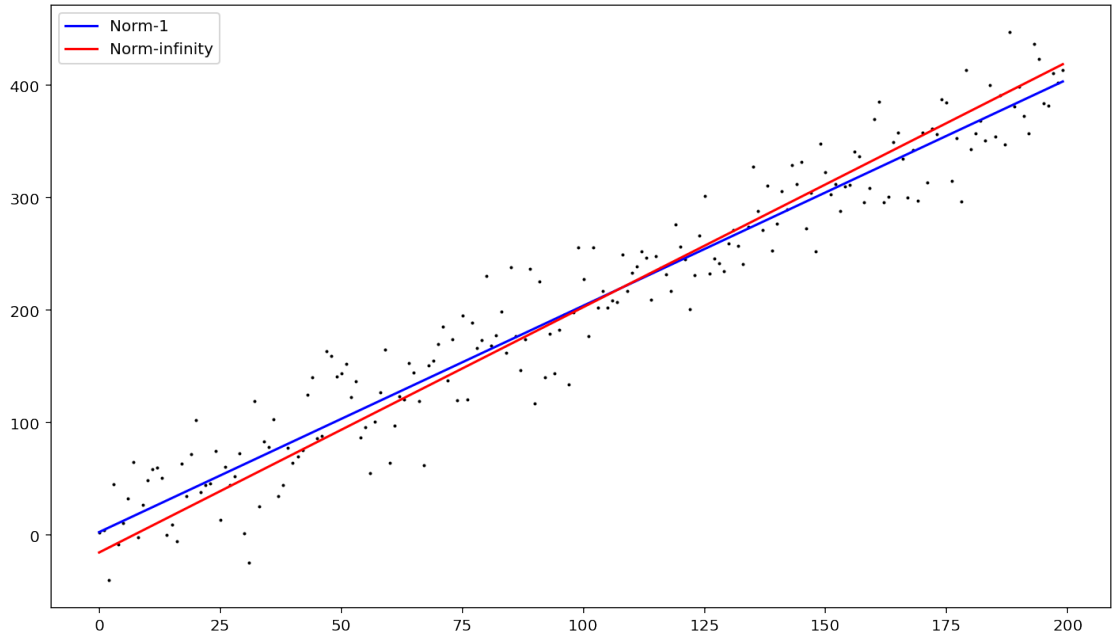
```
status: optimal (Norm-1)
optimal value: 4866.189128147369
optimal var: [2.01 2.63]
```

[4]:
```
#solving for l_infty
prob_inf = cvx.Problem(obj_inf,const_inf)
prob_inf.solve()
print("status:", prob_inf.status, "(Norm-infinity)")
print("optimal value:", prob_inf.value)
print("optimal var:", np.round(theta_inf.value, 2))
```

```
status: optimal (Norm-infinity)
optimal value: 76.35425992068578
optimal var: [2.18 -15.5]
```

[5]:
```
#finding the line of best fit using the terms found
line_1 = x*theta_1.value[0]+theta_1.value[1]
line_inf = x*theta_inf.value[0]+theta_inf.value[1]

#plotting the lines in different colors
plt.scatter(x, y, c="k")
plt.plot(x, line_1, c="blue", label="Norm-1")
plt.plot(x, line_inf, c="red", label="Norm-infinity")
plt.legend(loc="best")
```

[5]:



[46]:

```
lss_1 = [(line_1[i]-y.tolist()[0][i])**2 for i in range(len(line_1))]
lss_inf = [(line_inf[i]-y.tolist()[0][i])**2 for i in range(len(line_inf))]

print("Sum of squared residuals for norm-one:", sum(lss_1))
print("Sum of squared residuals for norm-infinity:", sum(lss_inf))
```

```
Sum of squared residuals for norm-one: 187345.33242414336
Sum of squared residuals for norm-infinity: 209423.827944027
```