Experimenting Empirical Evaluation of Dual Function and Duality Gap

Jackie Zhong (jackie.z@wustl.edu)

18 December 2021

Abstract

In mathematical optimization, the dual method provides a lower bound to the optimal solution of the primal minimization problem. Dual method is useful for discrete optimization (branch and bound) and separable programming (dual decomposition). Applying the dual method to optimally solve the primal problem, however, is not so straightforward due to potential issues such as dual function not having closed form and the existence of duality gap. This project creates a Python program that implements a general framework for solving dual problem empirically. The program is tested with different kinds of dual problems to illustrate its effectiveness as well as to provide more insights in dual properties and solving dual problems empirically.

1 Introduction

In mathematical optimization, a problem can be viewed from two different perspectives. The first perspective, known as the primal problem, expresses the problem as the minimization of the objective function with equality and inequality constraints. The second problem, known as the dual problem, expresses the problem as the maximization of a dual function that is the infimum of the Lagrangian equation of the original objective function, with the constraints that the Lagrangian multipliers are non-negative.

Dual problem has many interesting properties, such as always being concave (regardless of whether the primal problem is convex) and optimal solution to dual problem is a lower bound for the optimal solution to the primal problem. Such properties generate many applications of dual function. For example, when applying branch and bound to solve discrete optimization problem, at each node/sub-problem Lagrange relaxation that solves the dual problem can be applied to find the current lower bound that helps determining how to proceed.

Applying the dual method, however, is not so straightforward. The dual function may have worrying properties such as not having closed form or not being differentiable. In addition, even if the optimal solution to the dual problem is found, due to the existence of duality gap this does not guarantee

solving the primal problem optimally. As a result, dual method is usually applied on a case by case basis and a mixture of analytical and empirical approach is used to solve dual problem [Chen]. This project implements a Python program that solves the dual problem near optimally. By testing this program with different types of dual problems and analyzing the behaviour, this project report seeks to provide more insights in the properties of dual function and to empirically evaluate duality gap.

2 Background

Consider the general form of an optimization problem:

$$\begin{gathered} \min\ f(x) \\ \text{s.t. } h(x) = 0 \\ g(x) \leq 0 \\ x \in X \end{gathered}$$

The Lagrangian function of this problem can be written as:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda h(x) + \mu g(x)$$

The dual function is written as:

$$q(\lambda, \mu) = inf_{x \in X} \mathcal{L}(x, \lambda, \mu)$$

And the dual problem is:

$$\begin{gathered} \max\ q(\lambda, \mu) \\ \text{s.t. } \lambda \geq 0 \\ \mu \geq 0 \end{gathered}$$

Let f^* denotes the optimal solution to the primal problem and q^* denotes the optimal solution to the dual problem. According to the Weak Duality Theorem, we have $q^* \leq f^*$. In addition, the Weak Duality Theorem also states that the feasible set is convex (λ, μ such that q(λ, μ) $> -\infty$), and q is concave over this set. Furthermore, the dual problem has unified treatment for equality and inequality constraint in the Lagrangian function. Please refer to section 5 of [Bert] for further details.

3 A General Empirical Evaluation Framework and Implementation

3.1 Framework

Empirical evaluation of a dual problem has a two-fold consideration. First, we need to find the x the gives the infimum of the Lagrangian function. Then, we need to find the multipliers that maximize

the function from the first step. Note that x that gives the infimum of Lagrangian function may not be a constant vector, but may be a function of the multiplier. As a result, it is not possible to solve the dual function as a single maximization problem.

Therefore, a general framework for solving the dual function has an outer loop that searches the multiplier that maximizes the function, and an inner loop that searches for x that minimizes the function with fixed multiplier, as shown in the pseudo code below:

```
def solve_dual_function(multiplier, x):

    // ending criteria can be global maximum is found, after certain iteration, etc.
    while ending criteria not met:

        // multiple way to do this: ascending, random perturbation, etc.
        update multiplier

        // use unconstrained global optimization
        x = find x that minimizes L(x, multiplier)

    return multiplier, x
```

In each run of the outer loop, we get a new multiplier. For each multiplier value, we find x that would minimize the function with this multiplier in the inner loop. In other words, each iteration we find a point on q*(x, multiplier). Note that while the dual function is concave, it is not necessarily smooth. For example, after a certain point $q(\lambda, \mu)$ is $-\infty$. See experimental results for some examples.

3.2 Implementation

For both the inner loop and outer loop, the program uses scipy.optimize.basinhopping(). According to the official documentation, basinhopping() finds the global minimum using the basin-hopping algorithm. At each iteration, the algorithm finds a "random perturbation" of the variable (within a range that is controlled by the stepsize variable). It then (uses designated solver) to locally minimize the problem. Based on the result, it will accept or reject the updated variable. See the official documentation at [Basin] for more details about this function.

The inner loop of the program uses a constant multiplier value and applies basinhopping() to solve for value of x that minimizes the Lagrangian function. Like the previous section points out, this operation finds $q^*(multiplier)$ for a given multiplier value, such that $q^* \geq q^*(multiplier)$. It then updates x and return the current optimal point given the multiplier. See an example below.

```
// find global minimum
minimizer_kwargs = {"method": "SLSQP", "bounds" : bounds}
function = lambda x : (x[0] + x[1]) + multiplier[0] * (x[0]*x[1] - 1)
```

```
inner_result = basinhopping(function, [0, 0], minimizer_kwargs=minimizer_kwargs)

x_1 = inner_result.x[0]
x_2 = inner_result.x[1]

// the outer loop looks for maximization, so add a minus sign
return -(x_1 + x_2 + multiplier[0] * (x_1 * x_2 - 1))
```

The outer function will perform the operations mentioned above. For each iteration it will do the random perturbation to obtain a new multiplier, which is inputted to the inner function to find $q^*(multiplier)$. After getting that value, the outer function will update its status and start the next iteration.

```
result_outer = basinhopping(inner_function, [3], niter=100, minimizer_kwargs=minimizer_kwargs)
```

4 Experimental results

Multiple dual problems with different pattern is solved using the implementation described above. In most cases the program solves the problem optimally, though there are some special situations that are discussed below.

4.1 Example 1

Consider the following optimization problem (problem comes from CSE543T at Washington University in St. Louis taught by Professor Chen, FL21):

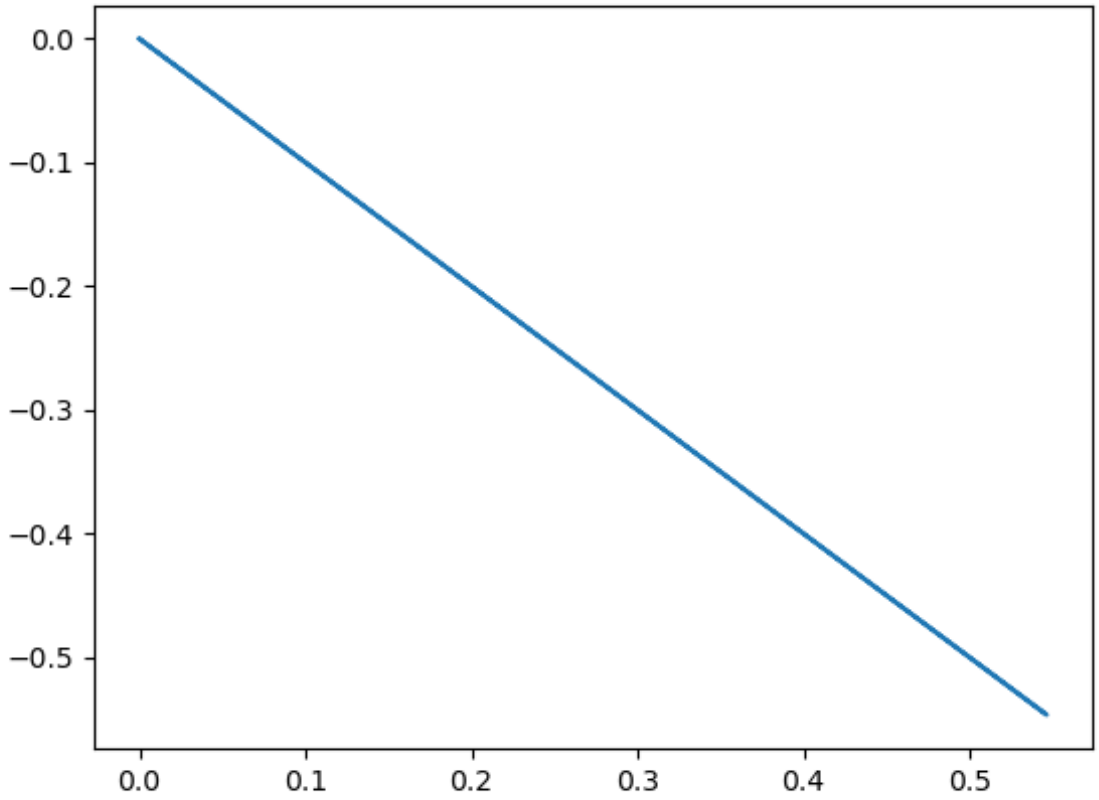
$$\begin{gathered} \min\ x_1 + x_2 \\ \text{s.t. } (x_1 * x_2) - 1 = 0 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{gathered}$$

The corresponding dual problem is:

$$\begin{gathered} \max\ inf_{x_1 \geq 0, x_2 \geq 0}\ x_1 + x_2 + \lambda((x_1 * x_2) - 1) \\ \text{s.t. } \lambda \geq 0 \end{gathered}$$

It is not too hard to observe from the dual problem that when $\lambda \geq 0$, the infimum of the Lagrangian function is $-\lambda$, and when $\lambda < 0$, the infimum is $-\infty$ by making x arbitrarily large.

The program solves the problem optimally by outputting $q^*(\lambda) = 0$ and $\lambda^* = 0$. In addition, by plotting $q^*(\lambda)$ vs λ at each iteration, it seems that the problem correctly captures the infimum of the Lagrangian function by having $q(\lambda) = -\lambda$.



Notably, if the starting point of λ is set to a negative value, in the next iteration it will be immediately adjusted to positive due to the bound $\lambda \geq 0$. Furthermore, note that the original problem has an optimal solution of $f^* = 2$, while the optimal solution to the dual problem is $q^* = 0$. There is a duality gap of $f^* - q^* = 2 - 0 = 2$. See README in the software directory to see how to duplicate the result.

4.2 Example 2

Consider the following optimization problem (problem comes from CSE543T at Washington University in St. Louis taught by Professor Chen, FL21):

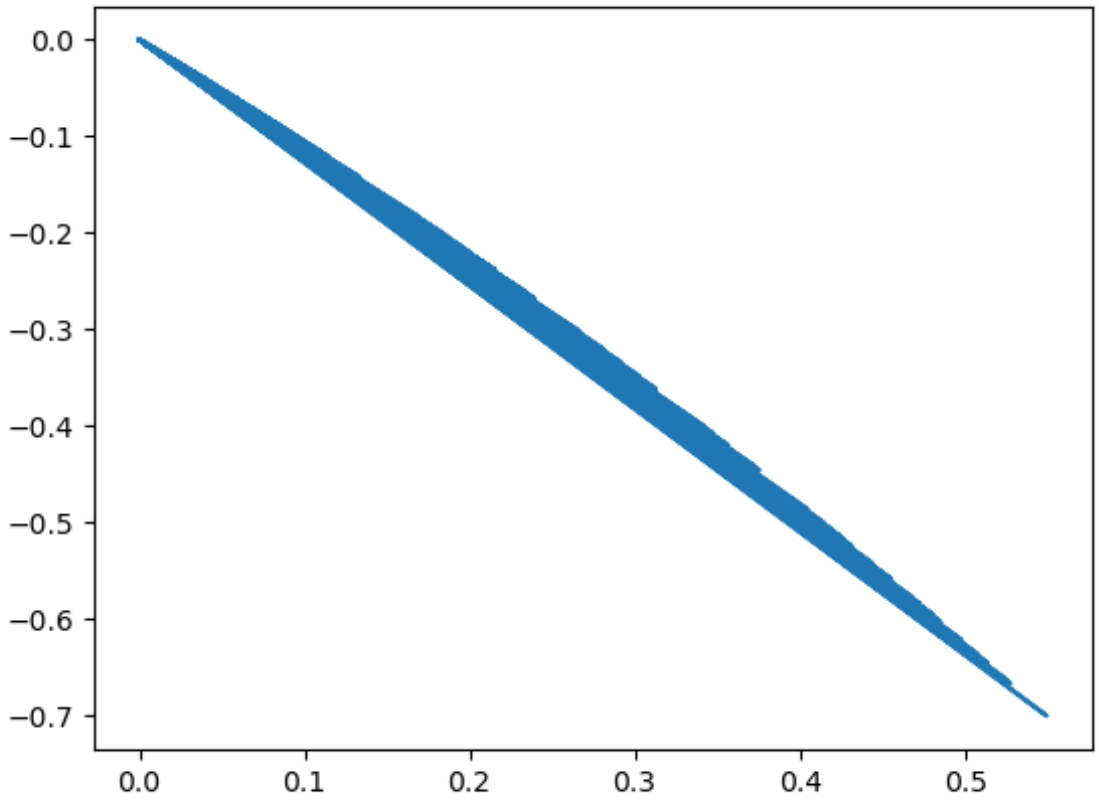
$$\begin{gathered} \min \ \tfrac{1}{2}(x_1^2 + x_2^2) \\ \text{s.t. } x_1 - 1 \leq 0 \\ x \in X = R^2 \end{gathered}$$

The corresponding dual problem is:

$$\max\ inf_{x \in X = R^2}\ \frac{1}{2}(x_1^2 + x_2^2) + \mu(x_1 - 1)$$
$$\text{s.t. } \mu \geq 0$$

The infimum of the Lagrangian function in this problem is not as straight forward as the one in the last problem largely due to x being unbounded. Solving the Lagrangian function analytically generates $q(\mu) = -\frac{1}{2}\mu^2 - \mu$. Therefore for $\mu \geq 0$ the $q^* = 0$, $\mu* = 2$

Again, The program solves the problem optimally by outputting $q^*(\lambda) = 0$ and $\mu^* = 0$. Though not drawn very well, the plotted μ and $q(\mu)$ does reflect the quadratic dual function when values are checked against each other.



This time, observe that there is no duality gap as $f^* = q^* = 0$.

4.3 Example 3

Consider the following optimization problem (problem comes from CSE543T at Washington University in St. Louis taught by Professor Chen, FL21):

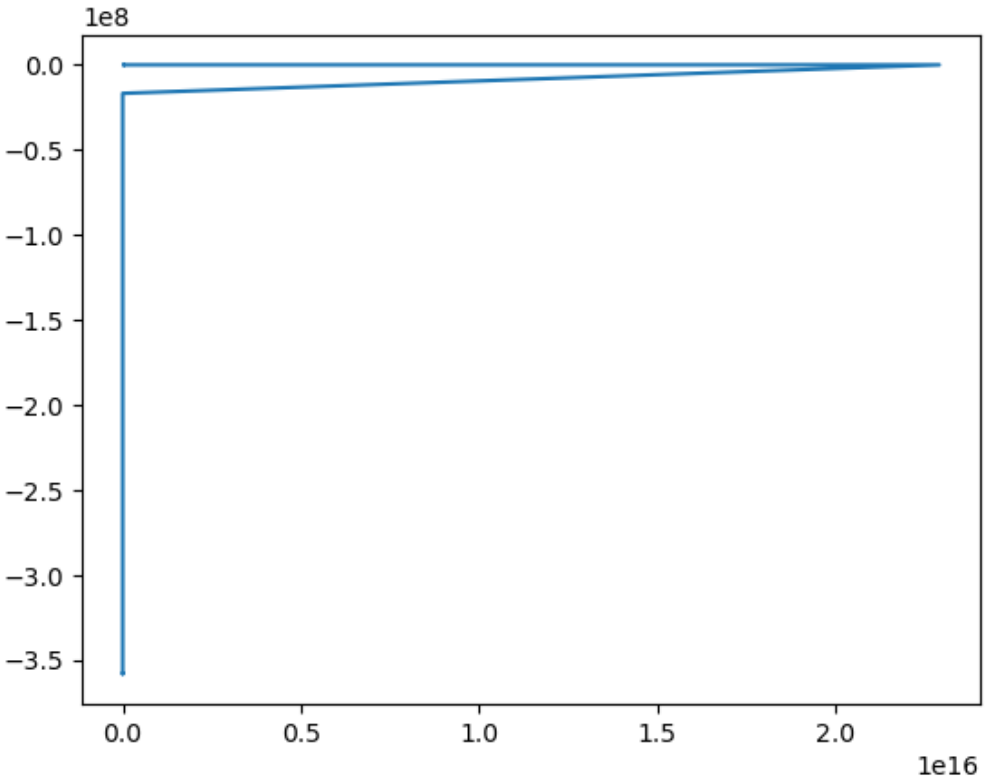
$$\begin{gathered}\min x \\ \text{s.t. } x^2 \leq 0 \\ x \in X = R^2\end{gathered}$$

The corresponding dual problem is:

$$\begin{gathered}\max inf_{x \in X = R^2}\ x + \mu(x^2) \\ \text{s.t. } \mu \geq 0\end{gathered}$$

This is a special dual problem whose optimal point is $\mu^* = \infty$ and $q^*(\mu) = -\frac{1}{4\mu} = 0$ when $\mu > 0$. The empirical evaluation was not able to output infinity as an answer. It does output a pretty large number ($1e8$ and $1e16$ as shown in the plot) that would make $q(\mu)$ very close or equals to 0.

Again, the plotted dual function does capture such behaviour. Note that some data points on the plot does not seem to correctly reflect the $q*$ value. By printing out the μ and q value at each iteration, it was found that μ value increases so quickly such that there are no data points for $\mu = 0.5, 1.2$, etc.



5 Discussions and Summary

As illustrated in the section above, for different dual problem, even special ones, the program has a reasonable behaviour and generate the optimal dual function that corresponds to the analytical analysis. This demonstrates the effectiveness of the program and the general solving framework described above. However, one of the problems is the slow speed solving the problem. For single, low dimension problem tested, the program can take as long as a few minutes (see test cases in python file).

There are some potential way to speed up the process at the sacrifice of preciseness. The inner loop would still use unconstrained global minimum solver, but the outer loop can apply a method that makes a better "guess" updating the multiplier at each iteration. For example, it may evaluate the slope of $q(\mu)$ and use a calculated or constant step size to determine the next multiplier value (resembling the descent method). With some prior analytical understanding of the dual problem, it is also possible to find some $q(\lambda, \mu)$, and fit them to a model (e.g. linear, quadratic, etc.) to estimate the dual function.

6 References

[Bert] Dimitri P. Bertsekas, "Nonlinear Programming", second addition, Athena Scientific.

[Chen] Yixin Chen, "CSE543 Class Notes", https://www.cse.wustl.edu/ yixin.chen/CSE543/notes

[Basin] Documents on scipy.optimize.basinhopping,
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html