ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis Econometrics and Operations Research

Differentially Private Convex Optimization with Piecewise Affine Objectives Using Projections and Average Subgradients

Name student: Niels Ota

Student ID number: 425442

Supervisor: prof. dr. S.I. Birbil

Second assessor: prof. C.U. Cakmak

July 8, 2019

Abstract

Han et al. introduced an iterative, differentially private method for solving convex constrained optimization problems with piecewise affine objectives: the differentially private subgradient method. In this paper, we propose extensions to this method to guarantee solutions stay within the feasible region and introduce a hyperparameter (γ) to allow the optimization algorithm to take an average subgradient (the differentially private average subgradient method). We find evidence that tuning γ and using projections can increase performance.

Contents

1 Introduction

Data is a hot topic. Data allows individuals, companies and countries to effectively manage their assets. Census bureaus over the whole world collect information about the population they are tasked with observing and disclose aggregate information through statistics. Companies conduct surveys to gain insights into the type of person that buys their product or uses their service. Individuals conducting empirical studies often require people to participate in their research to validate hypotheses. In short, data is a crucial part of decision making.

However, data requires information about individuals. Individuals might be reluctant to share information in fear of other parties using their data to deduce information about them they did not intend to share. For example, take a household using advanced metering infrastructures; meaning that the utility company can collect near real-time power consumption. If this single household drives an electric vehicle, zero demand from a power station could indicate that the home owner is away from home, which is interesting information for thieves. In short, individuals are exposed to the risk that potential eavesdroppers are attempting to learn information they want to hide.

An intuitive - yet naive - solution to protecting the privacy of individual users is anonymizing the dataset. Unfortunately, this is no guarantee for privacy due to the fact that side information is available [1]. The lack of privacy guarantees through anonymization was demonstrated through the identification of Netflix subscribers in the anonymized Netflix prize dataset through linkage with the Internet Movie Database [1]. This demonstrated the need for a rigorous method for preserving the privacy of individuals.

In 2006, Dwork and her collaborators introduced differential privacy as a manner of guaranteeing privacy [2]. Differential privacy translates the privacy of an individual to the effect that individual has on any random mechanisms working on that dataset. Roughly speaking, differential privacy ensures that the removal or addition of a single user (a row in the database) does not effect the outcome of an analysis on that dataset beyond a specified threshold.

In its original setting [2], differential privacy assumes a trustworthy curator gathers information from individuals (respondents) and uses this information to release aggregate information to the public. External queries are made to the curator by outside parties (possible adversaries). The curator is tasked with answering these queries in a way that protects respondents from adversaries trying to learn information about individual respondents. The curator does this by ensuring differential privacy.

An interesting point of research is how useful the results of queries are. In systems operation, user data is leveraged to make decisions that optimize the performance of the system. Hence, the query is the solution to an optimization problem. In this setting, user data corresponds to the parameters of the objective function or its constraints. Han et al. [3] introduced and analyzed the differentially private subgradient method (DPSM); a differentialy private version of the regular subgradient method. They applied this method to a piecewise affine objective with a hypercube feasible region. In this setting, Han et al. found that the DPSM performed better than other differentially private methods; like the Laplace- and exponential mechanism. Although these results were promising, they were attained under restrictive assumptions. The goal of this paper was to replicate their findings, and relax a number of their assumption about the shape of the feasible region and investigate whether their findings continue to hold. Also, we add projections to the DPSM, ensuring feasible solutions. Additionally, we introduce the differentially private average subgradient method (DPASM), which uses an average subgradient whist preserving differential privacy.

The mayor result of this paper is that the DPSM continues to perform better than the differentially private Laplace mechanism and the exponential mechanism for feasible regions in a variety of shapes. We altered the DPSM presented in [3] to guarantee feasible solutions, and motivated this modification. Furthermore, we found that the differentially private average subgradient method yielded results with lower suboptimality when the dimensionality of the problem increased and γ was large.

The next section discusses the position of the reference paper [3] and this paper in the field of differential privacy.

2 Literature Review

Dwork and her collaborators introduced the term differential privacy in 2006 [2]. It provided a mathematically rigorous guarantee for privacy, and consequently received a lot of attention. Dwork proposed the Laplace mechanism to guarantee differential privacy. The Laplace mechanism perturbs the query by adding noise. A problem with the Laplace mechanism is that the range of the output has to be the real numbers (if the possible outcomes are dog or cat, you can not add noise). Consequently, the need for a differentialy private method that allowed non-real ranges arose.

In 2007, McSherry and Talwar introduced the exponential mechanism to solve the above mentioned problem [4]. Rather than adding noise, this method assigns a score to every possible outcome of a query. The closer this outcome is to the true outcome, the higher the score is. An outcome is then chosen such that the probability of selecting it is proportional to its score. The range therefore no longer has to be real numbers. All that the exponential mechanism requires is a function that maps every outcome to a score. The exponential mechanism widened the array of differential privacy ensuring tools, which led to a wider array of differentially private applications.

An example of such an application was the use of differential privacy to optimization problems. Nozari et al. [5], amongst others, studied multi-agent differentially private distributed constrained optimization where the objective function of each agent is kept private and the agents aim to minimize the sum of their objective functions. Kusher et al. [6] applied differential privacy to Bayesian optimization - a powerful tool in machine learning - to ensure that no personal information can be inferred from Bayesian optimization output.

Han et al. [3] studied the class of convex optimization problems whose objective function is piecewise affine, with linear constraints. This problem surfaces in norm optimization and resource allocation problems [3]. They proposed an iterative method called the differentially private subgradient method (DPSM) to privately solve this class of problems. Their main finding was that by selecting a subgradient in a differentially private manner, the subgradient method becomes differentially private. Han et al. [3] reported increased performance compared to methods mentioned above. However, these findings were done under restrictive assumptions about the feasible region. Additionally, their findings relied on the generation of function parameters by a normal distribution, whose mean and standard deviation were left undisclosed. This makes replicating their findings impossible. Furthermore, the DPSM had no guarantees of providing a solution that lied within the feasible region.

In this paper we replicated and critically evaluated the finding made in [3] and tested the applicability of the DPSM to a larger variety feasible regions; Including spheres, linear equality constraints and linear convex polytopes. In doing so, other parties know when it is appropriate to use the DPSM and when it is not. Additionally, we present an algorithm guaranteeing a feasible solution, which has obvious benefits for any party using the DPSM. Finally, we introduce the differentially private average subgradient method (DPASM). The next section elaborates on the mathematical definition of differential privacy and introduces the problem statements.

3 Problem Statement

Before introducing the problem, we will now give a brief introduction to differential privacy. Differential privacy concerns itself with masking changes in the database caused by a single user. Changes in the database should be attributable to things other than a user joining or leaving the database, hence giving the user plausible deniability. Thus, the idea behind differential privacy is to equate ones privacy to changes caused by joining or leaving a database.

Consider the universe of all possible databases as $\mathcal{D}$. The points of interest are queries that take a database as input and map this database to a target domain $\mathcal{Q}$. Denote this mapping as $q : \mathcal{D} \to \mathcal{Q}$. If $q(\mathcal{D})$ is readily available, adversaries might infer properties about individuals in $\mathcal{D}$ that should remain private. A mechanism $M(\mathcal{D})$ is needed in order to preserve privacy in the sense that no single user can influence the outcome of $M(\mathcal{D})$ beyond a quantifiable limit.

In order to define this limit, the definition of an adjacent database is needed. Adjacent databases are databases that differ on a single user being present or not present in that database; databases that differ in terms of one row. With this tool, differential privacy is defined.

Definition 1. A randomized mechanism $M : \mathcal{D} \to \mathcal{Q}$ preserves ε-differential privacy if for all $R \subseteq \mathcal{Q}$ and all pairs of adjacent databases $D \in \mathcal{D}$ and $D' \in \mathcal{D}$:

$$P(M((D) \in R) \leq e^{\varepsilon} P(M(D) \in R). \tag{1}$$

Relation (1) guarantees that the probability of an outcome of the random mechanism differs by at most a factor $e^{\varepsilon} \approx 1 + \varepsilon$ for adjacent databases. Hence, lower value of ε mean greater levels of privacy.

Having established a formal representation of differential privacy, we will present the class of minimization problems studied in [3] and this paper.

3.1 Reference Paper Problem Statement

In this paper and [3], the following class of functions was minimized:

$$f(x) = \max_{i=1,2,\ldots,m} \{a_i^{\intercal} x + b_i\} \tag{2}$$

The function above is convex and piecewise affine and maps from $\mathbb{R}^d$ to $\mathbb{R}$ ($f : \mathbb{R}^d \to \mathbb{R}$). The optimal value for the objective was constrained to lay within a convex hypercube [3], so the optimization problem is defined as follows:

$$\min_x f(x) \quad s.t. \quad -c \leq x_i \leq c \quad \forall i \tag{3}$$

As in [3], $\{b_i\}_{i=1}^m$ was the set of user information that remained private. Both $\{a_i\}_{i=1}^m$ and $\mathcal{P}$ were public information. Thus, the database was: $D = \{b_i\}_{i=1}^m$. In this setting databases are defined as adjacent if the following holds:

$$\max_{i \in \{1,2,\ldots,m\}} |b_i - b_i'| \leq b_{max} \tag{4}$$

With the above equations, the first research problem is formulated as follows:

Problem 1. For problems of the form (3), find a mechanism M that outputs a approximate optimal solution that preserves ϵ-differential privacy under the adjacency relation (4).

Han et al. [3] used the DPSM to find a solution to Problem 1, and compared this to a number of other differentially private methods. They found that the DPSM substantially outperforms these methods. However, we saw no reason to assume that results found on a hypercube would generalize to feasible regions of any shape. Hence we investigated whether there is evidence that this generalisation indeed holds. The following section introduces this research question.

3.2 Problem Statement Extension

To test whether the DPSM outperforms other differentially private methods when the feasible region is another shapes, also considered the following formulations.

Problems where the feasible region is a sphere:

$$\min_x f(x) \quad s.t. \quad \|x\|_2 \leq c \tag{5}$$

Problems with a set of equality constraints:

$$\min_x f(x) \quad s.t. \quad Cx = d \tag{6}$$

And finally problems with a set of inequality constraints:

$$\min_x f(x) \quad s.t. \quad Cx \leq d \tag{7}$$

Problem 2. For problems of the form (5), (6) and (7), does the differentially private subgradient method continue to outperform other differentially private optimization methods?

4 Useful differential privacy tools

This section reviews several necessary tools in differential privacy. These tools include the Laplace mechanism, the exponential mechanism, the post-processing rule, and the composition of private mechanisms.

4.1 Laplace Mechanism

The Laplace mechanism [2] adds calibrated noise to components of a query. It adds as little noise as possible whilst guaranteeing ϵ-differential privacy. If queries on adjacent databases yield distant results (high sensitivity), more noise is added to mask these changes. Vice versa, the same holds.

The sensitivity measures the maximum change in the outcome of queries for adjacent databases. Suppose the sensitivity of query q is defines as

$$\Delta = \max_{D,D'} \left\| q(D) - q(D') \right\|_\infty \tag{8}$$

and is bounded. Dwork and her collaborators found that adding i.i.d. Laplace noise $Lap(d\Delta/\epsilon)$ to every component of q yields ϵ-differential privacy [2].

Han et al. [3] present a similar mechanism which requires less noise by noting that the l_2 sensitivity of Δ_2 is

$$\Delta_2 = \max_{D,D'} \left\| q(D) - q(D') \right\|_2 \leq \sqrt{d}\Delta \tag{9}$$

and hence bounded.

Theorem 1. ([3][2]) For a given query q, let $\Delta_2 = \max_{D,D'} \left\| q(D) - q(D') \right\|_2$ be the l_2 sensitivity of q. Then the mechanism $M(D) = q(D) + w$, where w is a random vector whose probability distribution is proportional to $\exp(\epsilon \left\| w \right\|_2 / \Delta_2)$, preserves ϵ differential privacy.

Where Dwork in [2] adds i.i.d. noise to each component of q, Han et al. [3] add an entire vector of noise in one step. They call it the vector-Laplace mechanism due to its close resemblance to Laplace mechanism. Unfortunately, both mechanisms share the same shortcoming: the pair can only be used when the range of the query is $\mathbb{R}$. The next section will discuss a mechanism that allows the range of q to be any type of set.

4.2 Exponential mechanism

The exponential mechanism preserves differential privacy and allows the range of the query to be any set [4]. This mechanism scores all possible candidate queries for a given database D. The score of a database paired with a candidate query is measured by a utility function $u : \mathcal{Q} \times \mathcal{D} \to \mathbb{R}$, where $\mathcal{Q}$ is the range of a query on database D. If a query R is preferred to R', then $u(D,R) > u(D,R')$. The exponential mechanism selects any candidate query with a probability proportional to its score. The mechanism $M_E(D;u)$ is hence defined as follows:

Definition 2. For any function $u : \mathcal{Q} \times \mathcal{D} \to \mathbb{R}$ with $R \subseteq \mathcal{Q}$ and $D \in \mathcal{D}$

$$\begin{aligned} M_E(D;u) := \text{Select candidate query } R \text{ with probability} \\ \text{proportional to } \exp(\frac{\epsilon u(R,D)}{2\Delta_u}) \end{aligned} \tag{10}$$

Here for any scoring function u, Δ_u describes sensitivity of the scoring function.

$$\Delta_u := \max_{R} \max_{D,D' \,:\, \mathrm{Adj}(D,D')} |u(R,\mathcal{D}) - u(R,\mathcal{D}')| \tag{11}$$

There are three terms in the exponent of (10). If ϵ is infinite, in (10) the query R with the largest score will be selected with certainty. Hence, there is no privacy. If ϵ is zero (complete privacy), all queries R could be chosen with equal probability.

Theorem 2. (McSherry and Talwar [4]) The exponential mechanism $M_E(D; u)$ guarantees ϵ-differential privacy.

In short, the exponential mechanism allows the selection of any query based on the score of those queries. Therefore the range of the query $\mathcal{Q}$ no longer has to be $\mathbb{R}$.

4.3 Useful Theorems

The following two theorems are useful tools in differential privacy. They facilitate the construction of new differentially private algorithms based on existing ones. They are presented without proofs in this paper. We recommend readers interested in the proofs to study [2] and [7].

Theorem 3. Post-processing. [2] If a mechanism $M : \mathcal{D} \to \mathcal{Q}$ preserves ϵ-differential privacy. Then for any function f, the functional composition $f \circ M$ also preserves ϵ-differential privacy.

Theorem 4. Sequential composition [7]. Suppose a mechanism $M_1 : \mathcal{D} \to \mathcal{Q}$ preserves ϵ_1-differential privacy, and another mechanism $M_2 : \mathcal{D} \to \mathcal{Q}$ preserves ϵ_2-differential privacy. The mechanism $M(\mathcal{D}) = (M_1(\mathcal{D}), M_2(\mathcal{D}))$ preserves $(\epsilon_1 + \epsilon_2)$-differential privacy.

The Laplace mechanism, the exponential mechanism, the post-processing theorem, and the sequential composition theorem form the theoretical foundation of the differentially private optimization algorithms that we will discuss in the next sections.

5 Methodology

This section presents the various (differentially private) approaches to optimizing the class of minimization problems discussed in the chapter 3. First we discuss several non-private minimization algorithms. Then we review the four mechanisms that obtain a differentially private solution discussed in [3]. Finally, we discuss their shortcomings and suggest several improvements in the subsequent section.

5.1 Optimization without privacy

5.1.1 Subgradient method

Disregarding privacy, problems of the form (3) can be solved using the subgradient method. This method converges if the best solution is stored whilst running the method. This is key since the subgradient is not necessarily a descent direction. Formally, g is a subgradient of f at x_0 if the following relationship holds for all x in the domain of f:

$$f(x) \geq f(x_0) + g^\intercal(x - x_0) \tag{12}$$

Let $k \in \{i =, 1, 2, ..., m\}$ be the index of the active affine function, such that:

$$a_k^\intercal x_0 + b_k = \max_{i=1,2,...,m} \{a_i^\intercal x_0 + b_i\} \tag{13}$$

Clearly, a_k is the subgradient of f at x_0. However (13) shows that computing the subgradient requires access to private information: $\{b_i\}_{i=1}^m$. A later section will discuss how to privately compute a subgradient.

5.1.2 Solving as Linear Program

Before moving on to private optimization, notice that (3) can be solved exactly as a linear program using the following formation:

$$\begin{aligned} &\textit{minimize} && z \\ &\textit{s.t.} && z \geq a_i^\intercal x_0 + b_i \quad \forall i \\ & && -c \leq x_i \leq c \quad \forall i \end{aligned}$$

Solving the above problem gives an exact solution, and consequently provides insight into the performance of the other minimization algorithms.

5.2 Laplace solution

The Laplace mechanism can be used to perturb a query with noise. This query can either be the data, or the solution itself. Both will be discussed, starting with the pertubation of the data.

5.2.1 Laplace mechanism acting on the problem data

Adding Laplace noise to the data (b) and solving (3) guarantees differential privacy. This means adding a vector of noise to the database $D = \{b_i\}_{i=1}^m$ using the vector Laplace mechanism described in Theorem 1. Solving the problem once the problem has been privatized is post-processing, hence the obtained solution is private due to Theorem 3.

Theorem 5. The mechanism that outputs $M_P(b) = b + w_P$, where w_P is drawn from the probability density function proportional to $\exp(-\epsilon \|w_P\|_2 / \sqrt{m} b_{max})$, is ϵ-differentially private.

5.2.2 Laplace mechanism acting on the problem solution

The second way of preserving ϵ-differential privacy is to first solve (3), and then apply a vector of Laplace noise to the solution $x_{opt} : M_S(D) = x_{opt}(D) + w_S$. Where w_S is drawn from a distribution proportional to $\exp(-\epsilon \|w_P\|_2 / \sqrt{d}\Delta)$. Here Δ is the sensitivity of the optimal solution. It is generally difficult to analyze how the optimal solution responds to changes in the database [3]. However, for certain shapes (cubes) the diameter of the feasible region provides an upper bound on Δ. This fact can be used to add Laplace noise to x_{opt}. For other feasible regions, the upper bound on Δ can be computed by finding the minimum-volume (Löwner-John) ellipsoid inscribing the feasible region $\mathcal{P}$, and subsequently using its longest axis as the upper bound.

5.3 Exponential Mechanism

The third private optimization algorithm is the exponential mechanism. In [3], Han et al. showed that if the negative function $(-f)$ is used as a scoring function (u), the sensitivity with respect to u (Δ_u) is equal to b_{max}. We refer to [3] for the proof. This leads to the following:

Theorem 6. The mechanism M_E that outputs $\bar{x}_{opt}$, according to the probability density function

$$\frac{\exp(\frac{\epsilon u(\bar{x}_{opt}, D)}{2 b_{max}})}{\int_{x \in \mathcal{P}} \exp(\frac{\epsilon u(\bar{x}_{opt}, D)}{2 b_{max}}) dx}$$

is ϵ-differentially private.

5.4 Differentially private subgradient method

The fourth private optimization algorithm is the differentially private subgradient method (DPSM) and is the main result in [3]. In essence, it is identical to the regular subgradient method. However, in order to preserve privacy, the gradient is computed privately.

At any given point x_0 the gradient is the magnitude of the slope of the affine function that is active in $f(x) = \underset{i=1,\ldots,m}{max} \{a_i^\intercal x_0 + b_i\}$. The set of all possible gradients therefore is $\{a_i\}_{i=1}^m$. Han et al. suggested choosing a gradient by selecting the index i using the exponential mechanism. The exponential mechanism requires a scoring function u. It it desirable to assign a higher score to indices i that attain relatively higher values in $a_i^\intercal x_0 + b_i$. Making it more likely to select the index corresponding to the true gradient at x_0. As such the proposed in [3] was $u_{sub}(i; x_0, D) = a_i^\intercal x_0 + b_i$. Readers should note that this is a function of i. The sensitivity is then

$$\begin{aligned}\Delta_u &= \max_{i=1,\ldots,m} \max_{\text{Adj}(D,D')} |u(i; x_0, D) - u(i; x_0, D')| \\ &= b_{max}\end{aligned} \tag{14}$$

We refer to [3] for a proof. Algorithm 1 shows how to privately compute a subgradient.

Algorithm 1: ϵ-differentially private subgradient.

```
1  Choose the scoring function $u : \{1, 2, ..., m\} \to \mathbb{R}$ as
       $u_{sub}(i; x_0, D) = a_i^\intercal x_0 + b_i$
2  Select the index $i^*$ using the exponential mechanism:
       $P(i^* = i) \propto \exp(\frac{\epsilon u_{sub}(i; x_0, D)}{2b_{max}})$
3  Output $a_i^*$ as the private approximate subgradient at $x_0$
```

The DPSM in [3] is constructed by replacing the subgradient in the regular subgradient method by a differentially private subgradient using Algorithm 1. The result is Algorithm 2.

Algorithm 2: ϵ-differentially private subgradient method.

```
1  Choose the number of iterations $k$, step sizes $\{\alpha_i\}_{i=1}^k$ and $x^{(1)} \in \mathcal{P}$
2  For $i = 1, ..., k$ repeat the following steps
       1. Obtain an $(\epsilon/k)$-private subgradient $g^{(i)}$ using Algorithm 1;
       2. Update $x^{(i+1)} \leftarrow x^{(i)} - \alpha_i g^{(i)}$
3  Output $x^{(k+1)}$ as the solution
```

If the subgradient used in Algorithm 2 is computed using Algorithm 1, Algorithm 1 (the modified subgradient method) is differentially private. This can be shown using the sequential composition theorem; if each iteration of Algorithm 1 preserves (ϵ/k)-differential privacy and the total number of iterations is k, Algorithm 2 preserves (ϵ)-differential privacy.

5.5 Limitations differentially private subgradient method

Although Han et al. [3] presented evidence that the DPSM outperforms other well-known differentially private optimization techniques, their findings were done under idealized circumstances. The limitations with their findings are:

1. In step 2 of Algorithm 2, x is updated using the subgradient method update rule. However, there is no guarantee that x will remain in the feasible region $\mathcal{P}$ once this step has been taken. As an extension to[3], we expanded the DPSM so it guarantees feasible solutions.

2. Han et al. [3] did not test the DPSM of feasible regions other than a hypercube. To validate the effectiveness of the DPSM, we included an extension that tests the DPSM on other feasible regions.

3. Han et al. [3] compared the performance of the DPSM to the regular subgradient method for (3). However, as shown in section 5.1.2, there is an exact solution to problems of this form. Therefore, comparing the DPSM to the exact solution offers a better basis for comparison. In this paper, we included the exact solution alongside the subgradient method solution.

The next chapter will discuss extensions aimed at overcoming the aforementioned shortcomings of the finding made in [3].

6 Extensions

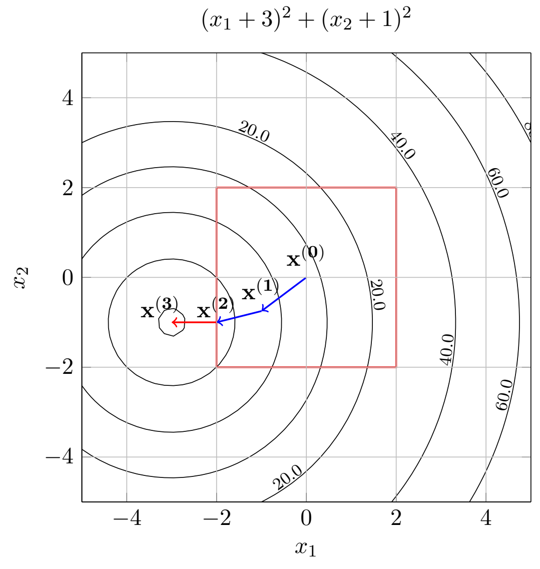
This chapter presents extensions to [3]. A number of critical remarks made in previous chapters warrant questions regarding generalization, relevance and feasibility of the differentially private subgradient method. The extensions are aimed at testing the DPSM with regards to these critical remarks.

6.1 Alternative feasible regions

A problematic assumption made by Han et al. [3] was the assumption that results found using a hypercube centered at the origin as the feasible region would generalize to convex feasible regions of any shape. In other words, assuming that the DPSM will work for any problem, given that it works on a simple problem. In section 3.2 we presented a number of other feasible regions to test the DPSM on. The results of these findings are discussed in the next chapter.

6.2 Private projected gradient descent

Although Han et al. [3] use a hypercube as the feasible region, there is no guarantee that x remains in the hypercube after each iteration of the DPSM without a projection back onto $\mathcal{P}$. The figure below illustrates this. It shows the contour lines of a sphere, with a square (2 dimensional cube) constraint. The arrows correspond to iterations of the gradient descent algorithm (f is a sphere, hence there is no subgradient). In the last step, the gradient descent algorithm steps towards the minimum, which lies beyond the boundaries of the constraint.



The figure above shows that Algorithm 2 does not guarantee that optimal solutions lie within the feasible region. To guarantee the DPSM is applicable to a wider range of objective functions and constraint sets, this guarantee should be added to the optimization algorithm. Doing so yields Algorithm 3.

Algorithm 3: ϵ-differentially private projected subgradient method.

```
1  Choose the number of iterations $k$, step sizes $\{\alpha_i\}_{i=1}^k$ and $x^{(1)} \in \mathcal{P}$
2  For $i = 1, ..., k$ repeat the following steps
     1. Obtain an $(\epsilon/k)$-private subgradient $g^{(i)}$
     2. Update $y^{(i+1)} \leftarrow x^{(i)} - \alpha_i g^{(i)}$
     3. $x^{(i+1)} \leftarrow$ projection $y^{(i+1)}$ onto $\mathcal{P}$
3  Output $x^{(k+1)}$ as the solution
```

Combining Algorithm 2 and Algorithm 3 yields the DPSM used in the remainder or this paper, and guarantees that solutions are feasible. Algorithm 3 requires a projection back onto the feasible region $\mathcal{P}$. The method to accomplish this projection depends on the formulation of $\mathcal{P}$. For instance, if $\mathcal{P}$ is box shaped, then the projection can be accomplished in a single step (decreasing the magnitude of x_i to the limit of the box $\forall i$). However, if $\mathcal{P} = \{x \mid Cx \leq d\}$ then a minimization problem surfaces at every iteration of the DPSM.

6.3 Private average subgradient

This section presents the differentially private average subgradient (DPASM) method as a potential improvement on the DPSM. It privately samples γ subgradients, and uses its average as the subgradient. The hypothesized result is that the average of multiple (γ) private subgradients will be closer to the true subgradient than a single private subgradient. This would allow the DPASM to outperform the DPSM.

To calculate the required level of privacy for each sampled subgradient consider drawing not one, but two subgradients using Algorithm 1 (a^*_{i1} and a^*_{i2}), and taking the average as the outputted subgradient: $a^*_i = \frac{1}{2}(a^*_{i1} + a^*_{i2})$. If a^*_{i1} and a^*_{i2} are drawn with a privacy level equal to $\epsilon/2k$, then according to the sequential composition theorem, their average will maintain ($\epsilon/2k + \epsilon/2k = \epsilon/k$)-differential privacy. After k iterations of the DPSM, the net level of privacy will be ϵ. For generality, define the hyperparameter γ as the number of private subgradients drawn at each DPSM iteration i. Then the set of subgradients is $S = \{a^*_{ij}\}_{j=1}^{\gamma}$. In order to maintain ($\epsilon/k$)-differential privacy when calculating the mean of S, each $a^*_{ij} \in S$ should attain ($\epsilon/\gamma k$)-differential privacy.

Algorithm 4: ϵ-differentially private subgradient.

```
1  Choose the scoring function $u : \{1, 2, ..., m\} \to \mathbb{R}$ as
       $u_{sub}(i; x_0, D) = a_i^\intercal x + b_i$
2  For $j = 1, ..., \gamma$ select the index $i^*$ using the exponential mechanism:
       1. $P(i^* = i) \propto \exp(\frac{\epsilon u_{sub}(i; x_0, D)}{2\gamma b_{max}})$
       2. Save $a^*_{ij}$
3  Output $a^*_i = \frac{1}{\gamma}\sum_{j=1}^{\gamma} a^*_{ij}$ as the private approximate subgradient at $x_0$
```

If the computation of the subgradient required in Algorithm 3 is executed by Algorithm 4, and every subgradient in Algorithm 4 preserves ($\epsilon/\gamma k$)-differential privacy, Algorithm 3 will be ϵ-differentially private. This can be shown using the sequential composition theorem.

Algorithm 4 makes γ a trainable hyperparameter. This potentially makes Algorithm 4 more robust to changes in the objective function, or more complex feasible regions. We tested the DPASM on piecewise affine objectives for different amounts of variables in the objective function. The results are discussed in the next section.

7 Results

In the following section, the results from the simulations are presented. The section consists of two parts: the first part replicates and tests the findings done in the reference paper [3]. The second part discusses the results from the extensions presented in the previous section.

7.1 Results Replication

We attempted to use the same hyperparameters used in [3]. This meant using $\epsilon = 0.1$ (privacy level), attaining the expected objective value from different privacy-preserving algorithms using 1000 runs, and using a d-dimensional hypercube as the constraint set (with $diam(\mathcal{P}) = 2\sqrt{d}c$). However, this is as much information Han et al. gave about the context in which their finding were done. This is one of the main critiques we have on [3], considering research has to be replicable to establish its validity. In this paper the problem data $\{(a_i, b_i)\}_{i=1}^{m}$ were generated from i.i.d standard standard Gaussian distributions. Here m is the number of affine functions in the objectives. Unless explicitly mentioned, m is set equal to 20 and d is set equal to 5 ($m = 20$, $d = 5$). The random seed for all simulations was 4. The DPSM in this paper consists of combining Algorithm 2 and Algorithm 3 (Algorithm 1 does not guarantee feasible solutions).

The different differentially private optimization algorithms require sampling from a number of different distributions.

1. **Laplace mechanism**: Both algorithms involving Laplace noise require sampling from $\exp(\epsilon \|w\|_2 / \Delta_2)$, where Δ_2 differs depending on whether Laplace noise is being added to the data ($\sqrt{m}b_{max}$) or to the solution ($2\sqrt{d}c$). To efficiently draw w from a distribution proportional to $\exp(\epsilon \|w\|_2 / \Delta_2)$, its direction e and magnitude $\hat{w}$ were drawn separately. Here $\hat{w}$ follows a Gamma distribution $\Gamma(\lambda, d)$ and can be sampled from by drawing d i.i.d samples $w_1, w_2, ..., w_d$ and adding the result to make $\hat{w} = \sum_{i=1}^{d} w_i$. Then $\hat{e}$ was sampled from a d-dimensional multivariate Gaussian, after which it was normalized, and multiplied by the magnitude $\hat{w}$ to make w.

2. **Exponential mechanism**: The exponential mechanism required drawing samples proportional to a non-negative distribution (6). Markov chain Monte Carlo (MCMC) methods are usually used in order to draw from a multidimensional target distribution [8]. MCMC

draws samples from a multivariate distribution whose stationary distribution is the target distribution. In [3], a multivariate Gaussian is used as the proposal distribution. Due to the fact that the constraint set is a d-dimensional hypercube, the covariance matrix Σ is set to be isotropic, with its magnitude proportional to the size of the constraint set, so $\Sigma = \eta c I_{dxd}$ [3]. Where I_{dxd} is a d-dimensional identity matrix and $\eta = 0.1$. Each sample is generated by running 5000 MCMC steps, and then using the next sample that is accepted by the Markov chain (after 5000 steps the Markov chain is considered to have reached its stationary distribution). We used this same sampling method.

3. **DPSM iterations**: Han et al. [3] provide a theorem that shows there is a theoretical optimal number of iterations for a given level of privacy ϵ. However, this suboptimality bound is loose, and optimizing the number of iterations on the bound does not provide enough guidance for choosing a number of iterations. Figure 1 shows that after 1000 iterations, the regular subgradient method has nearly approached the exact solution. The DPSM (with projections onto the hypercube) also appears to have reached close to its minimum after 1000 iterations. Han et al. find slightly different results from the one shown in Figure 1. This could be due to the fact that different step-sizes were used, or the sampling of the hyperparameters was different. In this paper, we used step-sizes of $1/k^{0.51}$, with k the current iteration. In [3], Han et al. claim the objective value is robust to the number of iterations after 100 iterations. we found results that contradict this. Although the subgradient method is close to the exact solution after 100 iterations, the inclusion of the exact solution shows that the subgradient method continues to decrease. However, since after 100 iterations the regular subgradient is sufficiently close to the exact solution, and to directly compare our findings to those done in [3], we used 100 iterations in the remainder of the numerical experiments unless mentioned otherwise. After 1000 iterations, the subgradient method has nearly reached the exact lower bound and the DPSM is no longer improving.

The following two figures compare the performance of the DPSM to other privacy-preserving optimization algorithms as a function of the size of the hypercube (c) and the number of affine function in the objective (m).

The image shows that both Laplace algorithms are the worst performing differentially private algorithms. In this paper, x values were projected back onto the box (this was never explicitly mentioned in the reference paper [3]). The noise generated for both Laplace algorithms was larger than the diameter of the constraint set. Therefore, the solution always had to be projected back onto the constraint set. This caused the x value to always lay on the box, meaning how

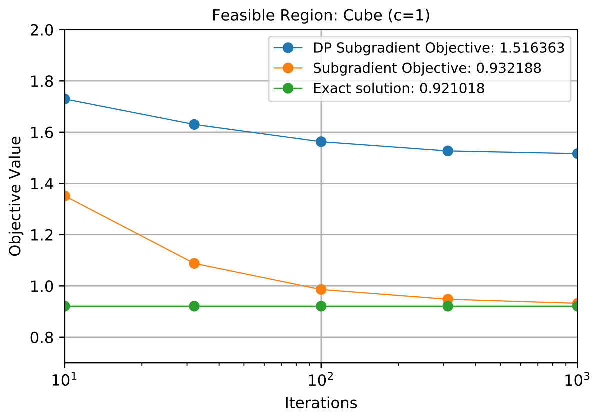
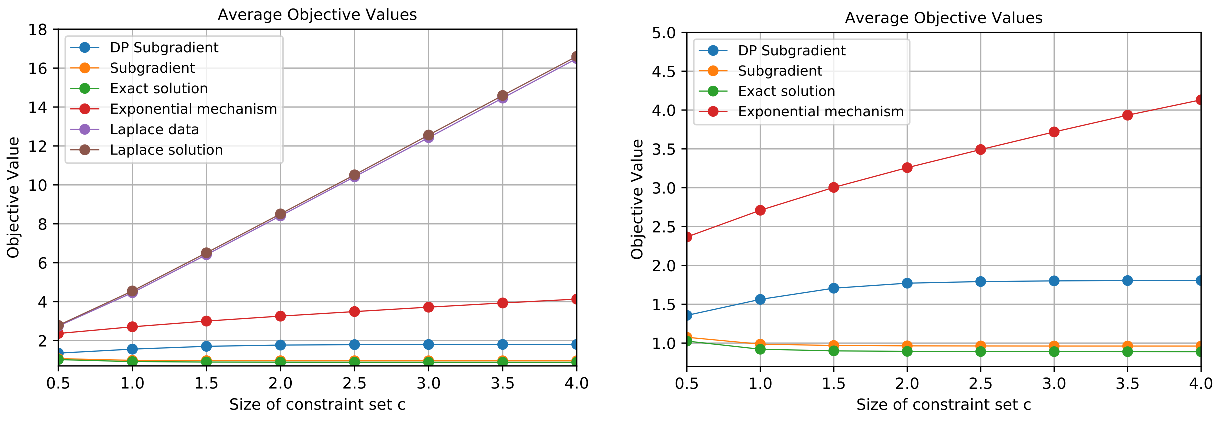


Figure 1: Expected objective value as function of k



(a) Including Laplace mechanisms

(b) Excluding Laplace mechanisms

Figure 2: Expected objective value as function of c

larger the box was, how further away the x value could be from the optimal solution, thus causing the Laplace mechanism to perform badly.

Figure 2 shows the DPSM performs better than the exponential mechanism. However, the inclusion of the exact solution shows that the suboptimality remains above 50%. Furthermore, the DPSM is less sensitive to the magnitude of c, whereas the suboptimality of solutions generated by the exponential mechanism show an almost linear increase with the size of the constraint set (although there is some marginal decrease with increases in c).

The result of increasing the number of affine function in the objective (m) yielded similar results as in [3]. Our results in Figure 2 differ on a few points. First of all, all the expected objective values increase monotonically when m increases. Given that all the affine functions contained in the objective for a given value of m are also contained in the objective when m increases, monotonic increase is in line with expectations. Furthermore, the distance between the different privacy preserving algorithms is larger compared to the results in [3]. This could be

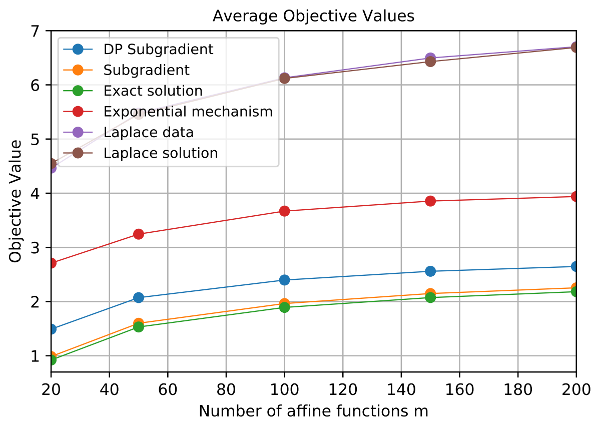


Figure 3: Expected objective value as function of m

attributable to the fact that different hyperparameters, or different ways of sampling $\{(a_i, b_i)\}_{i=1}^m$ were used.

7.2 Results Extensions

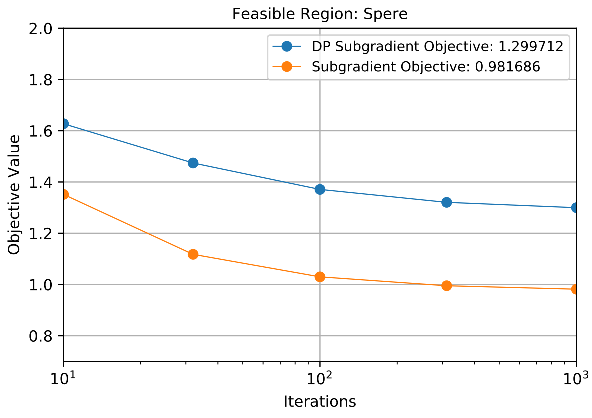
7.2.1 Alternative feasible regions

Figure 4 shows the attained average objective of the DPSM on feasible regions of different convex shapes as a function of the number of iterations. The matrices and vectors C and k contain 2 rows, and hence contain 2 constraints. The constraint parameters were generated from i.i.d standard Gaussian distributions.

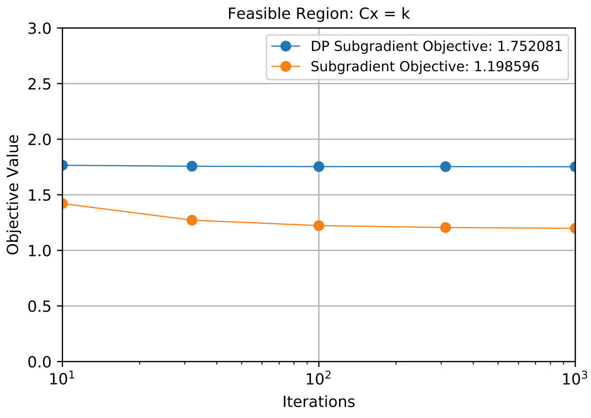
Table 1: Objectives for different feasible regions after 1000 iterations

Shape:	Cube: $c = 1$	Sphere: $r = 1$	$Cx = k$	$Cx \leq k$
Subgradient method	0.93	0.98	1.20	1.20
DPSM	1.51	1.30	1.75	1.80
Difference (%)	*62.4*	*32.7*	*45.8*	*50.0*

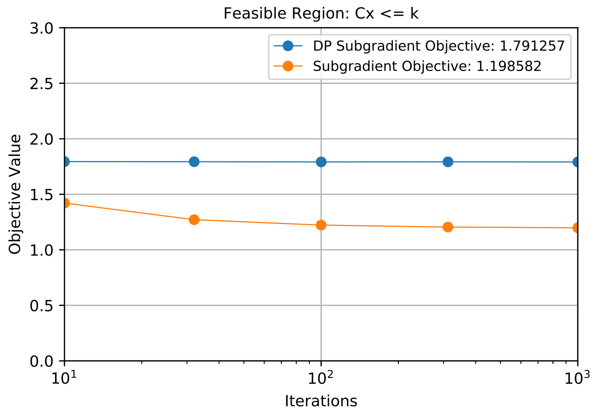
Table 1 shows that the DPSM reached lower levels of suboptimality on different feasible regions after 1000 iterations. More specifically, whereas in [3] the performance of the DPSM was only compared to the regular subgradient method when the feasible region was a hypercube, Table 1 suggests that the DPSM will also work for more general convex constraint sets.



(a) Feasible region: Sphere radius 1



(b) Feasible region: $Cx = k$ (2 constraints)



(c) Feasible region: $Cx \leq k$ (2 constraints)

Figure 4: Expected objective value for different feasible regions

7.2.2 Private projected gradient descent

The table below shows the different objective values attained by the projected DPSM on different convex feasible regions after 1000 iterations.

Table 2: Average objective values for different feasible regions

Shape:	Cube: $c = 1$	Sphere: $r = 1$	$Cx = k$	$Cx \leq k$	$\mathbb{R}^d$
Subgradient method	0.93	0.98	1.20	1.20	0.90
DPSM	1.51	1.30	1.75	1.80	1.81

Table 2 shows that the objective values for different feasible regions were not the same when solutions were projected back onto the feasible region, which they would have been without projection. This provides support of our hypothesis that a projection step is required to ensure feasible solutions when minimizing piecewise affine objective functions. When $\mathcal{P} = \mathbb{R}^d$, the subgradient method is the best performing non-private algorithm, which is in line with expectations. However, the DPSM is actually the worst performing algorithm when $\mathcal{P} = \mathbb{R}^d$. This indicates that the projection step is needed not only to ensure feasible solutions, but also to keep the DPSM from stepping to far away from the minimum. This is supported by the fact that the DPSM attains lower objectives on more restrictive constraints.

7.2.3 Differentially private average subgradient method

The following figure compares the performance of the differentially private average subgradient method for different values of d.

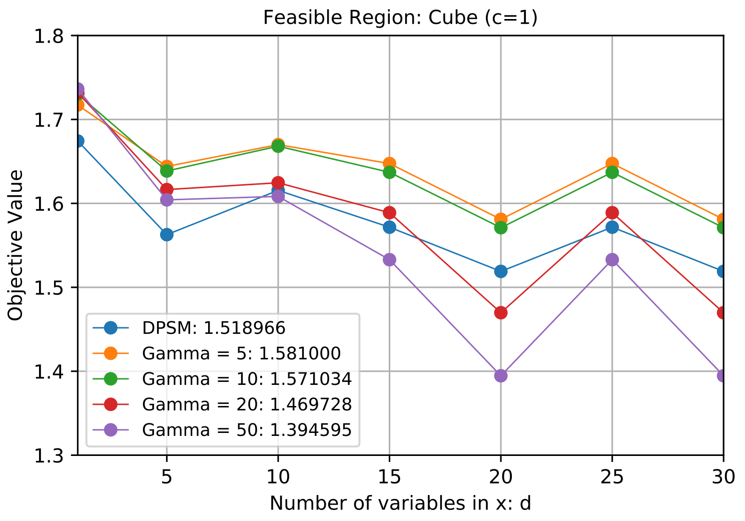


Figure 5: Expected objective value as function of d

The results indicate that for some larger problems (larger values of d), the DPSAM yields lower objectives than the DPSM for larger values of γ. However, using low values of *gamma* yields strictly higher objective values. The reasons behind this are points of potential further research. The evidence indicates that parties interested in using the DPSM should in fact use the DPSAM and treat γ as a trainable hyperparameter. Results show that one can not be worse of by doing this, since when $\gamma = 1$, the DPSM and the DPSAM are identical. However, for example when the problem and γ are large, the DPSAM could outperform the DPSM.

8 Conclusion

In this paper, we studied the class of constrained optimization problems where a piecewise affine objective function is minimized. We critically evaluated the DPSM proposed by Han et al. in [3] to solve this class of problems, and extended their findings to more general convex feasible regions. Additionaly, we introduced the DPASM.

Increasing the level of privacy increases the level of suboptimality. However, the DPSM offers lower levels of suboptimality than several other privacy preserving algorithms, including the Laplace mechanism and the exponential mechanism, for a given level of privacy. Also, we found that the DPSM performs comparably on different convex feasible regions. Then, we found evidence that a projection step should be included in the DPSM to guarantee feasible solutions and can also prevent the DPSM from stepping too far away from the minimum. Finally, the results of testing the DPASM indicated that tuning the value of γ could yield lower values compared to the DPSM, which is relevant for parties interested in implementing the DPSM. An interesting point of research would be to test if and how the DPSM and DPASM work on different objective functions. We leave this investigation to further research.

References

[1] A. Narayanan and V. Shmatikov, "How to break anonymity of the netflix prize dataset," *CoRR*, vol. abs/cs/0610105, 2006.

[2] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of Cryptography* (S. Halevi and T. Rabin, eds.), (Berlin, Heidelberg), pp. 265–284, Springer Berlin Heidelberg, 2006.

[3] G. J. P. Shuo Han, Ufuk Topcu, "Differentially private convex optimization with piecewise affine objectives," *arXiv:1403.6135*, 2014.

[4] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, (Washington, DC, USA), pp. 94–103, IEEE Computer Society, 2007.

[5] E. Nozari, P. Tallapragada, and J. Cortés, "Differentially private distributed convex optimization via functional perturbation," *IEEE Transactions on Control of Network Systems*, vol. 5, no. 1, pp. 395–408, 2016.

[6] M. Kusner, J. Gardner, R. Garnett, and K. Weinberger, "Differentially private bayesian optimization," in *International Conference on Machine Learning*, pp. 918–927, 2015.

[7] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, (New York, NY, USA), pp. 19–30, ACM, 2009.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* New York, NY, USA: Cambridge University Press, 3 ed., 2007.

Appendix A Program Descriptions

```
# Return the subgradient for a given point x
def subgrad_piecewise_affine(a, b, x):
```

```
# Evaluates a piecewise affine objective at x
def eval_piecewise_affine (a, b, x):
```

```
# This function privately  calculates  a subgradient using the exponential mechanism
def  private_subgrad_piecewise_affine (a,  b,  x,  eps,m):
```

```
# Calculates stepsizes  where steplength k  is  1/k^0.51
def  stepsizes ( iterations ) :
```

```
# Samples parameters from a standard Gaussian
def param_sampler(m,d):
```

```
# Samples Laplacian noise
def  laplace_noise (b,dim,eps):
```

```
# Target distribution  for  the MCMC needed for the exponential mechanism
def target_function (x,  a,  b,  bmax, eps):
```

```
# The following method performs k iterations of the subgradient method
def subgradient_method(iterations, a  ,b  ,  x0):
```

```
# The following method performs k iterations of the subgradient method and projects
    solutions back onto a box
def subgradient_method_cube(iterations, a  ,b  ,  x0,  upper, lower):
```

```
# The following method performs k iterations of the subgradient method and projects
    solutions back onto a sphere
def subgradient_method_sphere(iterations, a  ,b  ,  x0,  centre,  radius):
```

```
# The following method performs k iterations of the subgradient method and projects
    solutions back s.t.  Cx=k
def subgradient_method_projection(iterations,  a  ,b  ,  x0,  C, k):
```

```
# DPSM without projection
def private_subgradient_method(iterations,  a  ,b  ,  x0,  eps,m, gamma):
```

```
# DPSM with projection onto box: (lower <= x <= upper)
def private_subgradient_method_cube(iterations,  a  ,b  ,  x0,  eps,m,upper, lower, gamma):
```

```
# DPSM with projection onto a sphere
def private_subgradient_method_sphere(iterations, a ,b , x0, eps, m, centre, radius,
    gamma):
```

```
# DPSM with projection onto Cx=k
def private_subgradient_method_projection(iterations, a ,b , x0, eps, m, C, k, gamma):
```

```
# DPSM with projection s.t. Cx<=k
def private_subgradient_method_ineq_projection(iterations, a ,b , x0, eps, m, C, k, gamma)
    :
```

```
# Solve problem with box constraint exactly using an LP formulation
def lp_solver(ITERATIONS, A, B, EPS, M, D, UPPER, LOWER):
```

```
# Solve problem with Cx=k exactly using an LP formulation
def lp_solver_Ck(ITERATIONS, A, B, EPS, M, D, _C, K):
```

```
# The following method performs 'niter' MCMC steps to reach the target distribution,
    samples a value, and return the
# objecting belonging to the value
def MCMC(a, b, d, eps, niter, x_opt, upper):
```

```
# Adds laplace noise to the data and solves the problem using LP (defined above)
def laplace_data( iterations , a ,b , x0, upper, lower, eps, m):
```

```
# Adds laplace noise to the solution and solves the problem using LP (defined above)
def laplace_solution ( iterations , a ,b , x, upper, lower, eps, m):
```

```
# This method returns objectives of subgradient v. DPSM v. LP
def  main_method(ITERATIONS, runs, UPPER, LOWER, M):
```

```
# This method returns objectives of DPSM v. LaPlace v. Exp v. Subgrad v. LP
def  main_method2(ITERATIONS, runs, UPPER, LOWER, M):
```

```
# Figure 1 Training: runs main method for various numbers of iterations
Runs = 1000
```

```
M = 20

X_10 = main_method(10, Runs, 1, −1, M)
X_32 = main_method(32, Runs, 1, −1, M)
X_100 = main_method(100, Runs, 1, −1, M)
X_312 = main_method(312, Runs, 1, −1, M)
X_1000 = main_method(1000, Runs , 1, −1, M)

meanX_10 = mean(X_10, axis=0)
meanX_32 = mean(X_32, axis=0)
meanX_100 = mean(X_100, axis=0)
meanX_312 = mean(X_312, axis=0)
meanX_1000 = mean(X_1000, axis=0)

X = array([meanX_10, meanX_32, meanX_100, meanX_312, meanX_1000])

its = array([10, 32, 100, 312, 1000])
plt.plot(its,X[:,1], linewidth=0.75, marker='o',label="DP Subgradient Objective: {:.6f}".
    format(min(X[:,1])))
plt.plot(its,X[:,0], linewidth=0.75, marker='o',label= "Subgradient Objective: {:.6f}".
    format(min(X[:,0])))
plt.plot(its,X[:,2], linewidth=0.75, marker='o',label="Exact solution: {:.6f}".format(min
    (X[:,2])))
plt.xscale('log')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Feasible Region: Cube (c=1)", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Iterations')
plt.axis([10, 1000, 0.7 , 2])
plt.savefig('replicate−iterations−1.png', bbox_inches='tight')
plt.savefig('replicate−iterations−1.pdf', bbox_inches='tight')
plt.show()
```

```
# Figure 2 training, Comparison of methods for different values of c
```

```
Runs = 1000
M = 20
Iterations  = 100
d = 5

Y_05 = main_method2(Iterations, Runs, 0.5, −0.5, M)
Y_10 = main_method2(Iterations, Runs , 1, −1, M)
Y_15 = main_method2(Iterations, Runs, 1.5, −1.5, M)
Y_20 = main_method2(Iterations, Runs, 2, −2, M)
Y_25 = main_method2(Iterations, Runs , 2.5, −2.5, M)
Y_30 = main_method2(Iterations, Runs , 3.0, −3.0, M)
Y_35 = main_method2(Iterations, Runs , 3.5, −3.5, M)
Y_40 = main_method2(Iterations, Runs , 4.0, −4.0, M)

meanY_05 = mean(Y_05, axis=0)
meanY_10 = mean(Y_10, axis=0)
meanY_15 = mean(Y_15, axis=0)
meanY_20 = mean(Y_20, axis=0)
meanY_25 = mean(Y_25, axis=0)
meanY_30 = mean(Y_30, axis=0)
meanY_35 = mean(Y_35, axis=0)
meanY_40 = mean(Y_40, axis=0)

Y = array([meanY_05, meanY_10, meanY_15, meanY_20, meanY_25, meanY_30, meanY_35,
    meanY_40])
c  = array([0.5,  1,  1.5,  2,  2.5,  3.0,  3.5,  4.0])
plt.plot(c,Y [:,1],  linewidth=0.75, marker='o',label="DP Subgradient")
plt.plot(c,Y [:,0],  linewidth=0.75, marker='o',label= "Subgradient ")
plt.plot(c,Y [:,2],  linewidth=0.75, marker='o',label="Exact solution")
plt.plot(c,Y [:,3],  linewidth=0.75, marker='o',label="Exponential mechanism")
plt.plot(c,Y [:,4],  linewidth=0.75, marker='o',label="Laplace data")
plt.plot(c,Y [:,5],  linewidth=0.75, marker='o',label="Laplace solution")
plt.xscale(' linear ')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Average Objective Values", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Size of constraint set c')
plt.axis([0.5, 4.0, 0.7 , 18])
plt.savefig('replicate-c-lp.png', bbox_inches='tight')
plt.savefig('replicate-c-lp.pdf', bbox_inches='tight')
plt.show()
```

```
# Sensitivity to m, Comparison of methods for different values of m
Runs = 1000

M_20 = main_method2(100, Runs, 1, -1, 20)
M_50 = main_method2(100, Runs , 1, -1, 50)
M_100 = main_method2(100, Runs, 1, -1, 100)
M_150 = main_method2(100, Runs, 1, -1, 150)
M_200 = main_method2(100, Runs , 1, -1, 200)



meanM_20 = mean(M_20, axis=0)
meanM_50 = mean(M_50, axis=0)
meanM_100 = mean(M_100, axis=0)
meanM_150 = mean(M_150, axis=0)
meanM_200 = mean(M_200, axis=0)



M = array([meanM_20, meanM_50, meanM_100, meanM_150, meanM_200])

m = array([20, 50, 100, 150, 200])
plt.plot(m,M[:,1], linewidth=0.75, marker='o',label="DP Subgradient")
plt.plot(m,M[:,0], linewidth=0.75, marker='o',label= "Subgradient ")
plt.plot(m,M[:,2], linewidth=0.75, marker='o',label="Exact solution")
plt.plot(m,M[:,3], linewidth=0.75, marker='o',label="Exponential mechanism")
plt.plot(m,M[:,4], linewidth=0.75, marker='o',label="Laplace data")
plt.plot(m,M[:,5], linewidth=0.75, marker='o',label="Laplace solution")
plt.xscale('linear')
plt.grid(True)
plt.legend(loc=2, fontsize=9)
plt.title("Average Objective Values", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Number of affine functions m')
plt.axis([20, 200, 0.7 , 7])
plt.savefig('replicate-m.png', bbox_inches='tight')
plt.savefig('replicate-m.pdf', bbox_inches='tight')
plt.show()
```

```
# Method that compares DPSM to subgradient method, both with sphere projections.
    Returns array with objective
# per iteration.
def extension_method(ITERATIONS, runs, CENTRE, RADIUS, M):
```

```
# Figure 1 Training: runs main method for sphere projections for various numbers of
    iterations
Runs = 1000
d = 5
centre = zeros((d,))
radius = 1
M = 20

X_10 = extension_method(10, Runs, centre, radius, M)
X_32 = extension_method(32, Runs, centre, radius, M)
X_100 = extension_method(100, Runs, centre, radius, M)
X_312 = extension_method(312, Runs, centre, radius, M)
X_1000 = extension_method(1000, Runs , centre, radius, M)

meanX_10 = mean(X_10, axis=0)
meanX_32 = mean(X_32, axis=0)
meanX_100 = mean(X_100, axis=0)
meanX_312 = mean(X_312, axis=0)
meanX_1000 = mean(X_1000, axis=0)
```

```
X_sphere = array([meanX_10, meanX_32, meanX_100, meanX_312, meanX_1000])
its = array([10, 32, 100, 312, 1000])
plt.plot(its,X_sphere[:,1], linewidth=0.75, marker='o',label="DP Subgradient Objective:
    {:.6f}".format(min(X_sphere[:,1])))
plt.plot(its,X_sphere[:,0], linewidth=0.75, marker='o',label= "Subgradient Objective: {:.6f
    }".format(min(X_sphere[:,0])))
plt.xscale('log')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Feasible Region: Spere", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Iterations')
plt.axis([10, 1000, 0.7 , 2])
plt.savefig('sphere-objective.png', bbox_inches='tight')
plt.savefig('sphere-objective.pdf', bbox_inches='tight')
plt.show()
```

```
# Method that compares DPSM to subgradient method, both with projections s.t. Cx=k.
    Returns array with objective
# per iteration.
def projection_method(ITERATIONS, runs, D, Contraint_matrix, Constraint_vector, M):
```

```
# Figure 1 Training: runs main method for projections onto Cx=k for various numbers of
    iterations
Runs = 1000
l = 2
d = 5
C,k = param_sampler(l,d)
m = 20

X_10 = projection_method(10, Runs, d, C, k, m)
X_32 = projection_method(32, Runs, d, C, k, m)
X_100 = projection_method(100, Runs, d, C, k, m)
X_312 = projection_method(312, Runs, d, C, k, m)
```

```
X_1000 = projection_method(1000, Runs, d, C, k, m)

meanX_10 = mean(X_10, axis=0)
meanX_32 = mean(X_32, axis=0)
meanX_100 = mean(X_100, axis=0)
meanX_312 = mean(X_312, axis=0)
meanX_1000 = mean(X_1000, axis=0)

X_ck = array([meanX_10, meanX_32, meanX_100, meanX_312, meanX_1000])

its = array([10, 32, 100, 312, 1000])
plt.plot(its,X_ck[:,1], linewidth=0.75, marker='o',label="DP Subgradient Objective: {:.6f}
    ".format(min(X_ck[:,1])))
plt.plot(its,X_ck[:,0], linewidth=0.75, marker='o',label= "Subgradient Objective: {:.6f}".
    format(min(X_ck[:,0])))
#plt.plot(its,X_ck[:,2], linewidth=0.75, marker='o',label= "Exact solution: {:.6f}".format(
    min(X[:,2])))
plt.xscale('log')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Feasible Region: Cx = k", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Iterations')
plt.axis([10, 1000, 0 , 3])
plt.savefig('cxd-objective.png', bbox_inches='tight')
plt.savefig('cxd-objective.pdf', bbox_inches='tight')
plt.show()
```

```
# Method that compares DPSM to subgradient method, both with projections s.t. Cx<=k.
    Returns array with objective
# per iteration.
def ineq_projection_method(ITERATIONS, runs, D, Contraint_matrix, Constraint_vector, M
    ):
```

```
# Figure 1 Training: runs main method for projections s.t. Cx<k for various numbers of
```

```
    iterations
Runs = 1000
l = 2
d = 5
C,k = param_sampler(l,d)
m = 20

X_10 = ineq_projection_method(10, Runs, d, C, k, m)
X_32 = ineq_projection_method(32, Runs, d, C, k, m)
X_100 = ineq_projection_method(100, Runs, d, C, k, m)
X_312 = ineq_projection_method(312, Runs, d, C, k, m)
X_1000 = ineq_projection_method(1000, Runs, d, C, k, m)

meanX_10 = mean(X_10, axis=0)
meanX_32 = mean(X_32, axis=0)
meanX_100 = mean(X_100, axis=0)
meanX_312 = mean(X_312, axis=0)
meanX_1000 = mean(X_1000, axis=0)

X_ineq = array([meanX_10, meanX_32, meanX_100, meanX_312, meanX_1000])

its = array([10, 32, 100, 312, 1000])
plt.plot(its,X_ineq[:,1], linewidth=0.75, marker='o',label="DP Subgradient Objective: {:.6
    f}".format(min(X_ineq[:,1])))
plt.plot(its,X_ineq[:,0], linewidth=0.75, marker='o',label= "Subgradient Objective: {:.6f}"
    .format(min(X_ineq[:,0])))
#plt.plot(its,X[:,2], linewidth=0.75, marker='o',label= "Exact solution: {:.6f}".format(
    min(X[:,2])))
plt.xscale('log')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Feasible Region: Cx <= k", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Iterations')
plt.axis([10, 1000, 0 , 3])
plt.savefig('cx<d-objective.png', bbox_inches='tight')
plt.savefig('cx<d-objective.pdf', bbox_inches='tight')
plt.show()
```

```
# This method returns objectives of subgradient v. DPSM v. LP
def main_method_no_proj(ITERATIONS, runs, UPPER, LOWER, M):
```

```
# Find average objective without projections
Runs = 1000
M = 20
X_no_proj = main_method_no_proj(1000, Runs , Inf, -Inf, M)
meanX_no_proj = mean(X_no_proj, axis=0)
print(meanX_no_proj)
```

```
# This method returns DPSM opjectives for various values of gamma
def main_method_gamma(ITERATIONS, runs, UPPER, LOWER, M, D):
```

```
# Figure averages Training for various values of gamma
Runs = 1000
M = 20

X_1 = main_method_gamma(100, Runs, 1, -1, M, 1)
X_5 = main_method_gamma(100, Runs, 1, -1, M,5)
X_10 = main_method_gamma(100, Runs, 1, -1, M,10)
X_15 = main_method_gamma(100, Runs, 1, -1, M,15)
X_20 = main_method_gamma(100, Runs , 1, -1, M,20)
X_25 = main_method_gamma(100, Runs , 1, -1, M,25)
X_30 = main_method_gamma(100, Runs , 1, -1, M,30)

meanX_1 = mean(X_1, axis=0)
meanX_5 = mean(X_5, axis=0)
meanX_10 = mean(X_10, axis=0)
meanX_15 = mean(X_15, axis=0)
meanX_20 = mean(X_20, axis=0)
```

```
meanX_25 = mean(X_15, axis=0)
meanX_30 = mean(X_20, axis=0)

X_gamma = array([meanX_1, meanX_5, meanX_10, meanX_15, meanX_20, meanX_25,
    meanX_30])

d = array([1, 5, 10, 15, 20,25,30])

plt.plot(d,X_gamma[:,0], linewidth=0.75, marker='o',label= "DPSM: {:.6f}".format(min(
    X_gamma[:,0])))
plt.plot(d,X_gamma[:,1], linewidth=0.75, marker='o',label="Gamma = 5: {:.6f}".format(
    min(X_gamma[:,1])))
plt.plot(d,X_gamma[:,2], linewidth=0.75, marker='o',label="Gamma = 10: {:.6f}".format(
    min(X_gamma[:,2])))
plt.plot(d,X_gamma[:,3], linewidth=0.75, marker='o',label="Gamma = 20: {:.6f}".format(
    min(X_gamma[:,3])))
plt.plot(d,X_gamma[:,4], linewidth=0.75, marker='o',label="Gamma = 50: {:.6f}".format(
    min(X_gamma[:,4])))
#plt.plot(d,X_gamma[:,5], linewidth=0.75, marker='o',label="Gamma = 100: {:.6f}".format(
    min(X_gamma[:,5])))

plt.xscale('linear')
plt.grid(True)
plt.legend(loc="best", fontsize=9)
plt.title("Feasible Region: Cube (c=1)", fontsize=10)
plt.ylabel('Objective Value')
plt.xlabel('Number of variables in x: d')
plt.axis([1, 30, 1.3 , 1.8])
plt.savefig('gamma-objective.png', bbox_inches='tight')
plt.savefig('gamma-objective.pdf', bbox_inches='tight')
plt.show()
```