Approximate Sparse Linear Regression

Sariel Har-Peled[1]
Department of Computer Science, University of Illinois, Urbana, IL, USA
sariel@illinois.edu

Piotr Indyk
Department of Computer Science, MIT, Cambridge, MA, USA
indyk@mit.edu

Sepideh Mahabadi[2]
Data Science Institute, Columbia University, New York, NY, USA
mahabadi@mit.edu

Abstract

In the *Sparse Linear Regression* (SLR) problem, given a $d \times n$ matrix M and a d-dimensional query q, the goal is to compute a k-sparse n-dimensional vector τ such that the error $\|M\tau - q\|$ is minimized. This problem is equivalent to the following geometric problem: given a set P of n points and a query point q in d dimensions, find the closest k-dimensional subspace to q, that is spanned by a subset of k points in P. In this paper, we present data-structures/algorithms and conditional lower bounds for several variants of this problem (such as finding the closest induced k dimensional flat/simplex instead of a subspace).

In particular, we present *approximation* algorithms for the online variants of the above problems with query time $\widetilde{O}(n^{k-1})$, which are of interest in the "low sparsity regime" where k is small, e.g., 2 or 3. For $k = d$, this matches, up to polylogarithmic factors, the lower bound that relies on the *affinely degenerate conjecture* (i.e., deciding if n points in $\mathbb{R}^d$ contains $d+1$ points contained in a hyperplane takes $\Omega(n^d)$ time). Moreover, our algorithms involve formulating and solving several geometric subproblems, which we believe to be of independent interest.

2012 ACM Subject Classification Theory of computation → Computational geometry, Theory of computation → Data structures design and analysis

Keywords and phrases Sparse Linear Regression, Approximate Nearest Neighbor, Sparse Recovery, Nearest Induced Flat, Nearest Subspace Search

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.77

Related Version A full version of the paper is available at `https://arxiv.org/abs/1609.08739`.

Funding This research was supported by NSF and Simons Foundation.

1 Introduction

The goal of the *Sparse Linear Regression* (SLR) problem is to find a sparse linear model explaining a given set of observations. Formally, we are given a matrix $M \in \mathbb{R}^{d \times n}$, and a vector $q \in \mathbb{R}^d$, and the goal is to find a vector τ that is k-sparse (has at most k non-zero entries) and that minimizes $\|q - M\tau\|_2$. The problem also has a natural query/online variant

[1] [Work on this paper was partially supported by NSF AF awards CCF-1421231, and CCF-1217462.]
[2] [This work was done while this author was at MIT.]

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 77; pp. 77:1–77:14


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where the matrix M is given in advance (so that it can be preprocessed) and the goal is to quickly find τ given q.

Various variants of SLR have been extensively studied, in a wide range of fields including
(i) statistics and machine learning [15, 16],
(ii) compressed sensing [6], and
(iii) computer vision [17].
The query/online variant is of particular interest in the application described by Wright *et al.* [17], where the matrix M describes a set of image examples with known labels and q is a new image that the algorithm wants to label.

If the matrix M is generated at random or satisfies certain assumptions, it is known that a natural convex relaxation of the problem finds the optimum solution in polynomial time [3, 4]. However, in general the problem is known to be NP-Hard [13, 5], and even hard to approximate up to a polynomial factor [8] (see below for a more detailed discussion). Thus, it is likely that any algorithm for this problem that guarantees "low" approximation factor must run in exponential time. A simple upper bound for the offline problem is obtained by enumerating $\binom{n}{k}$ possible supports of τ and then solving an instance of the $d \times k$ least squares problem. This results in $n^k(d+k)^{O(1)}$ running time, which (to the best of our knowledge) constitutes the fastest known algorithm for this problem. At the same time, one can test whether a given set of n points in a d-dimensional space is degenerate by reducing it to n instances of SLR with sparsity d. The former problem is conjectured to require $\Omega(n^d)$ time [7] – this is the *affinely degenerate conjecture*. This provides a natural barrier for running time improvements (we elaborate on this below in Section 1.1.1).

In this paper, we study the complexity of the problem in the case where the sparsity parameter k is constant. In addition to the formulation above, we also consider two more constrained variants of the problem. First, we consider the *Affine SLR* where the vector τ is required to satisfy $\|\tau\|_1 = 1$, and second, we consider the *Convex SLR* where additionally τ should be non-negative. We focus on the approximate version of these problems, where the algorithm is allowed to output a k-sparse vector τ' such that $\|M\tau' - q\|_2$ is within a factor of $1+\varepsilon$ of the optimum.

Geometric interpretation. The SLR problem is equivalent to the *Nearest Linear Induced Flat* problem defined as follows. Given a set P of n points in d dimensions and a d-dimensional vector q, the task is to find a k-dimensional flat spanning a subset B of k points in P and the origin, such that the (Euclidean) distance from q to the flat is minimized. The Affine and Convex variants of SLR respectively correspond to finding the *Nearest Induced Flat* and the *Nearest Induced Simplex* problems, where the goal is to find the closest $(k-1)$-dimensional flat/simplex spanned by a subset of k points in P to the query.

Motivation for the problems studied. Given a large[3] set of items (e.g., images), one would like to store them efficiently for various purposes. One option is to pick a relatively smaller subset of representative items (i.e., support vectors), and represent all items as a combination of this *supporting set*. Note, that if our data-set is diverse and is made out of several distinct groups (say, images of the sky, and images of children), then naturally, the data items would use only some of the supporting set for representation (i.e., the representation over the

[3] *Bigger than the biggest thing ever and then some. Much bigger than that in fact, really amazingly immense, a totally stunning size, "wow, that's big", time.* – The Restaurant at the End of the Universe, Douglas Adams.

Table 1 Summary of results. Here, $S(n,d,\varepsilon)$ denotes the preprocessing time and space used by a $(1+\varepsilon)$-ANN (approximate nearest-neighbor) data-structure, and $T_Q(n,d,\varepsilon)$ denotes the query time (we assume all these bounds are at least linear in the dimension d). All the data-structures, except the last one, provide $(1+\varepsilon)$-approximation. In the nearest induced segment case (i.e., this is the offline convex SLR case) the algorithm answers a single query.

	Comment	Space	Query	See
SLR		$n^{k-1}S(n,d,\varepsilon)$	$n^{k-1}T_Q(n,d,\varepsilon)$	Theorem 12
Affine SLR		$n^{k-1}S(n,d,\varepsilon)$	$n^{k-1}T_Q(n,d,\varepsilon)$	Theorem 11
Convex SLR		$n^{k-1}S(n,d,\varepsilon)\log^k n$	$n^{k-1}T_Q(n,d,\varepsilon)\log^k n$	Lemma 25
	$k=2$ & $\varepsilon \le 1$	$nS(n,d,\varepsilon)\log n$	$nT_Q(n,d,\varepsilon)\varepsilon^{-2}\log n$	Full version
Approximate nearest induced segment	$k=2$ $2(1+\varepsilon)$ Approx	$n^{1+O(\frac{1}{(1+\varepsilon)^2})}$		Full version
	$d=O(1)$	$O(n\log n + n/\varepsilon^d)$		Full version

supporting set is naturally sparse). As such, it is natural to ask for a sparse representation of each item over the (sparse but still relatively large) supporting set. (As a side note, surprisingly little is known about how to choose such a supporting set in theory, and the problem seems to be surprisingly hard even for points in the plane.)

Now, when a new item arrives to the system, the task is to compute its best sparse representation using the supporting set, and we would like to do this as fast as possible (which admittedly is not going to be that fast, see below for details).

1.1 Our results

Data-structures. We present data-structures to solve the online variants of the SLR, Affine SLR and Convex SLR problems, for general value of k. Our algorithms use a provided approximate nearest-neighbor (ANN) data-structure as a black box. The new results are summarized in Table 1.

For small values of k, our algorithms offer notable improvements of the query time over the aforementioned naive algorithm, albeit at a cost of preprocessing. Below in Section 1.1.1, we show how our result matches the lower bound that relies on the affinely degenerate conjecture. Moreover, our algorithms involve formulating and solving several interesting geometric subproblems, which we believe to be of independent interest.

Conditional lower bound. We show a conditional lower bound of $\Omega(n^{k/2}/(e^k \log^{\Theta(1)} n))$, for the offline variants of all three problems. Improving this lower bound further, for the case of $k=4$, would imply a nontrivial lower bound for famous Hopcroft's problem. See full version of the paper for the description. Our conditional lower bound result presented in the full version of the paper follows by a reduction from the k-sum problem which is conjectured to require $\Omega(n^{\lceil k/2\rceil}/\log^{\Theta(1)} n)$ time (see e.g., [14], Section 5). This provides further evidence that the off-line variants of the problem require $n^{\Omega(k)}$ time.

1.1.1 Detecting affine degeneracy

Given a point set P in $\mathbb{R}^d$ (here d is conceptually small), it is natural to ask if the points are in general position – that is, all subsets of $d+1$ points are affinely independent. The *affinely degenerate conjecture* states that this problem requires $\Omega(n^d)$ time to solve [7]. This can be achieved by building the arrangement of hyperplanes in the dual, and detecting

any vertex that has $d+1$ hyperplanes passing through it. This problem is also solvable using our data-structure. (We note that since the approximation of our data structure is multiplicative, and in the reduction we only need to detect distance of 0 from larger than 0, we are able to solve the exact degeneracy problem as described next). Indeed, we instantiate Theorem 11, for $k=d$, and using a low-dimensional $(1+\varepsilon)$-ANN data-structure of Arya *et al.* [1]. Such an ANN data-structure uses $S(n,d,\varepsilon)=O(n)$ space, $O(n\log n)$ preprocessing time, and $T_Q(n,d,\varepsilon)=O(\log n+1/\varepsilon^d)=O(\log n)$ query time (for a fixed constant $\varepsilon<1$). Thus, by Theorem 11, our data structure has total space usage and preprocessing time of $\tilde{O}(n^k)$ and a query time of $\tilde{O}(n^{k-1})$. Detecting affine degeneracy then reduces to solving for each point of $q\in P$, the problem of finding the closest $(d-1)$-dimensional induced flat (i.e., passing through d points) of $P\setminus\{q\}$ to q. It is easy to show that this can be solved using our data-structure with an extra log factor[4]. This means that the total runtime (including the preprocessing and the n queries) will be $\tilde{O}(n^k)=\tilde{O}(n^d)$. Thus, up to polylogarithmic factor, the data-structure of Theorem 11 provides an optimal trade-off under the affinely degenerate conjecture. We emphasize that this reduction only rules out the existence of algorithms for online variants of our problems that improve both the preprocessing time from $O(n^k)$, and query time from $O(n^{k-1})$ by much; it does not rule out for example the algorithms with large preprocessing time (in fact much larger than n^k) but small query time.

1.2 Related work

The computational complexity of the approximate sparse linear regression problem has been studied, e.g., in [13, 5, 8]. In particular, the last paper proved a strong hardness result, showing that the problem is hard even if the algorithm is allowed to output a solution with sparsity $k'=k2^{\log^{1-\delta} n}$ whose error is within a factor of $n^c m^{1-\alpha}$ from the optimum, for any constants $\delta,\alpha>0$ and $c>1$.

The query/online version of the Affine SLR problem can be reduced to the *Nearest k-flat Search Problem* studied in [11, 2, 12], where the database consists of a set of k-flats (affine subspaces) of size N and the goal is to find the closest k-flat to a given query point q. Let P be a set of n points in $\mathbb{R}^d$ that correspond to the columns of M. The reduction proceeds by creating a database of all $N=\binom{n}{k}$ possible k-flats that pass through k points of P. However, the result of [2] does not provide multiplicative approximation guarantees, although it does provide some alternative guarantees and has been validated by several experiments. The result of [11], provides provable guarantees and fast query time of $(d+\log N+1/\varepsilon)^{O(1)}$, but the space requirement is quasi-polynomial of the form $2^{(\log N)^{O(1)}}=2^{(k\log n)^{O(1)}}$. Finally the result of [12] only works for the special case of $k=2$, and yields an algorithm with space usage $O\big(n^{14}\varepsilon^{-3}S\big(n^2,d,\varepsilon\big)\big)$ and query time $O\big(T_Q\big(n^2\varepsilon^{-4},d,\varepsilon\big)\log^2 n\big)$[5]. Similar results can be achieved for the other variants.

The SLR problem has a close relationship with the *Approximate Nearest Neighbor (ANN)* problem. In this problem, we are given a collection of N points, and the goal is to build a data structure which, given any query point q, reports the data point whose distance to the query is within a $(1+\varepsilon)$ factor of the distance of the closest point to the query. There are

[4] The details are somewhat tedious – one generates $O(\log n)$ random samples of P where each point is picked with probability half. Now, we build the data-structure for each of the random samples. With high probability, for each of the query point $q\in P$, one of the samples contains, with high probability, the d points defining the closest flat, while not containing q.

[5] The exact exponent is not specified in the main theorem of [12] and it was obtained by an inspection of the proofs in that paper.

many efficient algorithms known for the latter problem. One of the state of the art results for ANN in Euclidean space answers queries in time $(d \log(N)/\varepsilon^2)^{O(1)}$ using $(dN)^{O(1/\varepsilon^2)}$ space [10, 9].

1.3 Our techniques and sketch of the algorithms

Affine SLR (nearest flat). To solve this problem, we first fix a subset $B \subseteq P$ of $k-1$ points, and search for the closest $(k-1)$-flat among those that contain B. Note, that there are at most $n-k+1$ such flats. Each such flat f, as well as the query flat Q_{flat} (containing B and the query q), has only one additional degree of freedom, which is represented by a vector v_H (v_Q, resp.) in a $d-k+1$ space. The vector v_H that is closest to v_Q corresponds to the flat that is closest to q. This can be found approximately using standard ANN data structure, resulting in an algorithm with running time $O(n^{k-1} \cdot T_Q(n,d,\varepsilon))$. Similarly, by adding the origin to the set B, we could solve the SLR problem in a similar way.

Convex SLR (nearest simplex). This case requires an intricate combination of low and high dimensional data structures, and is the most challenging part of this work. To find the closest $(k-1)$-dimensional induced *simplex*, one approach would be to fix B as before, and find the closest corresponding flat. This will work only if the projection of the query onto the closest flat falls inside of its corresponding simplex. Because of that, we need to restrict our search to the flats of *feasible simplices*, i.e., the simplices S such that the projection of the query point onto the corresponding flat falls inside S. If we manage to find this set, we can use the algorithm for affine SLR to find the closest one. Note that finding the distance of the query to the closest non-feasible simplex can easily be computed in time n^{k-1} as the closest point of such a simplex to the query lies on its boundary which is a lower dimensional object.

Let S be the unique simplex obtained from B and an additional point p. Then we can determine whether S is feasible or not only by looking at (i) the relative positioning of p with respect to B, that is, how the simplex looks like in the flat going through S, (ii) the relative positioning of q with respect to B, and (iii) the distance between the query and the flat of the simplex. Thus, if we were given a set of simplices through B such that all their flats were at a distance r from the query, we could build a single data structure for retrieving all the feasible flats. This can be done by mapping all of them in advance onto a unified $(k-1)$ dimensional space (the "parameterized space"), and then using $k-1$ dimensional orthogonal range-searching trees in that space.

However, the minimum distance r is not known in general. Fortunately, as we show the feasibility property is monotone in the distance: the farther the flat of the simplex is from the query point, the weaker constraints it needs to satisfy. Thus, given a threshold value r, our algorithm retrieves the simplices satisfying the restrictions they need to satisfy if they were at a distance r from the query. This allows us to use binary search for finding the right value of r by random sampling. The final challenge is that, since our access is to an approximate NN data structure (and not an exact one), the above procedure yields a superset of feasible simplices. The algorithm then finds the closest flat corresponding to the simplices in this superset. We show that although the reported simplex may not be feasible, its distance to the query is still approximately at most r.

For overview of the offline nearest segment, and conditional lower bound, see the full version.

2 Preliminaries

2.1 Notations

Throughout the paper, we assume $P \subseteq \mathbb{R}^d$ is the set of input points which is of size n. In this paper, for simplicity, we assume that the point-sets are non-degenerate, however this assumption is not necessary for the algorithms. We use the notation $X \subset_i B$ to denote that X is a subset of B of size i, and use $\mathbf{0}$ to denote the origin. For two points $y, u \in \mathbb{R}^d$, the segment the form is denoted by yu, and the line formed by them by $\mathrm{line}(y, u)$.

▶ **Definition 1.** For a set of points S, let $\mathsf{f}_S = \mathrm{aff}(S) = \left\{ \sum_{i=1}^{|S|} \alpha_i p_i \;\middle|\; p_i \in S, \text{ and } \sum_{i=1}^{|S|} \alpha_i = 1 \right\}$ be the $(|S|-1)$-dimensional flat (or $(|S|-1)$-*flat* for short) passing through the points in the set S (aka the *affine hull* of S). The $(|S|-1)$-dimensional simplex ($(|S|-1)$-*simplex* for short) that is formed by the convex-hull of the points of S is denoted by $\triangle_S$. We denote the *interior* of a simplex $\triangle_S$ by $\mathrm{int}(\triangle_S)$.

▶ **Definition 2** (distance and nearest-neighbor)**.** For a point $q \in \mathbb{R}^d$, and a point $p \in \mathbb{R}^d$, we use $\mathsf{d}(q, p) = \|q - p\|_2$ to denote the *distance* between q and p. For a closed set $X \subseteq \mathbb{R}^d$, we denote by $\mathsf{d}(q, X) = \min_{p \in X} \|q - p\|_2$ the *distance* between q and X. The point of X realizing the distance between q and X is the *nearest neighbor* to q in X, denoted by $\mathsf{nn}(q, X)$. We sometimes refer to $\mathsf{nn}(q, X)$ as the *projection* of q onto X.

More generally, given a finite family of such sets $\mathcal{G} = \left\{ X_i \subseteq \mathbb{R}^d \;\middle|\; i = 1, \ldots, m \right\}$, the *distance* of q from $\mathcal{G}$ is $\mathsf{d}(q, \mathcal{G}) = \min_{X \in \mathcal{G}} \mathsf{d}(q, X)$. The *nearest-neighbor* $\mathsf{nn}(q, \mathcal{G})$ is defined analogously to the above.

▶ Assumption 3. Throughout the paper, we assume we have access to a data structure that can answer $(1+\varepsilon)$-ANN queries on a set of n points in $\mathbb{R}^d$. We use $S(n, d, \varepsilon)$ to denote the space requirement of this data structure, and by $T_Q(n, d, \varepsilon)$ to denote the query time.

2.1.1 Induced stars, bouquets, books, simplices and flats

▶ **Definition 4.** Given a point b and a set P of points in $\mathbb{R}^d$, the *star* of P, with the base b, is the set of segments $\mathrm{star}(b, P) = \{ bp \mid p \in P \setminus \{b\} \}$. Similarly, given a set B of points in $\mathbb{R}^d$, with $|B| = k - 1 \leq d$, the *book* of P, with the base B, is the set of simplices $\boldsymbol{\Delta}(B, P) = \left\{ \triangle_{B \cup \{p\}} \;\middle|\; p \in P \setminus B \right\}$. Finally, the set of flats induced by these simplices, is the *bouquet* of P, denoted by $\mathrm{bqt}(B, P) = \left\{ \mathsf{f}_{B \cup \{p\}} \;\middle|\; p \in P \setminus B \right\}$.

If B is a single point, then the corresponding book is a star, and the corresponding bouquet is a set of lines all passing through the single point in B.

▶ **Definition 5.** For a set $P \subseteq \mathbb{R}^d$, let $\mathcal{L}_k(P) = \left\{ \mathsf{f}_{S \cup \{\mathbf{0}\}} \;\middle|\; S \subset_k P \right\}$ be the set of all linear k-dimensional subspaces induced by P, and $\mathcal{F}_k(P) = \{ \mathsf{f}_S \mid S \subset_k P \}$ be the set of all $(k-1)$-flats induced by P. Similarly, let $\boldsymbol{\Delta}_k(P) = \{ \triangle_S \mid S \subset_k P \}$ be the set of all $(k-1)$-simplices induced by P.

2.2 Problems

In the following, we are given a set P of n points in $\mathbb{R}^d$, a query point q and parameters k and $\varepsilon > 0$. We are interested in the following problems:

I. *SLR* (nearest induced linear subspace): Compute $\mathsf{nn}\big(q, \mathcal{L}_k(P)\big)$.
II. *ANLIF* (approximate nearest linear induced flat): Compute a k-flat $\mathsf{f} \in \mathcal{L}_k(P)$, such that $\mathsf{d}(q, \mathsf{f}) \leq (1+\varepsilon)\mathsf{d}(q, \mathcal{L}_k(P))$.

III. *Affine SLR* (nearest induced flat): Compute $\mathsf{nn}\big(q, \mathcal{F}_k(P)\big)$.
IV. *ANIF* (Approximate Nearest Induced Flat): Compute a $(k-1)$-flat $\mathsf{f} \in \mathcal{F}_k(P)$, such that $\mathsf{d}(q, \mathsf{f}) \leq (1+\varepsilon)\mathsf{d}(q, \mathcal{F}_k(P))$.
V. *Convex SLR* (Nearest Induced Simplex): Compute $\mathsf{nn}\big(q, \mathbf{\Delta}_k(P)\big)$.
VI. *ANIS* (Approximate Nearest Induced Simplex): Compute a $(k-1)$-simplex $\triangle \in \mathbf{\Delta}_k(P)$, such that $\mathsf{d}(q, \triangle) \leq (1+\varepsilon)\mathsf{d}(q, \mathbf{\Delta}_k(P))$.

Here, the parameter k corresponds to the *sparsity* of the solution.

3 Approximating the nearest induced flats and subspaces

Here, we show how to solve approximately the online variants of SLR and affine SLR problems. These are later used in Section 4. We start with the simplified case of the uniform star.

3.1 Approximating the nearest neighbor in a uniform star

Input & task. We are given a base point b, a set P of n points in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. We assume that $\|b - p\| = 1$, for all $p \in P$. The task is to build a data structure that can report quickly, for a query point q that is also at distance one from b, the $(1+\varepsilon)$-ANN segment to q in $\mathrm{star}(b, P)$.

Preprocessing. The algorithm computes the set $V = \{p - b \mid p \in P \setminus \{b\}\}$, which lies on a unit sphere in $\mathbb{R}^d$. Next, the algorithm builds a data structure $\mathcal{D}_V$ for answering $(1+\varepsilon)$-ANN queries on V.

Answering a query. For a query point q, the algorithm does the following:
(A) Compute $\tau = q - b$.
(B) Compute $(1+\varepsilon)$-ANN to τ in V, denoted by u using $\mathcal{D}_V$.
(C) Let y be the point in P corresponding to u.
(D) Return $\min\big(\mathsf{d}(q, by),\, 1\big)$.

▶ **Lemma 6.** *Consider a base point b, and a set P of n points in $\mathbb{R}^d$ all on $\mathbb{S}(b,1)$, where $\mathbb{S} = \mathbb{S}(b,1)$ is the sphere of radius 1 centered at b. Given a query point $q \in \mathbb{S}$, the above algorithm reports correctly a $(1+\varepsilon)$-ANN in $\mathrm{star}(b, P)$. The query time is dominated by the time to perform a single $(1+\varepsilon)$-ANN query. (Proof in the full version)*

3.2 Approximating the nearest flat in a bouquet

▶ **Definition 7.** For a set X and a point p in $\mathbb{R}^d$, let $p' = \mathsf{nn}(p, X)$. We use $\mathsf{dir}(X, p)$ to denote the unit vector $(p - p')/\|p - p'\|$, which is the *direction* of p in relation to X.

Input & task. We are given sets B and P of $k-1$ and n points, respectively, in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$. The task is to build a data structure that can report quickly, for a query point q, a $(1+\varepsilon)$-ANN flat to q in $\mathrm{bqt}(B, P)$, see Definition 4.

Preprocessing. Let $F = \mathsf{f}_B$. The algorithm computes the set

$$V = \{\mathsf{dir}(F, p), -\mathsf{dir}(F, p) \mid p \in P \setminus B\},$$

which lies on a $d-k+2$ dimensional unit sphere in $\mathbb{R}^{d-k+1}$, and then builds a data structure $\mathcal{D}_V$ for answering (standard) ANN queries on V.

Answering a query. For a query point q, the algorithm does the following:
(A) Compute $\tau = \mathsf{dir}(F, q)$.
(B) Compute ANN to τ in V, denoted by u using the data structure $\mathcal{D}_V$.
(C) Let p be the point in P corresponding to u.
(D) Return the distance $\mathsf{d}(q, \mathsf{f}_{B\cup\{p\}})$.

▶ **Definition 8.** For sets $X, Y \subseteq \mathbb{R}^d$, let $\mathsf{proj}_X(Y) = \{\mathsf{nn}(q, X) \mid q \in Y\}$ be the *projection* of Y on X.

▶ **Lemma 9.** *Consider two affine subspaces $F \subseteq H$ with a base point $b \in F$, and the orthogonal complement affine subspace $F^\perp = \left\{b + \tau \;\middle|\; \langle \tau, u - v\rangle = 0 \text{ for all } u, v \in F, \tau \in \mathbb{R}^d\right\}$. For an arbitrary point $q \in \mathbb{R}^d$, let $q^\perp = \mathsf{proj}_{F^\perp}(q)$. We have that $\mathsf{d}(q, H) = \mathsf{d}(q^\perp, \mathsf{proj}_{F^\perp}(H))$. (Proof in the full version)*

Using the notation of Assumption 3 and Definition 4, we have the following:

▶ **Lemma 10** (ANN flat in a bouquet)**.** *Given sets B and P of $k-1$ and n points, respectively, in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$, one can preprocess them, using a single ANN data structure, such that given a query point, the algorithm can compute a $(1+\varepsilon)$-ANN to the closest $(k-1)$-flat in $\mathrm{bqt}(B, P)$. The algorithm space and preprocessing time is $O(S(n, d, \varepsilon))$, and the query time is $O(T_Q(n, d, \varepsilon))$. (Proof in the full version)*

3.3 The result

Here, we show simple algorithms for the ANIF and the ANLIF problems by employing Lemma 10. We assume $\varepsilon > 0$ is a prespecified approximation parameter.

Approximating the affine SLR. As discussed earlier, the goal is to find an approximately closest $(k-1)$-dimensional flat that passes through k points of P, to the query. To this end, we enumerate all possible $k-1$ subsets of points of $B \subset_{k-1} P$, and build for each such base set B, the data structure of Lemma 10. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

▶ **Theorem 11.** *The aforementioned algorithm computes a $(1+\varepsilon)$-ANN to the closest $(k-1)$-flat in $\mathcal{F}_k(P)$, see Definition 5. The space and preprocessing time is $O(n^{k-1}S(n, d, \varepsilon))$, and the query time is $O(n^{k-1}T_Q(n, d, \varepsilon))$.*

Approximating the SLR. The goal here is to find an approximately closest k-dimensional flat that passes through k points of P and the origin $\mathbf{0}$, to the query. We enumerate all possible $k-1$ subsets of points of $B' \subset_{k-1} P$, and build for each base set $B = B' \cup \{\mathbf{0}\}$, the data structure of Lemma 10. Given a query, we compute the ANN flat in each one of these data structures, and return the closest one found.

▶ **Theorem 12.** *The aforementioned algorithm computes a $(1+\varepsilon)$-ANN to the closest k-flat in $\mathcal{L}_k(P)$, see Definition 5, with space and preprocessing time of $O(n^{k-1}S(n, d, \varepsilon))$, and the query time of $O(n^{k-1}T_Q(n, d, \varepsilon))$.*

4 Approximating the nearest induced simplex

In this section we consider the online variant of the ANIS problem. Here, we are given the parameter k, and the goal is to build a data structure, such that given a query point q, it can find a $(1+\varepsilon)$-ANN induced $(k-1)$-simplex.

As before, we would like to fix a set B of $k-1$ points and look for the closest simplex that contains B and an additional point from P. The plan is to filter out the simplices for which the projection of the query on to them falls outside of the interior of the simplex. Then we can use the algorithm of the previous section to find the closest flat corresponding to the feasible simplices (the ones that are not filtered out). First we define a canonical space and map all these simplices and the query point to a unique $(k+1)$-dimensional space. As it will become clear shortly, the goal of this conversion is to have a common lower dimensional space through which we can find all feasible simplices using range searching queries.

4.1 Simplices and distances

4.1.1 Canonical realization

In the following, we fix a sequence $B = (p_1, \ldots, p_{k-1})$ of $k-1$ points in $\mathbb{R}^d$. We are interested in arguing about simplices induced by $k+1$ points, i.e., B, an additional input point p_k, and a query point q. Since the ambient dimension is much higher (i.e., d), it would be useful to have a common canonical space, where we can argue about all entities.

▶ **Definition 13.** For a given set of points B, let $F = \mathsf{f}_B$. Let $p \notin F$ be a given point in $\mathbb{R}^d$, and consider the two connected components of $\mathsf{f}_{B\cup\{p\}} \setminus F$, which are *halfflats*. The halfflat containing p is the *positive halfflat*, and it is denoted by $\mathsf{f}^+(B, p)$.

Fix some arbitrary point $\boldsymbol{s}^* \in \mathbb{R}^d \setminus F$, and let $G = \mathsf{f}^+(B, \boldsymbol{s}^*)$ be a *canonical* such halfflat. Similarly, for a fixed point $\boldsymbol{s}^{**} \in \mathbb{R}^d \setminus \mathsf{f}_{B\cup\{\boldsymbol{s}^*\}}$, let $H = \mathsf{f}^+(B \cup \boldsymbol{s}^*, \boldsymbol{s}^{**})$. Conceptually, it is convenient to consider $H = \mathbb{R}^{k-2} \times \mathbb{R} \times \mathbb{R}^+$, where the first $k-2$ coordinates correspond to F, and the first $k-1$ coordinates correspond to G (this can be done by applying a translation and a rotation that maps H into this desired coordinates system). This is the *canonical parameterization* of H.

The following observation formalizes the following: Given a $(k-1)$ dimensional halfflat G passing through B, a point on G is uniquely identified by its distances from the points in B.

▶ **Observation 14.** *Given a sequence of distances $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_{k-1})$, there might be only one unique point $p = p_G(\boldsymbol{\ell}) \in G$, such that $\|p - p_i\| = \ell_i$, for $i = 1, \ldots, k-1$. Such a point might not exist at all*[6].

Next, given G and H, a point q and a value $\ell < \mathsf{d}(q, F)$, we aim to define the points $q_G(\ell)$ and $q_H(\ell)$. Consider a point $q \in \mathbb{R}^d \setminus F$ (not necessarily the query point), and consider any positive $(k-1)$-halfflat g that contains B, and is in distance ℓ from q. Furthermore assume that $\ell = \mathsf{d}(q, \mathsf{g}) < \mathsf{d}(q, F)$. Let q_{g} be the projection of q to g. Observe that, by the Pythagorean theorem, we have that $d_i = \|q_{\mathsf{g}} - p_i\| = \sqrt{\|q - p_i\|^2 - \ell^2}$, for $i = 1, \ldots, k-1$. Thus, the above observation implies, that the canonical point $q_G(\ell) = p_G(d_1, \ldots, d_{k-1})$ (see Observation 14) is uniquely defined. Note that this is somewhat counterintuitive as the flat g and thus the point q_{g} are not uniquely defined. Similarly, there is a unique point $q_H(\ell) \in H$, such that:

- **(i)** the projection of $q_H(\ell)$ to G is the point $q_G(\ell)$,
- **(ii)** $\|q_H(\ell) - q_G(\ell)\| = \ell$, and these two also imply that
- **(iii)** $\|q_H(\ell) - p_i\| = \|q - p_i\|$, for $i = 1, \ldots, k-1$.

[6] *Trilateration* is the process of determining the location of $p \in G$ given $\boldsymbol{\ell}$. *Triangulation* is the process of determining the location when one knows the angles (not the distances).

Therefore, given G and H, a point q and a value $\ell < \mathsf{d}(q, F)$, the points $q_G(\ell)$ and $q_H(\ell)$ are uniquely defined. Intuitively, for a halfflat that passes through B and is at distance ℓ from the query, $q_G(\ell)$ models the position of the projection of the query onto the halfflat, and $q_H(\ell)$ models the position of the query point itself with respect to this halfflat. Next, we prove certain properties of these points.

4.1.2 Orbits

▶ **Definition 15.** For a set of points B in $\mathbb{R}^d$, define Φ_B to be the open set of all points in $\mathbb{R}^d$, such that their projection into F lies in the interior of the simplex $\triangle_B = \text{ConvexHull}(B)$. The set Φ_B is a *prism*.

Consider a query point $q \in \Phi_B$, and its projection $q_B = \mathsf{nn}(q, F)$. Let $r = \mathrm{r}_B(q) = \|q - q_B\|$ be the *radius* of q in relation to B. Using the above canonical parameterization, we have that $q_G(0) = (q_B, r)$, and $q_H(0) = (q_G(0), 0) = (q_B, r, 0)$. More generally, for $\ell \in [0, r]$, we have

$$q_G(\ell) = \left(q_B, \sqrt{r^2 - \ell^2}\right) \qquad \text{and} \qquad q_H(\ell) = \left(q_B, \sqrt{r^2 - \ell^2}, \ell\right). \tag{1}$$

The curve traced by $q_H(\ell)$, as ℓ varies from 0 to r, is the *orbit* of q – it is a quarter circle with radius r. The following lemma states a monotonicity property that is the basis for the binary search over the value of ℓ.

▶ **Lemma 16.** *(i) Define $\widehat{q}(\ell) = \left(\sqrt{r^2 - \ell^2}, \ell\right)$, and consider any point $p = (x, 0)$, where $x \geq 0$. Then, the function $d(\ell) = \|\widehat{q}(\ell) - p\|$ is monotonically increasing for $\ell \in [0, r]$.*

(ii) For any point p in the halfflat G, the function $\|q_H(\ell) - p\|$ is monotonically increasing. (Proof in the full version).

4.1.3 Distance to a simplex via distance to the flat

▶ **Definition 17.** Given a point q, and a distance ℓ, let $\triangle_G(q, \ell)$ be the unique simplex in G, having the points of B and the point $q_G(\ell)$ as its vertices. Similarly, let $\triangle_G(q) = \triangle_G(q, 0)$.

Next, we provide the necessary and sufficient conditions for a simplex to be feasible. This lemma lies at the heart of our data structure.

▶ **Lemma 18.** *(Proof in the full version) Given a query point $q \in \Phi_B$, and a point $p_k \in P \setminus B$, for a number $0 < x \leq \mathsf{d}(q, F)$ we have*

(A) $q_G(x) \in \triangle_G(p_k)$ *and* $\mathsf{d}(q, \mathsf{f}^+(B, p_k)) \leq x \implies \mathsf{d}(q, \triangle_{B \cup \{p_k\}}) \leq x$.

(B) $\mathsf{d}(q, \triangle_{B \cup \{p_k\}}) \leq x$ *and* $q \in \Phi_{B \cup \{p_k\}} \implies q_G(x) \in \triangle_G(p_k)$ *and* $\mathsf{d}(q, \mathsf{f}^+(B, p_k)) \leq x$.

4.2 Approximating the nearest page in a book

▶ **Definition 19.** Let P be a set of n points in $\mathbb{R}^d$, and let B be a sequence of $k - 1$ points. Consider the set of simplices having B and one additional point from P; that is, $\mathbf{\Delta} = \mathbf{\Delta}(B, P) = \left\{\triangle_{B \cup \{p\}} \;\middle|\; p \in P \setminus B\right\}$. The set $\mathbf{\Delta}$ is the *book* induced by (B, P), and to a single simplex in this book is (naturally) a *page*.

The task at hand, is to preprocess $\mathbf{\Delta}$ for ANN queries, as long as (i) the nearest point lies in the interior of one of these simplices and (ii) $q \in \Phi_B$. To this end, we consider the canonical representation of this set of simplices $\mathbf{\Delta}_G = \{\triangle_G(p) \mid p \in P \setminus B\}$.

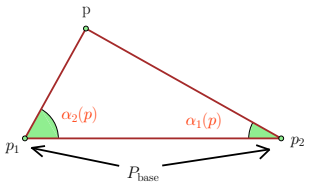

Figure 1 Example base angles when $k = 3$

Idea. The algorithm follows Lemma 18 (A). Given a query point, using standard range-searching techniques, we extract a small number of canonical sets of the points, that in the parametric space, their simplex contains the parameterized query point. This is described in Section 4.2.1. For each of these canonical sets, we use the data structure of Lemma 10 to quickly query each one of these canonical sets for their nearest positive flat (see Remark 4.2.2 below). This would give us the desired ANN.

4.2.1 Reporting all simplices containing a point

▶ **Definition 20.** Let $B = (p_1, \ldots, p_{k-1})$ be a sequence of $k-1$ points in $\mathbb{R}^d$. For a point $p \in \mathbb{R}^d$, consider the $(k-1)$-simplex $\triangle_{B\cup\{p\}}$, which is a full dimensional simplex in the flat $\mathsf{f}_{B\cup\{p\}}$ (see Definition 1). The *base angles* of p (with respect to B), is the $(k-1)$-tuple $\boldsymbol{\alpha}_B(p) = \big(\alpha_1(p), \ldots, \alpha_{k-1}(p)\big)$, where $\alpha_i(p)$ is the dihedral angle between the facet $\triangle_{B\cup\{p\}\setminus\{p_i\}}$ and the base facet $\triangle_B$. See Figure 1, where $k = 3$.

▶ **Observation 21** (Inclusion and base angles). *Let B be a set of $k-1$ points in $\mathbb{R}^{k-1}$ all with their $(k-1)$th coordinate being zero, and let p be an additional point with its $(k-1)$th coordinate being a positive number. Then, for a point $q \in \mathbb{R}^{k-1}$, we have that $q \in \triangle_{B\cup\{p\}} \iff \boldsymbol{\alpha}_B(q) \le \boldsymbol{\alpha}_B(p)$ (i.e., $(\forall i : \alpha_i(q) \le \alpha_i(p))$.*

▶ **Lemma 22.** *Given a set n of $(k-1)$-simplices $\boldsymbol{\Delta}_G$ in $\mathbb{R}^{k-1}$, that all share common $k-1$ vertices, one can build a data structure of size $O(n \log^{k-1} n)$, such that given a query point $q \in \mathbb{R}^{k-1}$, one can compute $O(\log^{k-1} n)$ disjoint canonical sets, such that the union of these sets, is the set of all simplices in $\boldsymbol{\Delta}_G$ that contain q. The query time is $O(\log^{k-1} n)$. (Proof in the full version)*

▶ **Lemma 23.** *The data structure of Lemma 22 can be used to report all simplices that contain a specific point p, and do not contain another point p', which is vertically above p (i.e., the same point with larger $(k-1)$th coordinate). This corresponds to k (possibly unbounded) box queries instead of quadrant query in the orthogonal data structure. The query time and number of canonical sets will be multiplied by at most k. The space bound remains the same. Moreover, we ensure these set of k boxes are disjoint. (Proof in the full version)*

4.2.2 Data structure and correctness

▶ Remark. For a set of points P and a base set B, consider the set of positive halfspaces (the *positive bouquet*) $\mathrm{bqt}^+(B, P) = \{\mathsf{f}^+(B, p) \mid p \in P \setminus B\}$. We can preprocess such a set for ANN queries readily, by using the data structure of Lemma 10. The only modification is that for every positive flat we assign one vector (in the positive direction), instead of two vectors in both directions which we put in the data structure of Section 3.2.

Preprocessing. The algorithm computes the set of canonical simplices $\mathbf{\Delta}_G$, see Eq. (4.2). Next, the algorithm builds the data structure of Lemma 22 for this set of simplices. For each canonical set V in this data structure, for the corresponding set of original points, we build the data structure of Remark 4.2.2 to answer ANN queries on the positive bouquet $\mathrm{bqt}^+(B, V)$. (Observe that the total size of these canonical sets is $O(n \log^{k-1} n)$.)

Answering a query. Given a query point $q \in \Phi_B$, the algorithm computes its projection $q_B = \mathsf{nn}(q, F)$, where $F = \mathsf{f}_B$. Let $r = \|q - q_B\|$ be the radius of q. The desired ANN distance is somewhere in the interval $[0, r]$, and the algorithm maintains an interval $[\alpha, \beta]$ where this distance lies, and uses binary search to keep pruning away on this interval, till reaching the desired approximation.

Observe that for every point $p \in P$, there is a critical value $\gamma(p)$, such that for $x \geq \gamma(p)$, the parameterized point $q_G(x)$ is inside the simplex $\triangle_G(p)$, and is outside if $x < \gamma(p)$. Note that this statement only holds for queries in Φ_B (otherwise it could have been false on simplices $\triangle_{B \cup \{p\}}$ with obtuse angles, see Remark 4.3 for handling the case of $q \notin \Phi_B$).

Now, by Lemma 23, we can compute a polylogarithmic number of canonical sets, such that the union of these sets, are (exactly) all the points with critical values in the range $[\alpha, \beta)$. As long as the number of critical values is at least one, we randomly pick one of these values (by sampling from the canonical sets – one can assume each canonical set is stored in an array), and let γ be this value. We have to decide if the desired ANN is smaller or larger than γ. To this end, we compute a representation, by polylogarithmic number of canonical sets, of all the points of P such that their simplex contains the parameterized point $q_G(\gamma)$, using Lemma 22. For each such canonical set, the algorithm computes the approximate closest positive halfflat, see Remark 4.2.2. Let τ be the minimum distance of such a halfflat computed. If this distance is smaller than γ, then the desired ANN is smaller than γ, and the algorithm continues the search in the interval $[\alpha, \gamma)$, otherwise, the algorithm continues the search in the interval $[\gamma, \beta)$.

After logarithmic number of steps, in expectation, we have an interval $[\alpha', \beta')$, that contains no critical value in it, and the desired ANN distance lies in this interval. We compute the ANN positive flats for all the points that their parameterized simplex contains $q_G(\beta')$, and we return this as the desired ANN distance.

For proof of correctness and query time analysis see the full version.

▶ **Lemma 24** (Approximate nearest induced page)**.** *Given a set P of n points in $\mathbb{R}^d$, a set B of $k-1$ points, and a parameter $\varepsilon > 0$, one can preprocess them, such that given a query point, the algorithm computes an $(1+\varepsilon)$-ANN to the closest page in $\mathbf{\Delta}(B, P)$, see Definition 19. This assumes that (i) the nearest point to the query lies in the interior of the nearest page, and (ii) $q \in \Phi_B$. The algorithm space and preprocessing time is $O(S(n, d, \varepsilon) \log^k n)$, and the query time is $O(T_Q(n, d, \varepsilon) \log^k n)$.*

4.3 Result: nearest induced simplex

The idea is to use brute-force to handle the distance of the query to the $\leq (k-2)$-simplices induced by the given point set which takes $O(n^{k-1})$ time. As such, the remaining task is to handle the $(k-1)$-simplices, and thus we can assume that the nearest point to the query lies in the interior of the nearest simplex, as desired by Lemma 24. To this end, we generate the $\binom{n}{k-1} = O(n^{k-1})$ choices for $B \subseteq P$, and for each one of them we build the data structure of Lemma 24, and query each one of them, returning the closet one found.

▶ Remark. Note that for a set of k points $X \subset_k P$, if the projection of the query onto the simplex $\triangle_A$ falls inside the simplex, i.e. $q \in \Phi_A$, then there exists a subset of $k-1$ points $B \subset_{k-1} X$ such that the projection of the query onto the simplex $\triangle_B$ falls inside the simplex, i.e., $q \in \Phi_B$. Therefore, either the brute-force component of the algorithm finds an ANN, or there exists a set B for which the corresponding data structure reports the correct ANN.

We thus get the following result.

▶ **Theorem 25** (Convex SLR). *Given a set P of n points in $\mathbb{R}^d$, and parameters k and $\varepsilon > 0$, one can preprocess them, such that given a query point, the algorithm can compute a $(1+\varepsilon)$-ANN to the closest $(k-1)$-simplex in $\mathbf{\Delta}_k(P)$, see Definition 5. The algorithm space and preprocessing time is $O(n^{k-1} S(n,d,\varepsilon) \log^k n)$, and the query time is $O(n^{k-1} T_Q(n,d,\varepsilon) \log^k n)$.*

References

1. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6):891–923, 1998. doi:10.1145/293347.293348.
2. Ronen Basri, Tal Hassner, and Lihi Zelnik-Manor. Approximate nearest subspace search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):266–278, 2011.
3. E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inf. Theor.*, 52(2):489–509, February 2006. doi:10.1109/TIT.2005.862083.
4. Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1998. doi:10.1137/S1064827596304010.
5. G. Davis, S. Mallat, and M. Avellaneda. Adaptive greedy approximations. *Constructive Approx.*, 13(1):57–98, 1997. doi:10.1007/BF02678430.
6. David L. Donoho. Compressed sensing. *IEEE Trans. Inf. Theor.*, 52(4):1289–1306, 2006. doi:10.1109/TIT.2006.871582.
7. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete Comput. Geom.*, 13:41–57, 1995. doi:10.1007/BF02574027.
8. Dean P. Foster, Howard J. Karloff, and Justin Thaler. Variable selection is hard. In Peter Grünwald, Elad Hazan, and Satyen Kale, editors, *Proc. 28th Annu. Conf. Comp. Learn. Theo.* (COLT), volume 40 of *JMLR Proceedings*, pages 696–709. JMLR.org, 2015. URL: http://jmlr.org/proceedings/papers/v40/Foster15.html.
9. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th Annu. ACM Sympos. Theory Comput.* (STOC), pages 604–613, 1998. doi:10.1145/276698.276876.
10. E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM J. Comput.*, 2(30):457–474, 2000. doi:10.1137/S0097539798347177.
11. Avner Magen. Dimensionality reductions that preserve volumes and distance to affine spaces, and their algorithmic applications. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 239–253. Springer, 2002.
12. Sepideh Mahabadi. Approximate nearest line search in high dimensions. In *Proc. 26th ACM-SIAM Sympos. Discrete Algs.* (SODA), SODA '15, pages 337–354. SIAM, 2015. URL: http://dl.acm.org/citation.cfm?id=2722129.2722154.
13. Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comput.*, 24(2):227–234, 1995. doi:10.1137/S0097539792240406.

14 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proc. 21st ACM-SIAM Sympos. Discrete Algs.* (SODA), pages 1065–1075. SIAM, 2010. doi:10.1137/1.9781611973075.86.

15 R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Royal Stat. Soc. Series B*, 58(1):267–288, 1996. URL: http://statweb.stanford.edu/~tibs/lasso/lasso.pdf.

16 Robert Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *J. Royal Stat. Soc. Series B*, 73(3):273–282, 2011. doi:10.1111/j.1467-9868.2011.00771.x.

17 John Wright, Allen Y Yang, Arvind Ganesh, Shankar S Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Trans. Pattern Anal. Machine Intel.*, 31(2):210–227, 2009. doi:10.1109/TPAMI.2008.79.