2015 IEEE 56th Annual Symposium on Foundations of Computer Science

# Efficient Inverse Maintenance and Faster Algorithms for Linear Programming

Yin Tat Lee
Department of Mathematics
MIT
Cambridge, USA
Email: yintat@mit.edu

Aaron Sidford
Department of EECS
MIT
Cambridge, USA
Email: sidford@mit.edu

**Abstract**

In this paper, we consider the following *inverse maintenance problem:* given $\mathbf{A} \in \mathbb{R}^{n \times d}$ and a number of rounds $r$, at round $k$, we receive a $n \times n$ diagonal matrix $\mathbf{D}^{(k)}$ and we wish to maintain an efficient linear system solver for $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ under the assumption $\mathbf{D}^{(k)}$ does not change too rapidly. This inverse maintenance problem is the computational bottleneck in solving multiple optimization problems. We show how to solve this problem with $\tilde{O}\left(\mathrm{nnz}(\mathbf{A}) + d^{\omega}\right)$ preprocessing time and amortized $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^2)$ time per round, improving upon previous running times.

Consequently, we obtain the fastest known running times for solving multiple problems including, linear programming and computing a rounding of a polytope. In particular given a feasible point in a linear program with $n$ variables, $d$ constraints, and constraint matrix $\mathbf{A} \in \mathbb{R}^{d \times n}$, we show how to solve the linear program in time $\tilde{O}((\mathrm{nnz}(\mathbf{A}) + d^2)\sqrt{d}\log(\epsilon^{-1}))$. We achieve our results through a novel combination of classic numerical techniques of low rank update, preconditioning, and fast matrix multiplication as well as recent work on subspace embeddings and spectral sparsification that we hope will be of independent interest.

**Index Terms**

Inverse Maintenance Problem; Linear Systems; Linear Programming

## I. Introduction

Solving a sequence of linear systems is the computational bottleneck in many state-of-the-art optimization algorithms, including interior point methods for linear programming [12], [29], [30], [19], the Dikin walk for sampling a point in a polytope [9], and multiplicative weight algorithms for grouped least squares problem [1], etc. In full generality, any particular iteration of these algorithms could require solving an arbitrary positive definite (PD) linear system. However, the PD matrices involved in these algorithms do not change too much between iterations and therefore the average cost per iteration can possibly be improved by maintaining an approximate inverse for the matrices involved.

This insight has been leveraged extensively in the field of interior point methods for linear programming. Combining this insight with classic numerical machinery including preconditioning, low ranks update, and fast matrix multiplication has lead to multiple non-trivial improvements to the state-of-the art running time for linear programming [28], [29], [33], [19], [12]. Prior to our previous work [19], the fastest algorithm for solving a general linear program with $d$ variables, $n$ constraints, and constraint matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, i.e. solving $\min_{\mathbf{A}\vec{x} \geq \vec{b}} \vec{c}^T\vec{x}$, depended intricately on the precise ratio of $d$, $n$, and $\mathrm{nnz}(\mathbf{A})$ the number of non-zero entries in $\mathbf{A}$ (See In Figure I.1). While, these running times were recently improved by our work in [19], the running time we achieved was still a complicated bound of $\tilde{O}(\sqrt{n\beta}(d^2 + nd^{\omega-1}r^{-1} + \beta^{-\omega}r^{2\omega} + \beta^{-(\omega-1)}dr^{\omega}))$ and $\tilde{O}(\sqrt{d}(\mathrm{nnz}(\mathbf{A}) + d^{\omega}))$[1] to compute an $\epsilon$-approximate solution where $\beta \in [\frac{d}{n}, 1]$ and $r \geq 1$ are free parameters to be tuned and $\omega < 2.3729$ is the matrix multiplication constant [35], [7].

In this paper we further improve upon these results. We cast this computational bottleneck of solving a sequence of slowly changing linear systems as a self-contained problem we call the *inverse maintenance problem* and improve upon the previous best running times for solving this problem. This yields an improved running time of $\tilde{O}((\mathrm{nnz}(\mathbf{A}) + d^2)\sqrt{d}\log(\epsilon^{-1}))$ for solving a linear program and improves upon the running time of various problems including multicommodity flow and computing rounding of an ellipse. We achieve our result by a novel application of recent numerical machinery such as subspace embeddings and spectral sparsification to classic techniques for solving this problem which we hope will be of independent interest.

[1] In this paper we use $\tilde{O}$ to hide factors polylogarithmic in $d$, $n$ and the width $U$. See Sec VII for the definition of $U$.

0272-5428/15 $31.00 © 2015 IEEE
DOI 10.1109/FOCS.2015.23

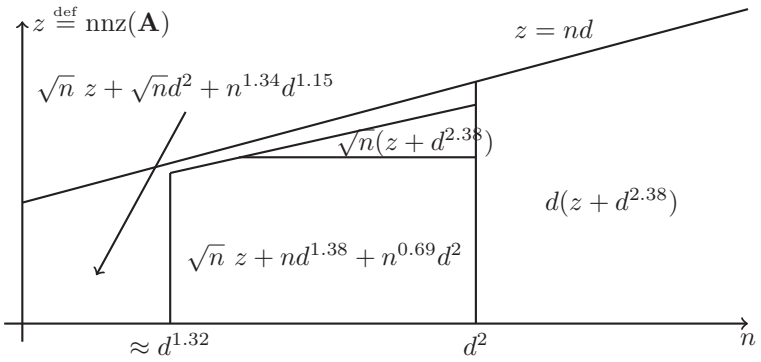


Figure I.1. The figure shows the fastest algorithms for solving a linear program, $\min_{\mathbf{A}\vec{x} \geq \vec{b}} \vec{c}^T \vec{x}$, before [19]. The horizontal axis is the number of constraints $n$ written as a function of $d$, the vertical access is the number of nonzero entries in $\mathbf{A}$ denoted by $z$. Each region of the diagram is the previous best running time for obtaining one bit of precision in solving a linear program with the given parameters.

### A. The Inverse Maintenance Problem

The computational bottleneck we address in this paper is as follows. We are given a fixed matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and an number of rounds $r$. In each round $k \in [r]$ we are given a non-negative vector $\vec{d}^{(k)} \in \mathbb{R}^n_{\geq 0}$ and we wish to maintain access to an approximate inverse for the matrix $(\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A})^{-1}$ where $\mathbf{D}^{(k)} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with $\mathbf{D}^{(k)}_{ii} = d^{(k)}_i$. We call this problem the *inverse maintenance problem* and wish to keep the cost of both maintaining the approximate inverse and applying it as low as possible under certain stability assumptions on $\mathbf{D}^{(k)}$.

For our applications we do not require that the approximate inverse to be stored explicitly. Rather, we simply need an algorithm to solve the linear system $\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A} \vec{x} = \vec{b}$ efficiently for any input $\vec{b} \in \mathbb{R}^d$. We formally define the requirements of this *solver* and define the *inverse maintenance problem* below.

**Definition 1** (Linear System Solver)**.** An algorithm $\mathtt{S}$ is a $\mathcal{T}$-time solver of a PD matrix $\mathbf{M} \in \mathbb{R}^{d \times d}$ if for all $\vec{b} \in \mathbb{R}^d$ and $\epsilon \in (0, 1/2]$, the algorithm outputs a vector $\mathtt{S}(\vec{b}, \epsilon) \in \mathbb{R}^d$ in time $O(\mathcal{T} \log(\epsilon^{-1}))$ such that with high probability in $n$, $\left\|\mathtt{S}(\vec{b}, \epsilon) - \mathbf{M}^{-1}\vec{b}\right\|^2_{\mathbf{M}} \leq \epsilon \left\|\mathbf{M}^{-1}\vec{b}\right\|^2_{\mathbf{M}}$. We call the algorithm $\mathtt{S}$ linear if $\mathtt{S}(\vec{b}, \epsilon) = \mathbf{Q}_\epsilon \vec{b}$ for some $\mathbf{Q}_\epsilon \in \mathbb{R}^{d \times d}$ that depends only on $\mathbf{M}$ and $\epsilon$.

**Definition 2** (Inverse Maintenance Problem)**.** We are given a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$ and a number of rounds $r > 0$. In each round $k \in [r]$ we receive $\vec{d}^{(k)} \in \mathbb{R}^n_{>0}$ and wish to find a $\mathcal{T}$-time solver $\mathtt{S}^{(k)}$ for PD $\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A}$ such that both $\mathcal{T}$ and the cost of constructing $\mathtt{S}^{(k)}$ is small. (See Algorithm 1.)

**Algorithm 1:** Inverse Maintenance Problem

**Input**: Full rank matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$, initial scaling $\vec{d}^{(0)} \in \mathbb{R}^n_{\geq 0}$, the number of rounds $r > 0$.
**for** *Each round* $k \in [r]$ **do**
  **Input**: Current point $\vec{d}^{(k)} \in \mathbb{R}^n_{>0}$
  **Output**: Solver $\mathtt{S}^{(k)}$ for $\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A}$ where $\mathbf{D}^{(k)} \in \mathbb{R}^{n \times n}$ is diagonal with $\mathbf{D}^{(k)}_{ii} = d^{(k)}_i$
**end**

The inverse maintenance problem directly encompasses a computational bottleneck in many interior point methods for linear programming [17], [18], [30] as well as similar methods for sampling a point in a polytope and computing approximate John Ellipsoids. (See Section VII and VIII.) Without further assumptions, we would be forced to pay the cost of solving a general system, i.e. $\tilde{O}(\text{nnz}(\mathbf{A}) + d^\omega)$ [26], [21], [3] where $\omega < 2.3729$ is the matrix multiplication constant [35], [7]. However, in these algorithms the $\mathbf{D}^{(k)}$ cannot change arbitrarily and we develop algorithms to exploit this property.

We consider two different assumptions on how $\mathbf{D}^{(k)}$ can change. The first assumption, we call the $\ell_2$ stability assumption, and is met in most short-step interior point methods and many of our applications. Under this assumption, the multiplicative change from $\vec{d}^{(k)}$ to $\vec{d}^{(k+1)}$ is bounded in $\ell_2$ norm, indicating that multiplicative changes to coordinates happen infrequently on average. The second assumption, we call the $\sigma$ stability assumption, is weaker

| Year | Author | Amortized Cost Per Iteration |
|---|---|---|
| Best known linear system solver [26], [21], [3] | | $\tilde{O}(nnz(\mathbf{A}) + d^{2.3729})$ |
| 1984 | Karmarkar [12] | $\tilde{O}(nnz(\mathbf{A}) + n^{0.5}d^{1.3729} + n^{0.1865}d^{2})$ |
| 1989 | Nesterov and Nemirovskii [27], [28, Thm 8.4.1] | $\tilde{O}(n^{0.5}d^{1.3729} + n^{0.1718}d^{1.9216} + n^{1.8987})$ |
| 1989 | Vaidya [33] | $O(nnz(\mathbf{A}) + d^{2} + n^{0.8334}d^{1.1441})$ |
| 2014 | Lee and Sidford [17] | $\tilde{O}(nnz(\mathbf{A}) + d^{2} + n^{0.8260}d^{1.1340})$ |
| 2015 | This paper | $\tilde{O}(nnz(\mathbf{A}) + d^{2})$ |

Figure I.2. Algorithms for solving the inverse maintenance problem under $\ell^2$ stability guarantee. To simplify comparison we let the *amortized cost per iteration* denote the worst of the average per iteration cost in maintaining the $\mathcal{T}$-time solver and $\mathcal{T}$.

and holds both for these algorithms as well as a new interior point method we provided recently in [19] to improve the running time for linear programming. Under this assumption, the multiplicative change from $\vec{d}^{(k)}$ to $\vec{d}^{(k+1)}$ is bounded in a norm induced by the leverage scores, $\vec{\sigma}_{\mathbf{A}}(\vec{d}^{(k)})$, a common quantity used to measure the importance of rows of a matrix (See Section II-C). Here, multiplicative changes to these important rows happen infrequently on average (See Section IV).

**Definition 3** ($\ell_2$ Stability Assumption)**.** The inverse maintenance problem satisfies the $\ell_2$ *stability assumption* if for all $k \in [r]$ we have $\left\| \log(\vec{d}^{(k)}) - \log(\vec{d}^{(k-1)}) \right\|_2 \leq 0.1$ and $\beta^{-1}\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A} \preceq \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A} \preceq \beta\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$ for $\beta = \mathsf{poly}(n)$.

**Definition 4** ($\sigma$ Stability Assumption)**.** We say that the inverse maintenance problem satisfies the $\sigma$ stability assumption if for each $k \in [r]$ we have $\left\| \log(\vec{d}^{(k)}) - \log(\vec{d}^{(k-1)}) \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{d}^{(k)})} \leq 0.1$, $\left\| \log(\vec{d}^{(k)}) - \log(\vec{d}^{(k-1)}) \right\|_{\infty} \leq 0.1$, and $\beta^{-1}\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A} \preceq \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A} \preceq \beta\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$ for $\beta = \mathsf{poly}(n)$.

Note that many inverse maintenance algorithms do not require the condition $\beta^{-1}\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A} \preceq \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A} \preceq \beta\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$; However, this is a mild technical assumption which holds for all applications we are interested in and it allows our algorithms to achieve further running time improvements.

Also note that the constant $0.1$ in these definitions is arbitrary. If $\vec{d}^{(k)}$ changes more quickly than this, then in many cases we can simply add intermediate $\vec{d}^{(k)}$ to make the conditions hold and only changes the number of rounds by a constant.

Finally, note that the $\sigma$ stability assumption is strictly weaker the $\ell_2$ stability assumption as $\sigma_i \leq 1$ (See Section II-C). We use $\ell^2$ assumption mainly for comparison to previous works and as a warm up case for our paper. In this paper, our central goal is to provide efficient algorithms for solving the inverse maintenance under the weaker $\sigma$ stability assumption.

### *B. Previous Work*

Work on the inverse maintenance problem dates back to the dawn of interior point methods, a broad class of iterative methods for optimization. In 1984, in the first proof of an interior point method solving a linear program in polynomial time, Karmarkar [12] observed that his algorithm needed to solve the inverse maintenance problem under the $\ell_2$ stability guarantee. Under this assumption if one entry of $\vec{d}^{(k)}$ changes by a constant then the rest barely change at all. Therefore the updates to $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ are essentially *low rank*, i.e. on average they are well approximated by the addition of a rank 1 matrix. He noted that it sufficed to maintain an approximation $(\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A})^{-1}$ and by using explicit formulas for rank 1 updates he achieved an average cost of $\tilde{O}(n^{(\omega-2)/2}d^2)$ for the inverse maintenance problem. This improved upon $\tilde{O}(nd^{\omega-1})$, the best running time for solving the necessary linear system known at that time.

Nesterov and Nemirovskii [27] built on the ideas of Karmarkar. They showed that how to maintain an even cruder approximation to $(\mathbf{A}^T\mathbf{D}\mathbf{A})^{-1}$ and use it as a preconditioner for the conjugate gradient method to solve the necessary linear systems. By using the same rank 1 update formulas as Karmarkar and analyzing the quality of these preconditioned systems they were able to improve the running time for solving the inverse maintenance problem.

Vaidya [33] noticed that instead of maintaining an approximation to $(\mathbf{A}^T\mathbf{D}\mathbf{A})^{-1}$ explicitly one can maintain the inverse implicitly and group the updates together into blocks. Using more general low-rank update formulas and fast matrix multiplication he achieved an improved cost of $O(\text{nnz}(\mathbf{A}) + d^2 + (nd^{(\omega-1)})^{5/6})$. His analysis was tailored to the case where the matrix $\mathbf{A}$ is dense; his running time is $O(nd)$ which is essentially optimal when

$\text{nnz}(\mathbf{A}) = nd$. Focusing on the sparse regime, we showed that these terms that were "lower order" in his analysis can be improved by a more careful application of matrix multiplication [19].

Note that, for a broad setting of parameters the previous fastest algorithm for solving the inverse maintenance problem was achieved by solving each linear system from scratch using fast regression algorithms [26], [21], [3] These algorithms all provide an $\tilde{O}(\text{nnz}(\mathbf{A}) + d^\omega)$-time solver for $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ directly by using various forms of dimension reduction. The algorithm in [26] directly projected the matrix to a smaller matrix and the algorithms in [21], [3] each provide iterative algorithms for sampling rows from the matrix to produce a smaller matrix.

Also note that we are unaware of a previous algorithm working directly with the $\sigma$ stability assumption. Under this assumption it is possible that many $\mathbf{D}^{(k)}$ change by a multiplicative constant in a single round and therefore updates to $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ are no longer "low rank" on average, making low rank update techniques difficult to apply cheaply. This is not just an artifact of an overly weak assumption; our linear programming algorithm in [19] had this same complication. In [19] rather than reasoning about the $\sigma$ stability assumption we simply showed how to trade-off the $\ell_2$ stability of the linear systems with the convergence of our algorithm to achieve the previous best running times.

### C. Our Approach

In this paper we show how to solve the inverse maintenance under both the $\ell_2$ stability assumption and the weaker $\sigma$ stability assumption such that the amortized maintenance cost per iteration $\tilde{O}(\text{nnz}(\mathbf{A}) + d^2)$ and the solver runs in time $\tilde{O}((\text{nnz}(\mathbf{A}) + d^2)\log(\epsilon^{-1}))$.

To achieve our results, we show how to use low rank updates to maintain a spectral sparsifier for $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$, that is a weighted subset of $\tilde{O}(d)$ rows of $\mathbf{A}$ that well approximate $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$. We use the well known fact that sampling the rows by leverage scores (see Section II-C) provides precisely such a guarantee and show that we can decrease both the number of updates that need to be performed as well as the cost of those updates.

There are two difficulties that need to be overcome in this approach. The first difficulty is achieving any running time that improves upon both the low rank update techniques and the dimension reduction techniques; even obtaining a running time of $\tilde{O}(\text{nnz}(\mathbf{A}) + d^{\omega-c})$ for the current $\omega$ and $c > 0$ for the inverse maintenance problem under the $\ell_2$ stability assumption was not known. Simply performing rank 1 updates in each iteration is prohibitively expensive as the cost of such an update is $O(d^2)$ and there would be $\Omega(d^c)$ such updates that need to be performed on average in each iteration for some $c > 0$. Furthermore, while Vaidya [33] overcame this problem by grouping the updates together and using fast matrix multiplication, his approach needed to preprocess the rows of $\mathbf{A}$ by exactly computing $(\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A})^{-1}\mathbf{A}$. This takes time $O(nd^{\omega-1})$ and is prohibitively expensive for our purposes if $n$ is much larger than $d$.

To overcome this difficulty, we first compute an initial sparsifier of $\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$ using only $\tilde{O}(d)$ rows of $\mathbf{A}$ in time $\tilde{O}(\text{nnz}(\mathbf{A}) + d^\omega)$ [26], [21], [3]. Having performed this preprocessing, we treat low rank updates to these original rows in the sparsifier and new rows separately. For the original rows we can perform the preprocessing as in Vaidya [33] and the techniques in [33], [17] to obtain the desired time bounds. For low rank updates to the new rows, we show how to use subspace embeddings [26] to approximate their contribution to sufficient accuracy without hurting the asymptotic running times. In Section III we show that this technique alone (even without use of sparsification), suffices to mildly improve the running time of solving the inverse maintenance assumption with the $\ell_2$ stability assumption (but not to obtain the fastest running times in this paper).

The second difficulty is how to use the $\sigma$ stability assumption to bound the number of updates to our sparsifier; under this weak assumption where there can be many low rank changes to $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$. Here we simply show that the $\sigma$ stability assumption implies that the leverage scores do not change too quickly in the norm induced by leverage scores (See Section IV). Consequently, if we sample rows by leverage scores and only re-sample when either $d_i^{(k)}$ or the leverage score changes sufficiently then we can show that the expected number of low rank updates to our sparsifier is small. This nearly yields the result, except that our algorithm will use its own output to compute the approximate leverage scores and the user of our algorithm might use the solvers to adversarially decide the next $\bar{d}^{(k)}$ and this would break our analysis. In Section V we show how to make our solvers indistinguishable from a true solver plus fixed noise with high probability, alleviating the issue.

Ultimately this yields amortized cost per iteration of $\tilde{O}(\text{nnz}(\mathbf{A}) + d^2)$ for the inverse maintenance problem. We remark that our algorithm is one of few instances we can think of where an algorithm needs to use both subspace embeddings [26] as well as iterative linear system solving and sampling techniques [3], [21] for different purposes to achieve a goal; for many applications they are interchangeable.

In Section VI we also show that our approach can be generalized to accommodate for multiple new rows to be added or removed from the matrix. We also show that our inverse maintenance problem yields an $\tilde{O}((\text{nnz}(\mathbf{A}) + d^2)\sqrt{d}\log(\epsilon^{-1}))$ algorithm for solving a linear program improving upon our previous work in [19] (See Section VII-A). In Section VII we provide multiple applications of our results to more specific problems including minimum cost flow, multicommodity flow, non-linear regression, computing a rounding of a polytope, and convex optimization and conclude with an open problem regarding the sampling of a random point in a polytope (Section VIII).

## II. Preliminaries

### A. Notation

Throughout this paper, we use vector notation, e.g $\vec{x} = (x_1, \ldots, x_n)$, to denote a vector and bold, e.g. $\mathbf{A}$, to denote a matrix. We use $\text{nnz}(\vec{x})$ or $\text{nnz}(\mathbf{A})$ to denote the number of nonzero entries in a vector or a matrix respectively. Frequently, for $\vec{x} \in \mathbb{R}^d$ we let $\mathbf{X} \in \mathbb{R}^{d \times d}$ denote $\mathbf{diag}(\vec{x})$, the diagonal matrix such that $\mathbf{X}_{ii} = x_i$. For a symmetric matrix, $\mathbf{M}$ and vector $\vec{x}$ we let $\|\vec{x}\|_{\mathbf{M}} \stackrel{\text{def}}{=} \sqrt{\vec{x}^T\mathbf{M}\vec{x}}$. For vectors $\vec{x}$ and $\vec{y}$ we let $\|\vec{y}\|_{\vec{x}} \stackrel{\text{def}}{=} \sqrt{\sum_i x_i y_i^2}$.

In most applications, we use $d$ to denote the smaller dimension of the problem, which can be either the number of variables or the number of constraints. For example, the linear programs $\min_{\mathbf{A}\vec{x}=\vec{b},\vec{l}\leq\vec{x}\leq\vec{u}} \vec{c}^T\vec{x}$ we solve in Theorem 23 has $n$ variables and $d$ constraints (and use this formulation because it is more general than $\min_{\mathbf{A}\vec{x}\geq\vec{b}} \vec{c}^T\vec{x}$).

### B. Spectral Approximation

For symmetric matrices $\mathbf{N}, \mathbf{M} \in \mathbb{R}^{n\times n}$, we write $\mathbf{N} \preceq \mathbf{M}$ to denote that $\vec{x}^T\mathbf{N}\vec{x} \leq \vec{x}^T\mathbf{M}\vec{x}$ for all $\vec{x} \in \mathbb{R}^n$ and we define $\mathbf{N} \succeq \mathbf{M}$, $\mathbf{N} \prec \mathbf{M}$ and $\mathbf{N} \succ \mathbf{M}$ analogously. We call a symmetric matrix $\mathbf{M}$ positive definite (PD) if $\mathbf{M} \succ \mathbf{0}$. We use $\mathbf{M} \approx_\epsilon \mathbf{N}$ to denote the condition that $e^{-\epsilon}\mathbf{N} \preceq \mathbf{M} \preceq e^{\epsilon}\mathbf{N}$. In the full version, we show that the problem of maintaining a linear solver and maintaining an approximate inverse are nearly equivalent.

**Lemma 5.** *Given a PD matrix $\mathbf{M}$ and a symmetric matrix $\mathbf{N} \approx_{O(1)} \mathbf{M}^{-1}$ that can be applied to a vector in $O(\mathcal{T})$ time we have a linear $(\text{nnz}(\mathbf{M}) + \mathcal{T})$-time solver $\mathtt{S}$ of $\mathbf{M}$.*

**Lemma 6.** *Let $\mathtt{S}$ be a linear solver of $\mathbf{M}$ such that $\mathtt{S}(\vec{b}, \epsilon) = \mathbf{Q}_\epsilon\vec{b}$ for $\epsilon \in [0, 0.1]$. Then, we have*

$$\mathbf{Q}_\epsilon^T\mathbf{M}\mathbf{Q}_\epsilon \approx_{4\sqrt{\epsilon}} \mathbf{M}^{-1} \text{ and } \mathbf{Q}_\epsilon\mathbf{M}\mathbf{Q}_\epsilon^T \approx_{4\sqrt{\epsilon}} \mathbf{M}^{-1}.$$

### C. Leverage Scores

Our algorithms make extensive use of *leverage scores*, a common measure of the importance of rows of a matrix. We denote the leverage scores of a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ by $\vec{\sigma} \in \mathbb{R}^n$ and say the *leverage score of row $i \in [n]$* is $\sigma_i \stackrel{\text{def}}{=} [\mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T]_{ii}$. For $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\vec{d} \in \mathbb{R}^n_{>0}$, and $\mathbf{D} \stackrel{\text{def}}{=} \mathbf{diag}(\vec{d})$ we use the shorthand $\vec{\sigma}_{\mathbf{A}}(\vec{d})$ to denote the leverage scores of the matrix $\mathbf{D}^{1/2}\mathbf{A}$. We frequently use well known facts regarding leverage scores, such as $\sigma_i \in [0, 1]$ and $\left\|\vec{\sigma}\right\|_1 \leq d$. (See [32], [24], [21], [3] for a more in-depth discussion of leverage scores, their properties, and their many applications.)

We use the following two key results regarding leverage scores. The first result states that one can sample $\tilde{O}(d)$ rows of $\mathbf{A}$ according to their leverage score and obtain a spectral approximation of $\mathbf{A}^T\mathbf{A}$ [5]. In Lemma 7, we use a variant of a random sampling lemma stated in [3]. The second, Lemma 8, is a generalization of a result in [32] that has been proved in various settings (see [17] for example) that states that given a solver for a $\mathbf{A}^T\mathbf{A}$ one can efficiently compute approximate leverage scores for $\mathbf{A}$.

**Lemma 7** (Leverage Score Sampling)**.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ and let $\vec{u} \in \mathbb{R}^n$ be an overestimate of $\vec{\sigma}$, the leverage scores of $\mathbf{A}$; i.e. $u_i \geq \sigma_i$ for all $i \in [n]$. Set $p_i \stackrel{\text{def}}{=} \min\left\{1, c_s\epsilon^{-2}u_i\log n\right\}$ for some absolute constant $c_s > 0$ and $\epsilon \in (0, 0.5]$ and let $\mathbf{H} \in \mathbb{R}^{n\times n}$ be a random diagonal matrix where independently $\mathbf{H}_{ii} = \frac{1}{p_i}$ with probability $p_i$ and is 0 otherwise. With high probability in $n$, we have $\text{nnz}(\mathbf{H}) = O(\left\|\vec{u}\right\|_1\epsilon^{-2}\log n)$ and $\mathbf{A}^T\mathbf{H}\mathbf{A} \approx_\epsilon \mathbf{A}^T\mathbf{A}$.*

**Lemma 8** (Computing Leverage Scores)**.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$, let $\vec{\sigma}$ denote the leverage scores of $\mathbf{A}$, and let $\epsilon > 0$. If we have a $\mathcal{T}$-time solver for $\mathbf{A}^T\mathbf{A}$ then in time $\tilde{O}((\text{nnz}(\mathbf{A}) + \mathcal{T})\epsilon^{-2}\log(\epsilon^{-1}))$ we can compute $\vec{\tau} \in \mathbb{R}^n$ such that with high probability in $n$, $(1-\epsilon)\sigma_i \leq \tau_i \leq (1+\epsilon)\sigma_i$ for all $i \in [n]$.*

### D. Matrix Results

Our algorithms combine the sampling techniques mentioned above with the following two results.

**Lemma 9** (Woodbury Matrix Identity)**.** *For any matrices* $\mathbf{A}$*,* $\mathbf{U}$*,* $\mathbf{C}$*,* $\mathbf{V}$ *with compatible size, if* $\mathbf{A}$ *and* $\mathbf{C}$ *are invertible, then,* $(\mathbf{A}+\mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1}+\mathbf{VA}^{-1}\mathbf{U}\right)^{-1}\mathbf{VA}^{-1}$.

**Theorem 10** (Theorem 9 in [26] )**.** *There is a distribution* $\mathcal{D}$ *over* $\tilde{O}(d\epsilon^{-2}) \times n$ *matrices such that for any* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, with high probability in* $n$ *over the choice of* $\mathbf{\Pi} \sim \mathcal{D}$*, we have* $\mathbf{A}^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{A} \approx_\epsilon \mathbf{A}^T\mathbf{A}$*. Sampling from* $\mathcal{D}$ *and computing* $\mathbf{\Pi}\mathbf{A}$ *for any* $\mathbf{\Pi} \in \mathrm{supp}(\mathcal{D})$ *can be done in* $\tilde{O}(\mathrm{nnz}(\mathbf{A}))$ *time.*

## III. Solving the Inverse Maintenance Problem Using $\ell^2$ Stability

In this section we provide an efficient algorithm to solve the inverse maintenance problem under the $\ell_2$ stability assumption (See Section I-A). The $\ell_2$ stability assumption is stronger than the $\sigma$ stability assumption and the result we prove in this section is weaker than the one we prove under the $\sigma$ stability assumption in the next section (although still a mild improvement over many previous results). This section serves as a "warm-up" to the more complicated results in Section IV. The main goal of this section is to prove the following theorem regarding exactly maintaining an inverse under a sequence of low-rank updates. We use this result for our strongest results on solving the inverse maintenance problem under the $\sigma$ stability assumption in Section IV.

**Theorem 11** (Low Rank Inverse Maintenance)**.** *Suppose we are given a matrix* $\mathbf{A} \in \mathbb{R}^{n\times d}$*, a vector* $\vec{b}^{(0)} \in \mathbb{R}^d_{>0}$*, a number of round* $r > 0$*, and in each round* $k \in [r]$ *we receive* $\vec{d}^{(k)} \in \mathbb{R}^n_{>0}$ *such that* $\mathbf{B}^{(k)} \stackrel{\text{def}}{=} \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ *is PD. Further, suppose that the number of pairs* $(i,k)$ *such that* $d_i^{(k)} \neq d_i^{(k-1)}$ *is bounded by* $\alpha \leq d$ *and suppose that there is* $\beta > 2$ *such that* $\beta^{-1}\mathbf{B}^{(0)} \preceq \mathbf{B}^{(k)} \preceq \beta\mathbf{B}^{(0)}$*. Then, in round* $k$*, we can implicit construct a symmetric matrix* $\mathbf{K}$ *such that* $\mathbf{K} \approx_{O(1)} \left(\mathbf{B}^{(k)}\right)^{-1}$ *such that we can apply* $\mathbf{K}$ *to an arbitrary vector in time* $\tilde{O}(d^2\log(\beta))$*. Furthermore, the algorithm in total takes time* $\tilde{O}(sd^{\omega-1} + \alpha r d\left(\frac{\alpha}{r}\right)^{\omega-2} + r\alpha^\omega)$ *where* $s \stackrel{\text{def}}{=} \max\{\mathrm{nnz}(\vec{d}_0), d\}$.

This improves upon the previous best expected running times in [33], [17] which had an additive $\tilde{O}(nd^{\omega-1})$ term which would be prohibitively expensive for our purposes.

To motivate this theorem and our approach, in Section III-A, we prove that Theorem 11 suffices to yield improved algorithms for the inverse maintenance problem under $\ell_2$ stability. Then in Section III-B we prove Theorem 11 using a combination of classic machinery involving low rank update formulas and new machinery involving subspace embeddings [26].

### A. Inverse Maintenance under $\ell_2$ Stability

Here we show how Theorem 11 can be used to solve the inverse maintenance problem under the $\ell_2$ stability assumption. Note this algorithm is primarily intended to illustrate Theorem 11 and is a warm-up to the stronger result in Section IV.

**Theorem 12.** *Suppose that the inverse maintenance problem satisfies the* $\ell^2$ *stability assumption. Then Algorithm 2 maintains a* $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^2)$*-time solver with high probability in total time* $\tilde{O}(sd^{\omega-1} + r\left(nd^{\omega-1}\right)^{\frac{2\omega}{2\omega+1}})$ *where* $s = \max\{\max_{k\in[r]}\mathrm{nnz}(d^{(k)}), d\}$ *and* $r$ *is the number of rounds.*

*Proof:* Recall that by the $\ell^2$ stability assumption $\left\|\log(\vec{d}^{(k)}) - \log(\vec{d}^{(k-1)})\right\|_2 \leq 0.1$ for all $k$. Therefore, in the $r$ rounds of the inverse maintenance problem at most $O(r^2)$ coordinates of $\vec{d}^{(k)}$ change by any fixed multiplicative constant. Consequently, the condition, $0.9d_i^{(apr)} \leq d_i^{(k)} \leq 1.1d_i^{(apr)}$ in Algorithm 2 is false at most $O(r^2)$ times during the course of the algorithm and at most $O(r^2)$ coordinates of the vector $\vec{d}^{(apr)}$ change during the course of the algorithm.

Therefore, we can use Theorem 11 on $\mathbf{A}^T\mathbf{D}^{(apr)}\mathbf{A}$ with $\alpha = O(r^2)$ and $s$ as defined in the theorem statement. Consequently, the total cost of maintaining an implicit approximation of $\left(\mathbf{A}^T\mathbf{D}^{(apr)}\mathbf{A}\right)^{-1}$ for $r$ rounds is $\tilde{O}\left(sd^{\omega-1} + dr^{\omega+1} + r^{2\omega+1}\right)$.

To further improve the running time we restart the maintenance procedure every $(sd^{\omega-1})^{\frac{1}{2\omega+1}}$ iterations. This yields a total cost of maintenance that is bounded by

$$\tilde{O}\left(\left\lceil \frac{r}{(sd^{\omega-1})^{\frac{1}{2\omega+1}}} \right\rceil \left(sd^{\omega-1} + d\left((sd^{\omega-1})^{\frac{1}{2\omega+1}}\right)^{\omega+1} + \left((sd^{\omega-1})^{\frac{1}{2\omega+1}}\right)^{2\omega+1}\right)\right).$$

**Algorithm 2:** Algorithm for the $\ell^2$ Stability Assumption

**Input:** Initial point $\vec{d}^{(0)} \in \mathbb{R}^n_{>0}$.
Set $\vec{d}^{(apr)} := \vec{d}^{(0)}$.
$\mathbf{Q}^{(0)} \stackrel{\text{def}}{=} \mathbf{A}^T \mathbf{D}^{(apr)} \mathbf{A}$.
Let $\mathbf{K}^{(0)}$ be an approximate inverse of $\mathbf{Q}^{(0)}$ computed using Theorem 11.
**Output:** A $\tilde{O}(d^2 + \text{nnz}(\mathbf{A}))$-time linear solver for $\mathbf{A}^T \mathbf{D}^{(0)} \mathbf{A}$ (using Lemma 5 on $\mathbf{K}^{(0)}$).
**for** *each round* $k \in [r]$ **do**
  **Input:** Current point $\vec{d}^{(k)} \in \mathbb{R}^n_{>0}$.
  **for** *each coordinate* $i \in [n]$ **do**
    **if** $0.9 d_i^{(apr)} \leq d_i^{(k)} \leq 1.1 d_i^{(apr)}$ *is false* **then**
      $d_i^{(apr)} := d_i^{(k)}$.
  **end**
  $\mathbf{Q}^{(k)} \stackrel{\text{def}}{=} \mathbf{A}^T \mathbf{D}^{(apr)} \mathbf{A}$.
  Let $\mathbf{K}^{(k)}$ be an approximate inverse of $\mathbf{Q}^{(k)}$ computed using Theorem 11.
  **Output:** A $\tilde{O}(d^2 + \text{nnz}(\mathbf{A}))$-time linear solver for $\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A}$ (using Lemma 5 on $\mathbf{K}^{(k)}$).
**end**

which is the same as

$$\tilde{O}\left( \left\lceil \frac{r}{(sd^{\omega-1})^{\frac{1}{2\omega+1}}} \right\rceil \left( sd^{\omega-1} + d(sd^{\omega-1})^{\frac{\omega+1}{2\omega+1}} \right) \right).$$

Since $\omega \leq 1 + \sqrt{2}$, we have $d(d^\omega)^{\frac{\omega+1}{2\omega+1}} \leq d^\omega$ and thus $sd^{\omega-1}$ dominates $d(sd^{\omega-1})^{\frac{\omega+1}{2\omega+1}}$ when $s = d$. Furthermore, as $s \geq d$ the term $sd^{\omega-1}$ grows faster than $d(sd^{\omega-1})^{\frac{\omega+1}{2\omega+1}}$. Consequently, $sd^{\omega-1}$ always dominates and we have the desired result. ■

### *B. Low Rank Inverse Maintenance*

Here we prove Theorem 11 and provide an efficient algorithm for maintaining the inverse of a matrix under a bounded number of low rank modifications. The algorithm we present is heavily motivated by the work in [33] and the slight simplifications in [17]. However, our algorithm improves upon the previous best cost of $\tilde{O}(nd^{\omega-1} + dr^{\omega+1} + r^{2\omega+1})$ achieved in [17] by a novel use of subspace embeddings [26] that we hope will be of independent interest.

We break our proof of Theorem 11 into several parts. First, we provide a simple technical lemma about maintaining the weighted product of two matrices using fast matrix multiplication.

**Lemma 13.** *Let $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n \times d}$ and suppose that in each round $k$ of $r$ we receive $\vec{x}^{(k)}, \vec{y}^{(k)} \in \mathbb{R}^n$. Suppose that the number of pairs $(i, k)$ such that $x_i^{(k)} \neq x_i^{(k-1)}$ or $y_i^{(k)} \neq y_i^{(k-1)}$ is upper bounded by $\alpha$. Suppose that $\text{nnz}(\vec{x}^{(1)}) \leq \alpha$, $\text{nnz}(\vec{y}^{(1)}) \leq \alpha$ and $\alpha \leq d$. With $O(d\alpha^{\omega-1})$ time precomputation, we can compute $\mathbf{X}^{(k)} \mathbf{A} \mathbf{B}^T \mathbf{Y}^{(k)}$ explicitly in an average cost of $O(\alpha d \left(\frac{\alpha}{r}\right)^{\omega-2})$ per iteration.*

*Proof:* For the initial round, we compute $\mathbf{X}^{(0)} \mathbf{A} \mathbf{B}^T \mathbf{Y}^{(0)}$ explicitly by multiplying an $\alpha \times d$ and a $d \times \alpha$ matrix. Since $\alpha \leq d$, we can compute $\mathbf{X}^{(0)} \mathbf{A} \mathbf{B}^T \mathbf{Y}^{(0)}$ by multiplying $O\left(\frac{d}{\alpha}\right)$ matrices of size $\alpha \times \alpha$. Using fast matrix multiplication this takes time $O(d\alpha^{\omega-1})$.

For all $k \in [r]$ let $\mathbf{\Delta}_X^{(k)} \stackrel{\text{def}}{=} \mathbf{X}^{(k)} - \mathbf{X}^{(k-1)}$ and $\mathbf{\Delta}_Y^{(k)} = \mathbf{Y}^{(k)} - \mathbf{Y}^{(k-1)}$. Consequently,

$$\mathbf{X}^{(k)} \mathbf{A} \mathbf{B}^T \mathbf{Y}^{(k)} = \left( \mathbf{X}^{(k-1)} + \mathbf{\Delta}_X^{(k)} \right) \mathbf{A} \mathbf{B}^T \left( \mathbf{Y}^{(k-1)} + \mathbf{\Delta}_Y^{(k)} \right).$$

Note that $\text{nnz}(\vec{x}^{(k)})$ and $\text{nnz}(\vec{y}^{(k)})$ are less than $2\alpha$. Thus, if we let $u_k$ denote the number of coordinates $i$ such that $x_i^{(k)} \neq x_i^{(k-1)}$ or $y_i^{(k)} \neq y_i^{(k-1)}$, then to compute $\mathbf{X}^{(k)} \mathbf{A} \mathbf{B}^T \mathbf{Y}^{(k)}$ we need only multiply a $u_k \times d$ with a $d \times u_k$ matrix and multiply two $2\alpha \times d$ matrices with $d \times u_k$ matrices. Since $u_k \leq \alpha$, the running time is dominated by the time to compute the $2\alpha \times d$ and a $d \times u_k$ . Since $u_k \leq \alpha \leq d$, we can do this by multiplying $O\left(\frac{\alpha}{u_k} \cdot \frac{d}{u_k}\right)$ matrices of size $u_k \times u_k$. Using fast matrix multiplication, this takes time $O(\alpha d u_k^{\omega-2})$.

Summing over all $u_k$ we see that the total cost of computing the $\mathbf{X}^{(k)}\mathbf{A}\mathbf{B}^T\mathbf{Y}^{(k)}$ is

$$O\left(\sum_{k=1}^{r}\alpha d u_k^{\omega-2}\right) \leq O\left(\alpha d r\left(\frac{1}{r}\sum_{k=1}^{r}u_k\right)^{\omega-2}\right) \leq O\left(\alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2}\right),$$

where in the second inequality we used the concavity of $x^{\omega-2}$. Since this is at least the $O(d\alpha^{\omega-1})$ cost of computing $\mathbf{X}^{(0)}\mathbf{A}\mathbf{B}^T\mathbf{Y}^{(0)}$ we obtain the desired result. ■

Next, for completeness we prove a slightly more explicit variant of a Lemma in [17].

**Lemma 14.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\vec{d}^{(0)}, \ldots, \vec{d}^{(r)} \in \mathbb{R}^n_{>0}$, and $\mathbf{B}^{(k)} \stackrel{\text{def}}{=} \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ for all $k$. Suppose that the number of pairs $(i,k)$ such that $d_i^{(k)} \neq d_i^{(k-1)}$ is bounded by $\alpha$ and $\alpha \leq d$. In $O(nd^{\omega-1} + \alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2})$ total time, we can explicitly output $\mathbf{C} \in \mathbb{R}^{n\times d}$ and a $\mathbf{V}^{(k)} \in \mathbb{R}^{n\times n}$ in each round $k$ such that*

$$\left(\mathbf{B}^{(k)}\right)^{-1} = \left(\mathbf{B}^{(0)}\right)^{-1} - \mathbf{C}^T\Delta^{(k)}\mathbf{V}^{(k)}\Delta^{(k)}\mathbf{C}$$

*where $\Delta^{(k)}$ is a $n \times n$ diagonal matrix with $\Delta^{(k)}_{ii} = \vec{d}_i^{(k)} - \vec{d}_i^{(0)}$. Furthermore, $\mathbf{V}^{(k)}$ is an $\tilde{O}(\alpha) \times \tilde{O}(\alpha)$ matrix if we discard the zero rows and columns.*

*Proof:* Let $\mathbf{B} \stackrel{\text{def}}{=} \mathbf{B}^{(0)}$ for notational simplicity. Note that we can compute $\mathbf{B}$ directly in $O(nd^{\omega-1})$ time using fast matrix multiplication. Then, using fast matrix multiplication we can then compute $\mathbf{B}^{-1}$ explicitly in $O(d^\omega)$ time. Furthermore, we can compute $\mathbf{C} \stackrel{\text{def}}{=} \mathbf{A}\mathbf{B}^{-1}$ in $O(nd^{\omega-1})$ time.

Let $u_k$ be the number of non-zero entries in $\Delta^{(k)}$ and $\mathbf{P}^{(k)}$ be a $u_k \times n$ matrix such that the $\mathbf{P}^{(k)}_{ij} = 1$ if $j$ is the index of the $i^{th}$ non-zero diagonal entries in $\Delta^{(k)}$ and 0 otherwise. Let $\mathbf{S}^{(k)} = \left(\mathbf{P}^{(k)}\right)\left(\Delta^{(k)}\right)^{\dagger}\left(\mathbf{P}^{(k)}\right)^T$. By definition of $\mathbf{S}^{(k)}$ and $\mathbf{P}^{(k)}$, we have

$$\Delta^{(k)} = \Delta^{(k)}\left(\mathbf{P}^{(k)}\right)^T\mathbf{S}^{(k)}\mathbf{P}^{(k)}\Delta^{(k)}.$$

By the Woodbury matrix identity (Lemma 9), we know that

$$\begin{aligned}\left(\mathbf{B}^{(k)}\right)^{-1} &= \left(\mathbf{B} + \mathbf{A}^T\Delta^{(k)}\left(\mathbf{P}^{(k)}\right)^T\mathbf{S}^{(k)}\mathbf{P}^{(k)}\Delta^{(k)}\mathbf{A}\right)^{-1}\\ &= \mathbf{B}^{-1} - \mathbf{C}^T\Delta^{(k)}\left(\mathbf{P}^{(k)}\right)^T\mathbf{T}^{(k)}\mathbf{P}^{(k)}\Delta^{(k)}\mathbf{C}\end{aligned}$$

where $\mathbf{T}^{(k)} = ((\mathbf{S}^{(k)})^{-1} + \mathbf{P}^{(k)}\Delta^{(k)}\mathbf{C}\mathbf{A}^T\Delta^{(k)}(\mathbf{P}^{(k)})^T)^{-1}$. Now by Lemma 13, we know that we can maintain $\Delta^{(k)}\mathbf{C}\mathbf{A}^T\Delta^{(k)}$ in an additional $O(\alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2})$ time, using that $\mathbf{C} = \mathbf{A}\mathbf{B}^{-1}$ was precomputed in $O(nd^{\omega-1})$ time. Having the matrix $\Delta^{(k)}\mathbf{C}\mathbf{A}^T\Delta^{(k)}$ explicitly, one can compute $\mathbf{P}^{(k)}\Delta^{(k)}\mathbf{C}\mathbf{A}^T\Delta^{(k)}\left(\mathbf{P}^{(k)}\right)^T$ in $O(u_k^2)$ time and hence compute $\mathbf{T}^{(k)}$ in $O(u_k^\omega)$ time using fast matrix multiplication. Hence, the total running time is $O(nd^{\omega-1} + \alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2} + \sum u_k^\omega)$. The convexity of $x^\omega$ yields $\sum u_k^\omega \leq \alpha^\omega$. Hence, the total time is bounded by

$$O(nd^{\omega-1} + \alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2} + \alpha^\omega) = O(nd^{\omega-1} + \alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2}).$$

By setting $\mathbf{V}^{(k)} = \left(\mathbf{P}^{(k)}\right)^T\mathbf{T}^{(k)}\mathbf{P}^{(k)}$, we have the desired formula. Note that $\mathbf{T}^{(k)}$ is essentially $\mathbf{V}^{(k)}$ with the zero columns and rows and hence we have the desired running time. ■

We now everything we need to prove Theorem 11.

*Proof of Theorem 11:* Let $S \subset [n]$ denote the indices for which $\vec{d}^{(0)}$ is nonzero. Furthermore, let us split each vector $\vec{d}^{(k)}$ into a vector $\vec{e}^{(k)}$ for the coordinates in $S$ and $\vec{f}^{(k)}$ for the coordinates not in $S$, i.e. $\vec{e}^{(k)}, \vec{f}^{(k)} \in \mathbb{R}^n_{>0}$ such that $\vec{d}^{(k)} = \vec{e}^{(k)} + \vec{f}^{(k)}$. By the stability guarantee, we know $\beta^{-1}\mathbf{B}^{(0)} \preceq \mathbf{B}^{(k)} \preceq \beta\mathbf{B}^{(0)}$ for some $\beta$. We make $\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}$ invertible for all $k$ by adding $\frac{1}{10\beta}\vec{d}^{(0)}$ to all of $\vec{d}^{(k)}$and $\vec{e}^{(k)}$; we will show this only increases the error slightly.

Now, we can compute $\mathbf{B}^{(0)}$ and $\left(\mathbf{B}^{(0)}\right)^{-1}$ in $O(sd^{\omega-1})$ time using fast matrix multiplication where $s = |S|$. Furthermore, using Lemma 14 we can compute $\mathbf{C}$ and maintain $\mathbf{V}^{(k)}$ such that

$$\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1} = \left(\mathbf{B}^{(1)}\right)^{-1} - \mathbf{C}^T\Delta^{(k)}\mathbf{V}^{(k)}\Delta^{(k)}\mathbf{C}$$

in total time $O(sd^{\omega-1} + \alpha d r\left(\frac{\alpha}{r}\right)^{\omega-2})$. All that remains is to maintain the contribution from $\vec{f}^{(k)}$.

Using our representation of $\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}$ and the Woodbury matrix identity (Lemma 9), we have

$$\begin{aligned}\left(\mathbf{B}^{(k)}\right)^{-1} &= \left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A} + \mathbf{A}^T\mathbf{F}^{(k)}\mathbf{A}\right)^{-1} \\ &= \left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A} + \mathbf{A}^T\mathbf{F}^{(k)}\left(\mathbf{F}^{(k)}\right)^{\dagger}\mathbf{F}^{(k)}\mathbf{A}\right)^{-1} \\ &= \left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1} - \left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}\left(\mathbf{M}^{(k)}\right)^{-1}\mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1} \end{aligned} \tag{III.1}$$

where

$$\mathbf{M}^{(k)} = \left(\left(\mathbf{F}^{(k)}\right)^{\dagger}\right)^{-1} + \mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} \tag{III.2}$$

Now note that

$$\begin{aligned}\mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} &= \mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} \\ &= \left(\mathbf{N}^{(k)}\right)^T\mathbf{N}^{(k)}\end{aligned}$$

where

$$\begin{aligned}\mathbf{N}^{(k)} &\stackrel{\text{def}}{=} \left(\mathbf{E}^{(k)}\right)^{1/2}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} \\ &= \left(\left(\mathbf{D}^{(0)}\right)^{1/2} + \left(\left(\mathbf{E}^{(k)}\right)^{1/2} - \left(\mathbf{D}^{(0)}\right)^{1/2}\right)\right)\mathbf{A}\left(\left(\mathbf{B}^{(0)}\right)^{-1} - \mathbf{C}^T\Delta^{(k)}\mathbf{V}^{(k)}\Delta^{(k)}\mathbf{C}\right)\mathbf{A}^T\mathbf{F}^{(k)}.\end{aligned}$$

Note that computing the $\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}$ term directly would be prohibitively expensive. Instead, we compute a spectral approximation to $\left(\mathbf{N}^{(k)}\right)^T\mathbf{N}^{(k)}$ and show that suffices. Since $\left(\mathbf{N}^{(k)}\right)^T\mathbf{N}^{(k)}$ is a rank $\alpha$ matrix, we use Theorem 10 to $\mathbf{\Pi} \in \mathbb{R}^{\tilde{O}(\alpha)\times d}$ such that

$$\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)} \approx_1 \left(\mathbf{N}^{(k)}\right)^T\mathbf{N}^{(k)} \tag{III.3}$$

for all $k$ with high probability. Now, we instead consider the cost of maintaining

$$\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}.$$

To see the cost of maintaining $\mathbf{\Pi}\mathbf{N}^{(k)}$, we separate the matrix as follows:

$$\begin{aligned}\mathbf{\Pi}\mathbf{N}^{(k)} &= \mathbf{\Pi}\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} \\ &\quad + \mathbf{\Pi}\left(\left(\mathbf{E}^{(k)}\right)^{1/2} - \left(\mathbf{D}^{(0)}\right)^{1/2}\right)\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} \\ &\quad - \mathbf{\Pi}\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\mathbf{C}^T\Delta^{(k)}\mathbf{V}^{(k)}\Delta^{(k)}\mathbf{C}\mathbf{A}^T\mathbf{F}^{(k)} \\ &\quad - \mathbf{\Pi}\left(\left(\mathbf{E}^{(k)}\right)^{1/2} - \left(\mathbf{D}^{(0)}\right)^{1/2}\right)\mathbf{A}\mathbf{C}^T\Delta^{(k)}\mathbf{V}^{(k)}\Delta^{(k)}\mathbf{C}\mathbf{A}^T\mathbf{F}^{(k)}.\end{aligned} \tag{III.4}$$

For the first term in (III.4), note that $\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}$ is a $s\times d$ matrix and hence we can precompute $\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}$ in $\tilde{O}(sd^{\omega-1})$ time and therefore precompute $\mathbf{\Pi}\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}$ in the same amount of time. Note that $\mathbf{\Pi}$ is a $\tilde{O}(\alpha)\times d$ matrix, so, we can write

$$\mathbf{\Pi}\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)} = \mathbf{\Lambda}\mathbf{K}\mathbf{A}^T\mathbf{F}^{(k)}$$

where $\mathbf{K}$ is some explicitly computed $n\times d$ matrix and $\mathbf{\Lambda}$ is a diagonal matrix with only $\tilde{O}(\alpha)$ non-zeros. Consequently, we can use Lemma 13 to maintain $\mathbf{\Pi}(\mathbf{D}^{(0)})^{1/2}\mathbf{A}(\mathbf{B}^{(0)})^{-1}\mathbf{A}^T\mathbf{F}^{(k)}$ in total time $\tilde{O}(sd^{\omega-1} + \alpha dr\left(\frac{\alpha}{r}\right)^{\omega-2})$.

For the second term in (III.4), we can precompute $\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}$ in $\tilde{O}(sd^{\omega-1})$ time. Therefore, using Lemma 13 we can maintain $\left(\left(\mathbf{E}^{(k)}\right)^{1/2}-\left(\mathbf{D}^{(0)}\right)^{1/2}\right)\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}$ in total time $\tilde{O}\left(\alpha dr\left(\frac{\alpha}{r}\right)^{\omega-2}\right)$ and by Theorem 10 we can maintain $\mathbf{\Pi}\left(\left(\mathbf{E}^{(k)}\right)^{1/2}-\left(\mathbf{D}^{(0)}\right)^{1/2}\right)\mathbf{A}\left(\mathbf{B}^{(0)}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}$ in the same amount of time.

For the last two terms in (III.4), we use Lemma 13 on $\left(\left(\mathbf{E}^{(k)}\right)^{1/2}-\left(\mathbf{D}^{(0)}\right)^{1/2}\right)\mathbf{A}\mathbf{C}^T\Delta^{(k)}$, $\Delta^{(k)}\mathbf{C}\mathbf{A}^T\mathbf{F}^{(k)}$ and $\mathbf{\Pi}\left(\mathbf{D}^{(0)}\right)^{1/2}\mathbf{A}\mathbf{C}^T\Delta^{(k)}$ all in total time $\tilde{O}\left(\alpha dr\left(\frac{\alpha}{r}\right)^{\omega-2}\right)$. Note that all of those matrices including $\mathbf{V}^{(k)}$ are essentially $\tilde{O}(\alpha)\times\tilde{O}(\alpha)$ matrices if we discard the zero rows and columns. So, having those matrices explicitly computed, we can compute the last two terms in an additional of $\tilde{O}(\alpha^\omega)$ time per iteration.

Finally computing $\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}$ only requires an additional $\tilde{O}(\alpha^\omega)$ time per-iteration. Hence, the total cost of maintaining $\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}$ is

$$\tilde{O}\left(\alpha dr\left(\frac{\alpha}{r}\right)^{\omega-2}+sd^{\omega-1}+r\alpha^\omega\right).$$

Using (III.1) and (III.2), we have shown how to approximate $\left(\mathbf{B}^{(k)}\right)^{-1}$. Now, we show how to compute a better approximation. Recall from (III.1) that

$$\left(\mathbf{B}^{(k)}\right)^{-1}=\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}+\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}\left(\mathbf{M}^{(k)}\right)^{-1}\mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}.$$

Using Lemma 14 as we have argued, we can apply the first term $\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}$ exactly to a vector in $\tilde{O}(d^2)$ time. The only difficulty in applying the second term to a vector comes from the term $\left(\mathbf{M}^{(k)}\right)^{-1}$. Using (III.3), we have

$$\mathbf{M}^{(k)}\approx_1\left(\mathbf{F}^{(k)}\right)^\dagger+\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}$$

and hence

$$\left(\mathbf{M}^{(k)}\right)^{-1}\approx_1\left(\left(\mathbf{F}^{(k)}\right)^\dagger+\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}\right)^{-1}.$$

Since computing $\left(\left(\mathbf{F}^{(k)}\right)^\dagger+\left(\mathbf{N}^{(k)}\right)^T\mathbf{\Pi}^T\mathbf{\Pi}\mathbf{N}^{(k)}\right)^{-1}$ only requires $\tilde{O}(\alpha^\omega)$ time and (III.2) shows that we can apply $\mathbf{M}^{(k)}$ to a vector exactly in $\tilde{O}(d^2)$ time, by Lemma 5 we have a symmetric matrix $\mathbf{M}_\epsilon^{(k)}\approx_\epsilon\mathbf{M}^{(k)}$ such that we can apply $\left(\mathbf{M}_\epsilon^{(k)}\right)^{-1}$ to a vector in $\tilde{O}(d^2\log(\epsilon^{-1}))$ for any $\epsilon>0$. Hence, we obtain

$$\mathbf{\Lambda}_\epsilon^{(k)}\approx_\epsilon\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{F}^{(k)}\left(\mathbf{M}^{(k)}\right)^{-1}\mathbf{F}^{(k)}\mathbf{A}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}$$

such that we can apply $\mathbf{\Lambda}_\epsilon^{(k)}$ to a vector in $\tilde{O}(d^2\log(\epsilon^{-1}))$ time. All that remains is to compute what value of $\epsilon$ is needed for this to be a spectral approximation to the $\left(\mathbf{B}^{(k)}\right)^{-1}$.

Recall that by the assumption $\beta^{-1}\mathbf{B}^{(0)}\preceq\mathbf{B}^{(k)}\preceq\beta\mathbf{B}^{(0)}$. As we mentioned in the beginning, we replaced $\vec{d}^{(k)}$ with $\vec{d}^{(k)}+\frac{1}{10\beta}\vec{d}^{(0)}$ and compute a constant spectral approximation to the new $\mathbf{B}^{(k)}$ that will suffice for the theorem statement. Consequently

$$\mathbf{\Lambda}^{(k)}\stackrel{\text{def}}{=}\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}-\left(\mathbf{B}^{(k)}\right)^{-1}\preceq 10\beta\left(\mathbf{B}^{(0)}\right)^{-1}$$

Furthermore, since $\vec{f}^{(k)}\geq 0$ we have $\mathbf{0}\preceq\mathbf{\Lambda}^{(k)}$ and therefore

$$\mathbf{\Lambda}^{(k)}-10\epsilon\beta\left(\mathbf{B}^{(0)}\right)^{-1}\preceq\mathbf{\Lambda}_\epsilon^{(k)}\preceq\mathbf{\Lambda}^{(k)}+10\epsilon\beta\left(\mathbf{B}^{(0)}\right)^{-1}.$$

Using the assumption $\beta^{-1}\mathbf{B}^{(0)}\preceq\mathbf{B}^{(k)}\preceq\beta\mathbf{B}^{(0)}$ again, we have

$$\mathbf{\Lambda}^{(k)}-10\epsilon\beta^2\left(\mathbf{B}^{(k)}\right)^{-1}\preceq\mathbf{\Lambda}_\epsilon^{(k)}\preceq\mathbf{\Lambda}^{(k)}+10\epsilon\beta^2\left(\mathbf{B}^{(k)}\right)^{-1}.$$

Picking $\epsilon=O(\frac{1}{20\beta^2})$, we have a matrix $\left(\mathbf{A}^T\mathbf{E}^{(k)}\mathbf{A}\right)^{-1}-\mathbf{\Lambda}_\epsilon^{(k)}\approx_{O(1)}\left(\mathbf{B}^{(k)}\right)^{-1}$ which we can apply in $\tilde{O}(d^2\log(\beta))$ time. Furthermore, again since $\beta^{-1}\mathbf{B}^{(0)}\preceq\mathbf{B}^{(k)}\preceq\beta\mathbf{B}^{(0)}$ we see that our replacement of d $\vec{d}^{(k)}$ with $\vec{d}^{(k)}+\frac{1}{10\beta}\vec{d}^{(0)}$only affected our approximation quality by a constant. ■

## IV. An algorithm for the $\sigma$ Stable Case

In this section, we provide our algorithm for solving the inverse maintenance problem under the $\sigma$ stability assumption. The central result of this section is the following:

**Theorem 15.** *Suppose that the inverse maintenance problem satisfies the $\sigma$ stability assumption. Then Algorithm 3 maintains a $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^2)$-time solver with high probability in total time $\tilde{O}(d^\omega + r(\mathrm{nnz}(\mathbf{A}) + d^2))$ where $r$ is the number of rounds.*

To prove this we first provide a technical lemma showing that leverage scores are stable in leverage score number assuming $\sigma$ stability (See Section IV-A). Using this lemma we then prove the Theorem 15 (See Section IV-B). This proof will assume that the randomness we use to maintain our solvers is independent from the output of our solvers. In Section V we show how to make this assumption hold.

### A. Leverage Scores are Stable under $\sigma$ Stability

Here we show that leverage scores are stable in the leverage score norm assuming $\sigma$ stability. This technical lemma, Lemma 16, is crucial to showing that we do not need to perform too many low-rank updates on our sparsifier in our solution to the inverse maintenance problem under the $\sigma$ stability assumption. (See Section I-C for more intuition.)

**Lemma 16.** *For all $\mathbf{A} \in \mathbb{R}^{n \times d}$ and any vectors $\vec{x}, \vec{y} \in \mathbb{R}^n_{>0}$ such that $\left\| \ln(\vec{x}) - \ln(\vec{y}) \right\|_\infty \leq \epsilon$, we have*

$$\left\| \ln \vec{\sigma}_{\mathbf{A}}(\vec{x}) - \ln \vec{\sigma}_{\mathbf{A}}(\vec{y}) \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{x})} \leq e^\epsilon \left\| \ln \vec{x} - \ln \vec{y} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{x})}.$$

*Proof:* For $t \in [0, 1]$, let $\ln \vec{\theta}_t$ denote a straight line from $\ln \vec{x}$ to $\ln \vec{y}$ with $\vec{\theta}_0 = \vec{x}$ and $\vec{\theta}_1 = \vec{y}$ or equivalently let $\vec{\theta}_t \stackrel{\text{def}}{=} \exp(\ln x + t(\ln y - \ln x)))$. Since $\left\| \ln(\vec{x}) - \ln(\vec{y}) \right\|_\infty \leq \epsilon$ we have for all $i \in [n]$

$$\begin{aligned} \vec{\sigma}_{\mathbf{A}}(\vec{x})_i &= \vec{1}_i^T \sqrt{\mathbf{X}} \mathbf{A} \left( \mathbf{A}^T \mathbf{X} \mathbf{A} \right)^{-1} \mathbf{A}^T \sqrt{\mathbf{X}} \vec{1}_i \\ &\approx_{2\epsilon} \vec{1}_i^T \sqrt{\Theta} \mathbf{A} \left( \mathbf{A}^T \Theta \mathbf{A} \right)^{-1} \mathbf{A}^T \sqrt{\Theta} \vec{1}_i \\ &= \vec{\sigma}_{\mathbf{A}}(\vec{\theta}_i) \end{aligned} \tag{IV.1}$$

Consequently, for all $\vec{z}$, we have $\left\| \vec{z} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t)} \leq e^\epsilon \left\| \vec{z} \right\|_{\vec{\sigma}_{\mathbf{A}}}$ and by Jensen's inequality we have

$$\left\| \ln \vec{\sigma}_{\mathbf{A}}(\vec{x}) - \ln \vec{\sigma}_{\mathbf{A}}(\vec{y}) \right\|_{\sigma_{\mathbf{A}}(\vec{x})} \leq \left\| \int_0^1 \left( \frac{d}{dt} \ln \vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t) \right) dt \right\|_{\sigma_{\mathbf{A}}(\vec{x})} \leq e^\epsilon \cdot \int_0^1 \left\| \frac{d}{dt} \ln \vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t) \right\|_{\sigma_{\mathbf{A}}(\vec{\theta}_t)} dt \tag{IV.2}$$

Now, for all $z \in \mathbb{R}^n_{>0}$ let $\mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{z}))$ denote the Jacobian matrix $\left[ \frac{\partial \ln \vec{\sigma}_{\mathbf{A}}(\vec{z})_i}{\partial \ln \vec{z}_j} \right]_{ij} = \left[ \frac{\partial \ln \vec{\sigma}_{\mathbf{A}}(\vec{z})_i}{\partial \vec{z}_j} z_j \right]_{ij}$ and suppose that for all $\vec{z}$ and $\vec{u}$ we have

$$\left\| \mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{z})) \vec{u} \right\|_{\vec{\sigma}_{\mathbf{A}}} \leq \left\| \vec{u} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{z})}. \tag{IV.3}$$

Then, using IV.1 as well as the definition of $\vec{\theta}_t$, we have

$$\int_0^1 \left\| \frac{d}{dt} \ln \vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t) \right\|_{\sigma_{\mathbf{A}}(\vec{\theta}_t)} dt = \int_0^1 \left\| \mathbf{J}_{\ln \vec{\theta}_t}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t)) \left( \frac{d}{dt} \ln \vec{\theta}_t \right) \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{\theta}_t)} \leq e^\epsilon \left\| \ln \vec{x} - \ln \vec{y} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{x})}. \tag{IV.4}$$

All that remains is to prove (IV.3). For this, we first note that in [17, Ver 1, Lemma 36] we showed that $\mathbf{J}_{\vec{z}}(\vec{\sigma}_{\mathbf{A}}(\vec{z})) = (\Sigma_{\mathbf{A}}(\vec{z}) - \mathbf{M}) \mathbf{Z}^{-1}$ where $\mathbf{M}_{ij}(\vec{z}) \stackrel{\text{def}}{=} \left( \sqrt{\mathbf{Z}} \mathbf{A} (\mathbf{A}^T \mathbf{Z} \mathbf{A})^{-1} \mathbf{A}^T \sqrt{\mathbf{Z}} \right)^2$. Consequently $\mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{z})) = \Sigma_{\mathbf{A}}(\vec{z})^{-1} \left( \Sigma_{\mathbf{A}}(\vec{z}) - \mathbf{M}(\vec{z}) \right)$. Now note that

$$\begin{aligned} \sum_i \mathbf{M}_{ij} &= \sum_i \left( \sqrt{\mathbf{Z}} \mathbf{A} (\mathbf{A}^T \mathbf{Z} \mathbf{A})^{-1} \mathbf{A}^T \sqrt{\mathbf{Z}} \right)^2_{ij} \\ &= \left\langle \sqrt{\mathbf{Z}} \mathbf{A} (\mathbf{A}^T \mathbf{Z} \mathbf{A})^{-1} \mathbf{A}^T \sqrt{\mathbf{Z}} \vec{1}_j, \sqrt{\mathbf{Z}} \mathbf{A} (\mathbf{A}^T \mathbf{Z} \mathbf{A})^{-1} \mathbf{A}^T \sqrt{\mathbf{Z}} \vec{1}_j \right\rangle \\ &= \left( \sqrt{\mathbf{Z}} \mathbf{A} (\mathbf{A}^T \mathbf{Z} \mathbf{A})^{-1} \mathbf{A}^T \sqrt{\mathbf{Z}} \right)_{jj} = \left( \vec{\sigma}_{\mathbf{A}}(\vec{z}) \right)_j. \end{aligned}$$

Consequently, $\Sigma_{\mathbf{A}}(\vec{z}) - \mathbf{M}$ is a symmetric diagonally dominant matrix and therefore $\Sigma_{\mathbf{A}}(\vec{z}) \succeq \Sigma_{\mathbf{A}}(\vec{z}) - \mathbf{M} \succeq \mathbf{0}$ and $\mathbf{0} \preceq \Sigma_{\mathbf{A}}(\vec{z})^{1/2} \mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{z})) \Sigma_{\mathbf{A}}(\vec{z})^{-1/2} \preceq \mathbf{I}$. Using this fact, we have that for all $\vec{z} \in \mathbb{R}^n_{>0}$ and $\vec{u} \in \mathbb{R}^n$

$$\left\| \mathbf{J}_{\ln \vec{z}}(\ln \vec{\sigma}_{\mathbf{A}}(\vec{z})) \vec{u} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{z})} = \left\| \Sigma_{\mathbf{A}}(\vec{z})^{1/2} \vec{u} \right\|_{\left( \Sigma_{\mathbf{A}}(\vec{z})^{-1/2} (\Sigma_{\mathbf{A}}(\vec{z}) - \mathbf{M}) \Sigma_{\mathbf{A}}(\vec{z})^{-1/2} \right)^2} \leq \left\| \vec{u} \right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{z})}$$

and this proves (IV.3). Combining (IV.2) and (IV.4) yields the result. ■

### B. Algorithm for $\sigma$ Stability

Here we prove Theorem 15. The full pseudocode for our algorithm, with the exception of how we compute leverage scores and maintain the inverse of $\mathbf{A}\mathbf{H}^{(k)}\mathbf{A}$ can be seen in Algorithm 3.

**Algorithm 3:** Algorithm for $\sigma$ Stability Assumption

**Input:** Initial point $\vec{d}^{(0)} \in \mathbb{R}^n_{>0}$.
Set $\vec{d}^{(old)} := \vec{d}^{(0)}$ and $\gamma \stackrel{\text{def}}{=} 1000 c_s \log d$ where $c_s$ defined in Lemma 7.
Use Lemma 8 to find $\sigma^{(apr)}$ such that $0.99\sigma_i^{(apr)} \leq \sigma(\vec{d}^{(0)})_i \leq 1.01\sigma_i^{(apr)}$.
For each $i \in [n]$ : let $h_i^{(0)} := d_i / \min\{1, \gamma \cdot \sigma_i^{(apr)}\}$ with probability $\min\{1, \gamma \cdot \sigma_i^{(apr)}\}$ and is set to 0 otherwise.
$\mathbf{Q}^{(0)} \stackrel{\text{def}}{=} \mathbf{A}^T \mathbf{H}^{(0)} \mathbf{A}$.
Let $\mathbf{K}^{(0)}$ be an approximate inverse of $\mathbf{Q}^{(0)}$ computed using Theorem 11.
**Output:** A $\tilde{O}(d^2 + \text{nnz}(\mathbf{A}))$-time linear solver for $\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$ (using Theorem 12 on $\mathbf{K}^{(0)}$).
**for** *each round* $k \in [r]$ **do**
  **Input:** Current point $\vec{d}^{(k)} \in \mathbb{R}^n_{>0}$.
  Use Lemma 8 and the solver from the previous round to find $\sigma^{(apr)}$ such that $0.99\sigma_i^{(apr)} \leq \sigma(\vec{d}^{(k)})_i \leq 1.01\sigma_i^{(apr)}$.
  **for** *each coordinate* $i \in [n]$ **do**
    **if** *either* $0.9\sigma_i^{(old)} \leq \sigma_i^{(apr)} \leq 1.1\sigma_i^{(old)}$ *or* $0.9d_i^{(old)} \leq d_i^{(k)} \leq 1.1d_i^{(old)}$ *is violated* **then**
      $d_i^{(old)} := d_i^{(k)}$.
      $\sigma_i^{(old)} := \sigma_i^{(apr)}$.
      $h_i^{(k)} := d_i^{(k)} / \min\{1, \gamma \cdot \sigma_i^{(apr)}\}$ with probability $\min\{1, \gamma \cdot \sigma_i^{(apr)}\}$ and is set to 0 otherwise.
    **else**
      $h_i^{(k)} := h_i^{(k-1)}$.
    **end**
  **end**
  $\mathbf{Q}^{(k)} \stackrel{\text{def}}{=} \mathbf{A}^T \mathbf{H}^{(k)} \mathbf{A}$.
  Let $\mathbf{K}^{(k)}$ be an approximate inverse of $\mathbf{Q}^{(k)}$ computed using Theorem 11.
  **Output:** A linear $\tilde{O}(d^2 + \text{nnz}(\mathbf{A}))$-time solver for $\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$ (using Theorem 12 on $\mathbf{K}^{(k)}$).
**end**

First, we bound the number of coordinates $\mathbf{H}$ that will change during the algorithm.

**Lemma 17.** *Suppose that the changes of $\vec{d}$ and the error occurred in computing $\vec{\sigma}$ is independent of our sampled matrix. Under the $\sigma$ stability guarantee, during first $r$ iterations of Algorithm 3, the expected number of coordinates changes in $\mathbf{H}^{(k)}$ over all $k$ is $O(r^2 \log d)$.*

*Proof:* Since our error in computing $\sigma$ is smaller than the re-sampling threshold on how much change of $\sigma$, the re-sampling process for the $i^{th}$ row happens only when either $\sigma_{\mathbf{A}}(\vec{d})_i$ or $d_i$ changes by more than a multiplicative constant. In order for re-sampling to affect $\mathbf{A}^T\mathbf{H}\mathbf{A}$, it must be the case that it is either currently in $\mathbf{A}^T\mathbf{H}\mathbf{A}$ or about to be put in $\mathbf{A}^T\mathbf{H}\mathbf{A}$. However, since whenever re-sampling occurs the resampling probability has changed by at most a multiplicative constant, we have that both these events happen with probability $O(\gamma \cdot \sigma_{\mathbf{A}}(\vec{d})_i)$ using the independence between $\vec{d}$ and the approximate $\vec{\sigma}$. By union bound we have that the probability of sampling row $i$ changing the matrix $\mathbf{A}^T\mathbf{H}\mathbf{A}$ is $O(\sigma_{\mathbf{A}}(\vec{d})_i \log(d))$.

Observe that whenever we re-sampled the $i^{th}$ row, either $\sigma_{\mathbf{A}}(\vec{d})_i$ or $d_i$ has changed by more than a multiplicative constant. Let $k_1$ be the last iteration we re-sampled the $i^{th}$ row and let $k_2$ be the current iteration. Then, we know that $\sum_{k=k_1}^{k_2-1} \left| \ln \sigma_{\mathbf{A}}(\vec{d}^{(k+1)})_i - \ln \sigma_{\mathbf{A}}(\vec{d}^{(k)})_i \right| = \Omega(1)$ or $\sum_{k=k_1}^{k_2-1} \left| \ln \vec{d}_i^{(k+1)} - \ln \vec{d}_i^{(k)} \right| \geq \Omega(1)$. Since there are only $r$ steps, we have $|k_2 - k_1| \leq r$ and hence

$$\sum_{k=k_1}^{k_2-1} \left( \ln \sigma_{\mathbf{A}}(\vec{d}^{(k+1)})_i - \ln \sigma_{\mathbf{A}}(\vec{d}^{(k)})_i \right)^2 = \Omega\left(\frac{1}{r}\right) \text{ or } \sum_{k=k_1}^{k_2-1} \left( \ln \vec{d}_i^{(k+1)} - \ln \vec{d}_i^{(k)} \right)^2 = \Omega\left(\frac{1}{r}\right).$$

Since $\sigma_{\mathbf{A}}(\vec{d})_i$ does not change more by a constant between re-sample (by $\sigma$-stability assumption), we have

$$\sum_{k=k_1}^{k_2-1} \sigma_{\mathbf{A}}(\vec{d}^{(k)})_i \left(\ln \sigma_{\mathbf{A}}(\vec{d}^{(k+1)})_i - \ln \sigma_{\mathbf{A}}(\vec{d}^{(k)})_i\right)^2 = \Omega\left(\frac{\sigma_{\mathbf{A}}(\vec{d}^{(k_2)})_i}{r}\right) \text{ or}$$

$$\sum_{k=k_1}^{k_2-1} \sigma_{\mathbf{A}}(\vec{d}^{(k)})_i \left(\ln \vec{d}_i^{(k+1)} - \ln \vec{d}_i^{(k)}\right)^2 = \Omega\left(\frac{\sigma_{\mathbf{A}}(\vec{d}^{(k_2)})_i}{r}\right).$$

In summary, re-sampling the $i^{th}$ row indicates that the sum of the $\sigma$ norm square of the changes of either $\ln \sigma_{\mathbf{A}}$ or $\ln d$ is at least $\sigma_{\mathbf{A}}(\vec{d})_i/r$ and with $O(\sigma_{\mathbf{A}}(\vec{d})_i \log d)$ probability, the sampled matrix is changed. Since $\sum_k \left\| \ln \vec{d}^{(k+1)} - \ln \vec{d}^{(k)} \right\|^2_{\sigma_{\mathbf{A}}(\vec{d}^{(k)})} \leq O(r)$ and by Lemma 16 $\sum_k \left\| \ln \sigma_{\mathbf{A}}(\vec{d}^{(k+1)}) - \ln \sigma_{\mathbf{A}}(\vec{d}^{(k)}) \right\|^2_{\sigma_{\mathbf{A}}(\vec{d}^{(k)})} \leq O(r)$ we have the desired result. ∎

We now everything we need to prove our main theorem assuming that the changes of $\vec{d}$ and the error occurred in computing $\sigma$ is independent of our sampled matrix. (In Section V we show how to drop this assumption.)

*Proof of Theorem 15:* Note that by design in each iteration $k \in [r]$ we have that $\mathbf{D}^{(old)} \approx_{0.2} \mathbf{D}^{(k)}$, $\mathbf{\Sigma}^{(old)} \approx_{0.2} \mathbf{\Sigma}(\vec{d}^{(k)})$ where $\mathbf{\Sigma} \stackrel{\text{def}}{=} \mathbf{diag}(\sigma)$. Thus, we see that the sample probability was chosen precisely so that we can apply Lemma 7. Hence, we have $\mathbf{A}^T\mathbf{H}^{(k)}\mathbf{A} \approx_{0.1} \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}$. Thus, the algorithm is correct, it simply remains to bound the running time.

To maintain inverse of $\mathbf{A}^T\mathbf{H}^{(k)}\mathbf{A}$, we can simply apply Theorem 11. By Lemma 7, we know that with high probability $\text{nnz}(\mathbf{H}^{(0)}) \leq \tilde{O}(d)$. By Lemma 17 we know there are only $\tilde{O}(r^2)$ coordinate changes during the algorithm in expectation. Therefore, we can use Theorem 11 on $\mathbf{A}^T\mathbf{H}^{(k)}\mathbf{A}$ with $\alpha = \tilde{O}(r^2)$ and $s = \tilde{O}(d)$. Hence, the average cost of maintain a linear $\tilde{O}(\text{nnz}(\mathbf{A}) + d^2)$-time solver of $\left(\mathbf{A}^T\mathbf{H}^{(k)}\mathbf{A}\right)^{-1}$ is $\tilde{O}\left(\frac{d^\omega}{r} + dr^\omega + r^{2\omega}\right)$. Similar to Theorem 12 , we restart the algorithm every $r = (nd^{\omega-1})^{\frac{1}{2\omega+1}}$ and see that the total cost of maintenance is $\tilde{O}(d^\omega + rd^{\frac{2\omega^2}{2\omega+1}})$. Since $\omega \leq 1 + \sqrt{2}$, we have the total maintenance cost is $\tilde{O}\left(d^\omega + rd^2\right)$.

Thus, all the remains is to bound the cost of computing $\vec{\sigma}^{(k)}$. However, since by the $\sigma$stability assumption $\left\| \log(\vec{d}_{k+1}) - \log(\vec{d}_k) \right\|_\infty \leq 0.1$ we know that $\mathbf{D}^{(k)} \approx_{0.1} \mathbf{D}^{(k+1)}$ and thus $\mathbf{Q}^{(k)} \approx_{0.2} \mathbf{A}^T\mathbf{D}^{(k+1)}\mathbf{A}$. Thus, using Lemma 8 we can compute $\vec{\sigma}^{(k)}$ for all $k \geq 2$ in $\tilde{O}(\text{nnz}(\mathbf{A}) + d^2 \log \beta)$ time within 0.01 multiplicative factor. Furthermore, using this same Lemma and fast matrix multiplication, we can compute $\vec{\sigma}^{(1)}$ in $\tilde{O}(\text{nnz}(\mathbf{A}) + d^\omega)$ time; therefore, we have our result.

Note that Lemma 17 assume that the changes of $\vec{d}$ and the error occurred in computing $\sigma$ is independent of our sampled matrix. Given any linear solver, Theorem 19 in Section V shows how to construct a solver that has same running time up to $\tilde{O}(1)$ factor and is statistically indistinguishable from the true solver $\left(\mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A}\right)^{-1}$ and thus circumvent this issue; thereby completing the proof. ∎

## V. Hiding Randomness in Linear System Solvers

In many applications (see Section VII), the input to a particular round of the inverse maintenance problem depends on our output in the previous round. However, our solution to the inverse maintenance problem is randomized and if the input to the inverse maintenance problem was chosen adversarially based on this randomness, this could break the analysis of our algorithm. Moreover, even within our solution to the inverse maintenance problem we needed to solve linear systems and if the output of these linear systems was adversarially correlated with our randomized computation our analysis would break (see Section IV)

In this section, we show how to fix both these problems and hide the randomness we use to approximately solve linear system. We provide a general transformation, `NoisySolver` in Algorithm 4, which turns a linear solver for $\mathbf{A}^T\mathbf{A}$ into a nonlinear solver for $\mathbf{A}^T\mathbf{A}$ that with high probability is indistinguishable from an exact solver for $\mathbf{A}^T\mathbf{A}$ plus a suitable Gaussian noise. The algorithm simply solves the desired linear system using the input solver and then add a suitable Gaussian noise.

We break the proof that this works into two parts. First, in Lemma 18 we show that `NoisySolver`, is in fact a solver and then in Theorem 19 we show that with high probability it is indistinguishable from an exact solver plus a Gaussian noise.

**Lemma 18.** *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ and let $\mathtt{S}$ be a linear $\mathcal{T}$-time solver for $\mathbf{A}^T\mathbf{A}$. For all $\vec{b} \in \mathbb{R}^n$ and $\epsilon \in (0, 1/2)$, the algorithm $\mathtt{NoisySolver}(\vec{b}, \epsilon)$ is a $(\text{nnz}(\mathbf{A}) + \mathcal{T})$-time solver for $\mathbf{A}^T\mathbf{A}$.*

**Algorithm 4:** $\texttt{NoisySolver}(\vec{b}, \epsilon)$

**Input:** a linear $\mathcal{T}$-time solver $\mathtt{S}$ of $\mathbf{A}^T\mathbf{A}$, vector $\vec{b} \in \mathbb{R}^n$, and accuracy $\epsilon \in (0, 1/2)$.
$\vec{y}_1 := \mathtt{S}(\vec{b}, \epsilon_1)$ where $\epsilon_1 = \epsilon \left(32n^7\right)^{-2}$.
Let $\vec{\eta} \in \mathbb{R}^n$ be sampled randomly from the normal distribution with mean $\vec{0}$ and covariance $\mathbf{I}$.
$\vec{y}_2 := \mathtt{S}(\mathbf{A}^T\vec{\eta}, \epsilon_2)$ where $\epsilon_2 = \left(12dn^6\right)^{-2}$.
**Output:** $\vec{y} \stackrel{\text{def}}{=} \vec{y}_1 + \alpha\vec{y}_2$ where $\alpha = \frac{1}{8}\sqrt{\frac{\epsilon}{n}}\left\|\vec{y}_1\right\|_{\mathbf{A}^T\mathbf{A}}$.

*Proof:* By the definition of $\vec{y}$ and the inequality $(a+b)^2 \leq 2a^2 + 2b^2$, we have

$$\left\|\vec{y} - \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \quad \leq \quad 2\left\|\vec{y}_1 - \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 + 2\alpha^2\left\|\vec{y}_2\right\|_{\mathbf{A}^T\mathbf{A}}^2.$$

To bound the first term, recall that by the definition of $\mathtt{S}$

$$\left\|\vec{y}_1 - \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \leq \epsilon_1\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2.$$

To bound the second term, we note that by the definition of $\vec{\eta}$, $\left\|\vec{\eta}\right\|_2^2$ follows $\chi^2$-distribution with $n$ degrees of freedom. It is known that [16, Lem 1] for all $t > 0$,

$$\mathbb{P}\left(\left\|\vec{\eta}\right\|_2^2 > n + 2\sqrt{nt} + 2t\right) \leq \exp(-t) \quad .$$

Hence, with high probability in $n$, $\left\|\vec{\eta}\right\|_2^2 < 2n$. Using this and the definition of $\mathtt{S}$ yields

$$\begin{aligned}\left\|\vec{y}_2\right\|_{\mathbf{A}^T\mathbf{A}}^2 &\leq 2\left\|\vec{y}_2 - \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\eta}\right\|_{\mathbf{A}^T\mathbf{A}}^2 + 2\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\eta}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \\ &\leq 2\left(1+\epsilon_2\right)\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\vec{\eta}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \leq 4\left\|\vec{\eta}\right\|_2^2 \leq 8n.\end{aligned}$$

where we used that $\epsilon_2 \leq 1$ and $\left\|\mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\right\|_2 \leq 1$. Using that $\epsilon_1 \leq 1$ and a similar proof, we have

$$\alpha^2 = \frac{\epsilon}{64n}\left\|\vec{y}_1\right\|_{\mathbf{A}^T\mathbf{A}}^2 \leq \frac{\epsilon}{32n}\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2$$

Consequently, with high probability in $n$,

$$\left\|\vec{y} - \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \leq 2\epsilon_1\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 + 16\alpha^2 n \leq \epsilon\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}^2 \quad .$$

■

**Theorem 19.** *Let* $\texttt{IdealSolver}(\vec{b}) = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b} + \vec{c}$ *where* $\vec{c}$ *follows the normal distribution with mean* $0$ *and covariance* $\beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}$ *where* $\beta = \frac{1}{8}\sqrt{\frac{\epsilon}{n}}\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}$. *Then, the total variation distance between the outcome of* $\texttt{NoisySolver}$ *and* $\texttt{IdealSolver}$ *is less than* $1/n^3$. *Therefore, any algorithm calls* $\texttt{IdealSolver}$ *less than* $n^2$ *times cannot distinguish between* $\texttt{NoisySolver}$ *and* $\texttt{IdealSolver}$.

*Proof:* Since $\mathtt{S}$ is linear, we have $\vec{y}_2 = \mathbf{Q}_{\epsilon_2}\mathbf{A}^T\vec{\eta}$ for some matrix $\mathbf{Q}_{\epsilon_2}$. Since $\vec{\eta}$ follows the normal distribution with mean $0$ and covariance matrix $\mathbf{I}$, we have

$$\mathbb{E}\left[\vec{y}_2\vec{y}_2^T\right] = \mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbb{E}\left[\vec{\eta}\vec{\eta}^T\right]\mathbf{A}\mathbf{Q}_{\epsilon_2}^T = \mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T$$

Lemma 6 shows that $\mathbf{Q}_{\epsilon_2}^T\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2} \approx_{4\sqrt{\epsilon_2}} \left(\mathbf{A}^T\mathbf{A}\right)^{-1}$ and hence

$$\mathbb{E}\left[\vec{y}_2\vec{y}_2^T\right] \approx_{4\sqrt{\epsilon_2}} \left(\mathbf{A}^T\mathbf{A}\right)^{-1}. \tag{V.1}$$

Since $\vec{y}_2$ is a linear transformation of $\vec{\eta}$, $\vec{y}_2$ follows is given by the normal distribution with mean $0$ and covariance $\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T$, i.e. $\vec{y}_2 \in \mathcal{N}(0, \mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T)$.

Now let, $\vec{y}, \vec{z}$ be the output of $\texttt{NoisySolver}(\vec{b})$ and $\texttt{IdealSolver}(\vec{b})$ respectively; we know that

$$\vec{y} \in \mathcal{N}\left(\vec{y}_1, \alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T\right) \text{ and } \vec{z} \in \mathcal{N}\left(\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}, \beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\right)$$

where $\alpha = \frac{1}{8}\sqrt{\frac{\epsilon}{n}}\left\|\vec{y}_1\right\|_{\mathbf{A}^T\mathbf{A}}$ and $\beta = \frac{1}{8}\sqrt{\frac{\epsilon}{n}}\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}$. To bound $\left\|\vec{y}-\vec{z}\right\|_{\text{tv}}$, Pinsker's inequality shows that $\left\|\vec{y}-\vec{z}\right\|_{\text{tv}} \leq \sqrt{\frac{1}{2}D_{\text{KL}}(\vec{y}||\vec{z})}$ and using an explicit formula for the KL divergence for the normal distribution we in turn have that $\left\|\vec{y}-\vec{z}\right\|_{\text{tv}}$ is bounded by

$$\frac{1}{2}\sqrt{\text{tr}\left(\left(\frac{\alpha}{\beta}\right)^2\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T\right) + \left\|\vec{y}_1-\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|^2_{\beta^{-2}\mathbf{A}^T\mathbf{A}} - d + \ln\left(\frac{\det\beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}}{\det\alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T}\right)}$$

Hence, we need to prove $\alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T \approx \beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}$ and $\vec{y}_1 \approx \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}$

First, we first prove $\vec{y}_1 \approx \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}$:

$$\left\|\vec{y}_1-\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|^2_{\beta^{-2}(\mathbf{A}^T\mathbf{A})} \leq \beta^{-2}\epsilon_1\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|^2_{\mathbf{A}^T\mathbf{A}} \leq \frac{64n\epsilon_1}{\epsilon}. \qquad \text{(Definition of S, } \vec{y}_1 \text{ and } \beta)$$

Next, to prove $\alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T \approx \beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}$, we note that by triangle inequality and the definition of S and $\vec{y}_1$ we have

$$(1-\sqrt{\epsilon_1})\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}} \leq \left\|\vec{y}_1\right\|_{\mathbf{A}^T\mathbf{A}} \leq (1+\sqrt{\epsilon_1})\left\|\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\vec{b}\right\|_{\mathbf{A}^T\mathbf{A}}.$$

Therefore by the definition of $\alpha$ and $\beta$ we have $\left(1-3\sqrt{\epsilon_1}\right) \leq \frac{\alpha^2}{\beta^2} \leq \left(1+3\sqrt{\epsilon_1}\right)$. Using (V.1) then yields that

$$\alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T \approx_{4\sqrt{\epsilon_2}+4\sqrt{\epsilon_1}} \beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}.$$

Therefore, we have

$$\begin{aligned}\text{tr}\left((\frac{\alpha}{\beta})^2\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T\right) &= \text{tr}\left((\frac{\alpha}{\beta})^2\left(\mathbf{A}^T\mathbf{A}\right)^{1/2}\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T\left(\mathbf{A}^T\mathbf{A}\right)^{1/2}\right)\\ &\leq de^{4\sqrt{\epsilon_2}+4\sqrt{\epsilon_1}} \leq d+8\sqrt{\epsilon_2}d+8\sqrt{\epsilon_1}d\end{aligned}$$

and

$$\begin{aligned}\ln\left(\frac{\det\beta^2\left(\mathbf{A}^T\mathbf{A}\right)^{-1}}{\det\alpha^2\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T}\right) &= \ln\det\left((\frac{\alpha}{\beta})^2\left(\mathbf{A}^T\mathbf{A}\right)^{1/2}\mathbf{Q}_{\epsilon_2}\mathbf{A}^T\mathbf{A}\mathbf{Q}_{\epsilon_2}^T\left(\mathbf{A}^T\mathbf{A}\right)^{1/2}\right)^{-1}\\ &\leq d\ln\left(e^{4\sqrt{\epsilon_2}+4\sqrt{\epsilon_1}}\right) = 4\sqrt{\epsilon_2}d+4\sqrt{\epsilon_1}d.\end{aligned}$$

Combining inequalities above and using our choice of $\epsilon_1$ and $\epsilon_2$ we have

$$\left\|\vec{y}-\vec{z}\right\|_{\text{tv}} \leq \frac{1}{2}\sqrt{12\sqrt{\epsilon_2}d+12\sqrt{\epsilon_1}d+64n\frac{\epsilon_1}{\epsilon}} \leq 1/n^3.$$

■

## VI. Rows Insertion and Removal

For some applications we need to solve the inverse maintenance problem when the matrix $\mathbf{A}$ might change between rounds. In particular some rows of $\mathbf{A}$ might be entirely removed or some new rows might be added. For example, in each iteration of cutting plane methods, a new constraint is added to the current polytope $\{\mathbf{A}\vec{x} \geq \vec{b}\}$; each iteration of quasi newton methods, the current approximate Hessian is updated by a rank 1 matrix. If the matrix is updated only by a rank 1 matrix each iteration, the inverse can be updated efficiently using explicit formula. However, it is less obvious when the matrix $\mathbf{A}$ and the diagonal $\mathbf{D}$ can be changed by a high rank matrix.

Here we formally define the more general set of assumptions under which we would like to solve.

**Definition 20** ($K$ Stability Assumption)**.** The inverse maintenance problem satisfies the $K$ stability assumption if

1) A row of $\mathbf{A}$ is revealed to the algorithm at round $k$ if and only if the corresponding entry in $\vec{d}^{(k)}$ is non-zero.
2) For each $k \in [r]$, we have either
   a) $\left\|\log(\vec{d}^{(k)})-\log(\vec{d}^{(k-1)})\right\|_{\vec{\sigma}_{\mathbf{A}}(\vec{d}^{(k)})} \leq 0.1$ and $\left\|\log(\vec{d}^{(k)})-\log(\vec{d}^{(k-1)})\right\|_{\infty} \leq 0.1$; or
   b) The vectors $\vec{d}^{(k)}$ and $\vec{d}^{(k-1)}$ differ in at most $K$ coordinates.
3) $\beta^{-1}\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A} \preceq \mathbf{A}^T\mathbf{D}^{(k)}\mathbf{A} \preceq \beta\mathbf{A}^T\mathbf{D}^{(0)}\mathbf{A}$ for $\beta = \text{poly}(n)$.

Some of the previous algorithms for the inverse maintenance, such as [33], rely on the assumption that the entire matrix $\mathbf{A}$ is given explicitly. In particular these algorithms perform precomputation on the entirety of $\mathbf{A}$ that they use to decrease the amortize cost of later steps. In the full version, we show that the Algorithm 3 we proposed does not have this drawback and can be easily modified to solve this version of inverse maintenance problem under the $K$ stability assumption for a fairly large $K$.

**Theorem 21.** *Suppose the inverse maintenance problem satisfies the $K$ stability assumption for $K \leq d^{(3-\omega)/(\omega-1)}$, where $\omega$ is the matrix multiplication constant. Then there is a variant of Algorithm 3 that maintains a $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^2)$-time solver with high probability in total time $\tilde{O}(d^\omega + r(\mathrm{nnz}(\mathbf{A}) + d^2))$ where $r$ is the number of rounds.*

*Remark* 22. Using $\omega < 2.37287$ [7], the above theorem shows how to solve inverse maintenance problem with $d^{0.4568}$ rows addition and removal in amortized $\tilde{O}(d^2)$ time.

## VII. Applications

In this section, we provide multiple applications of our algorithm for solving the inverse maintenance problem.

### A. Linear Programming

Here we show how to use our solution to the inverse maintenance problem under the $\sigma$ stability assumption (See Section IV) to improve the running time of solving a linear program. Our main result is the following:

**Theorem 23.** *Let $\mathbf{A} \in \mathbb{R}^{d \times n}$, $\vec{c}, \vec{l}, \vec{u} \in \mathbb{R}^n$, and $\vec{b} \in \mathbb{R}^d$ for $d \leq n$ and suppose $\vec{x}$ is an interior point of the polytope*

$$S = \left\{ \vec{x} \in \mathbb{R}^n \, : \, \mathbf{A}\vec{x} = \vec{b} \text{ and } l_i \leq x_i \leq u_i \text{ for all } i \in [n] \right\} \quad .$$

*Let $U \stackrel{\text{def}}{=} \max\left( \left\| \frac{\vec{u}-\vec{l}}{\vec{u}-\vec{x}} \right\|_\infty, \left\| \frac{\vec{u}-\vec{l}}{\vec{x}-\vec{l}} \right\|_\infty, \left\| \vec{u} - \vec{l} \right\|_\infty, \left\| \vec{c} \right\|_\infty \right)$. Then, consider the linear program $OPT \stackrel{\text{def}}{=} \min_{\vec{x} \in S} \vec{c}^T \vec{x}$ . In time $\tilde{O}\left( \sqrt{d} \left( \mathrm{nnz}(\mathbf{A}) + d^2 \right) \log\left( U/\epsilon \right) \right)$, we can compute $\vec{y}$ such that $\left\| \mathbf{A}\vec{y} - \vec{b} \right\|_{\mathbf{A}^T \mathbf{S}^{-2} \mathbf{A}} \leq \epsilon$, $l_i \leq x_i \leq u_i$ and $\vec{c}^T \vec{y} \leq OPT + \epsilon$ where $\mathbf{S}$ is a diagonal matrix $\mathbf{S}_{ii} = \min\left( u_i - y_i, y_i - l_i \right)$.*

*Proof:* In [18, ArXiv v2, Thm 28], we showed how to solve linear program of this form by solving a sequence of slowly changing linear systems $\mathbf{A}^T \mathbf{D}^{(k)} \mathbf{A} \vec{x} = \vec{q}^{(k)}$ where $\mathbf{D}^{(k)}$ is a diagonal matrix corresponding to a weighted distance of $\vec{x}_k$ to the boundary of the polytope. In [18, ArXiv v2, Lem 32] we showed that this sequence of linear systems satisfied the $\sigma$ stability assumption. Furthermore, we showed that it suffices to solve these linear systems to $1/\mathsf{poly}(n)$ accuracy. Since, the algorithm consists of $\sqrt{d} \log(U/\epsilon)$ rounds of this algorithm plus additional $\tilde{O}(\mathrm{nnz}(\mathbf{A}) + d^2)$ time per round we have the desired result. ∎

We remark that we can only output an almost feasible point but it is difficult to avoid because finding any point $\vec{x}$ such that $\mathbf{A}\vec{x} = \vec{b}$ takes $O(nd^{\omega-1})$ which is slower than our algorithm when $n \gg d$. Similarly, we have an algorithm for the dual as follows:

**Theorem 24.** *Let $\mathbf{A} \in \mathbb{R}^{n \times d}$ where $d \leq n$. Suppose we have an initial point $\vec{x} \in \mathbb{R}^n$ such that $\mathbf{A}^T \vec{x} = \vec{b}$ and $-1 < x_i < 1$. Then, we can find $\vec{y} \in \mathbb{R}^d$ such that*

$$\vec{b}^T \vec{y} + \left\| \mathbf{A}\vec{y} + \vec{c} \right\|_1 \leq \min_{\vec{y}} \left( \vec{b}^T \vec{y} + \left\| \mathbf{A}\vec{y} + \vec{c} \right\|_1 \right) + \epsilon. \tag{VII.1}$$

*in $\tilde{O}\left( \sqrt{d} \left( \mathrm{nnz}(\mathbf{A}) + d^2 \right) \log\left( U/\epsilon \right) \right)$ time where $U = \max\left( \left\| \frac{2}{1-\vec{x}} \right\|_\infty, \left\| \frac{2}{\vec{x}+1} \right\|_\infty, \left\| \vec{c} \right\|_\infty \right)$.*

*Proof:* It is same as Theorem 23 except we invoke [18, ArXiv v2, Thm 29]. ∎

*Remark* 25. The existence of the interior point in Theorem 24 certifies the linear program has bounded optimum value. Standard tricks can be used to avoid requiring such interior point but may yield a more complicated looking running time (see Appendix E of [17] for instance).

### B. $\ell^1$ and $\ell^\infty$ Regression

The $\ell^p$ regression problem involves finding a vector $\vec{x}$ that minimize $\left\| \mathbf{A}\vec{x} - \vec{c} \right\|_p$ for some $n \times d$ matrix $\mathbf{A}$ and some vector $\vec{c}$. Recently, there has been much research [25], [26], [21], [2], [3] on solving overdetermined problems (i.e. $n \gg d$) as these arises naturally in applications involving large datasets. While there has been recent success on achieving algorithms whose running time is nearly linear input plus something polynomial in $d$, in the case that $p \neq 2$ these algorithms achieve a polynomial dependence on the desired accuracy $\epsilon$ [4]. Here we show how to improve the dependence on $\epsilon$ by paying a multiplicative $\sqrt{d}$ factor.

**Corollary 26.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$, $\vec{c} \in \mathbb{R}^n$, and $p = 1$ or $p = \infty$. There is an algorithm to find $\vec{x}$ such that $\left\|\mathbf{A}\vec{x} - \vec{c}\right\|_p \leq \min_{\vec{y}} \left\|\mathbf{A}\vec{y} - \vec{c}\right\|_p + \epsilon\left\|\vec{c}\right\|_p$ in $\tilde{O}\left(\sqrt{d}\left(\mathrm{nnz}(\mathbf{A}) + d^2\right)\log\left(\epsilon^{-1}\right)\right)$ time.*

*Proof:* The $\ell^1$ case is the special case of Theorem 24 with $\vec{b} = \vec{0}$ and an explicit initial point $\vec{0}$.

For the $\ell^\infty$ case, we consider the following linear program

$$\min_{\vec{x},t}\left(-2n + \frac{1}{2}\right)t + \left\|\mathbf{A}\vec{x} - \vec{c} - t\vec{1}\right\|_1 + \left\|\mathbf{A}\vec{x} - \vec{c} + t\vec{1}\right\|_1. \tag{VII.2}$$

where $\vec{1} \in \mathbb{R}^n$ is the all ones vector. Letting $\mathrm{dist}(a, [-t, t])$ denote the distance from $a$ to the interval $[-t, t]$ (and $|a| + |t|$ if $t \leq 0$), it is easy to see that $|a - t| + |a + t| = dist(a, [-t, t]) + 2t$ and consequently the linear program (VII.2) is equivalent to

$$\min_{\vec{x},t} f(\vec{x}, t) \stackrel{\text{def}}{=} \frac{t}{2} + \sum_{i=1}^{n} \mathrm{dist}\left(\left[\mathbf{A}\vec{x} - c\right]_i, [-t, t]\right).$$

Note that when $t \geq \left\|\mathbf{A}\vec{x} - \vec{c}\right\|_\infty$ we have $f(\vec{x}, t) = \frac{t}{2}$ and when $t \leq \left\|\mathbf{A}\vec{x} - \vec{c}\right\|_\infty$ we have $f(\vec{x}, t) \geq \frac{1}{2}\left\|\mathbf{A}\vec{x} - \vec{c}\right\|_\infty$. Consequently, the linear program (VII.2) is equivalent to $\ell_\infty$ regression. To solve (VII.2), we rewrite it as follows

$$\min_{\vec{x},t}\left(-2n + \frac{1}{2}\right)t + \left\|\begin{pmatrix}\mathbf{A} & -\vec{1}\\ \mathbf{A} & \vec{1}\end{pmatrix}\begin{pmatrix}\vec{x}\\ t\end{pmatrix} - \vec{c}\right\|_1. \tag{VII.3}$$

Since

$$\begin{pmatrix}\mathbf{A} & -\vec{1}\\ \mathbf{A} & \vec{1}\end{pmatrix}^T \vec{y} = \begin{pmatrix}\vec{0}_d\\ -2n\end{pmatrix} \quad \text{for} \quad \vec{y} \stackrel{\text{def}}{=} \begin{pmatrix}\vec{1}\\ -\vec{1}\end{pmatrix},$$

we can use $\frac{2n-\frac{1}{2}}{2n}\vec{y}$ as the initial point for Theorem 24 and apply it to (VII.2) to find $\vec{x}, t$ as desired. ∎

*Remark* 27. We wonder if it is possible to obtain further running time improvements for solving $\ell^p$ regression when $p \notin \{1, 2, \infty\}$.

### C. $\tilde{O}(d)$ Rounding Ellipsoid for Polytopes

For any convex set $K$, we call an ellipsoid $E$ is an $\alpha$-rounding if $E \subset K \subset \alpha E$. It is known that every convex set in $d$ dimension has a $d$-rounding ellipsoid and that such rounding have many applications. (See [14], [34]) For polytopes $K = \{\vec{x} \in \mathbb{R}^d : \mathbf{A}\vec{x} \geq \vec{b}\}$, the previous best algorithm for finding a $(1 + \epsilon)d$ rounding takes time $\tilde{O}(n^{3.5}\epsilon^{-1})$ [14] and $\tilde{O}(nd^2\epsilon^{-1})$ [15]. In the full paper, we showed how to compute an $O(d)$ rounding in time $\tilde{O}(\sqrt{d}(\mathrm{nnz}(\mathbf{A}) + d^2)\log(U))$, which is enough for many applications. Note that this is an at least $O(\sqrt{d})$ improvement over previous results and for the case $\mathbf{A}$ is sparse, this is an $O(d^{1.5})$ improvement. The proof relies on the following technical lemma proved in the full paper, which shows conditions under which we a point in a polytope is the center of a suitable ellipsoid.

**Lemma 28.** *Let $P \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^d : \mathbf{A}\vec{x} \geq \vec{b}\}$ be a polytope for $\mathbf{A} \in \mathbb{R}^{n\times d}$ and $d \leq n$. Furthermore, let $\vec{w} \in \mathbb{R}^n_{>0}$ and let $p(\vec{x}) \stackrel{\text{def}}{=} -\sum_{i=1}^n w_i \ln\left[\mathbf{A}\vec{x} - \vec{b}\right]_i$ for all $\vec{x} \in P$. Now let $\vec{x}_* = \arg\min_{\vec{y}\in\mathbb{R}^d} p(\vec{x})$ and suppose that we have $\gamma \geq 1$ such that $\left|(\mathbf{A}\vec{h})_i/(\mathbf{A}\vec{x}_* - \vec{b})_i\right| \leq \gamma\left\|\vec{h}\right\|_{\nabla^2 p(\vec{x}_*)}$ for all $\vec{h} \in \mathbb{R}^d$. Then the ellipsoid $E = \{\vec{h} \in \mathbb{R}^d : \left\|\vec{h}\right\|_{\nabla^2 p(\vec{x}_*)} \leq 1\}$ satisfies $\vec{x} + \frac{1}{\gamma}E \subset P \subset \vec{x} + \gamma\left\|\vec{w}\right\|_1 E$.*

**Theorem 29.** *Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ for $d \leq n$ and suppose we have an initial point $\vec{x}_0 \in \mathbb{R}^d$ such that $\mathbf{A}\vec{x}_0 \geq \vec{b}$. Then, we can find an ellipsoid $E$ such that $E \subset \{\vec{x} \in \mathbb{R}^d : \mathbf{A}\vec{x} \geq \vec{b}\} \subset 100d \cdot E$ in time $\tilde{O}(\sqrt{d}\left(\mathrm{nnz}(\mathbf{A}) + d^2\right)\log(U))$ where $U = \max\left\{\max_{\mathbf{A}\vec{x}\geq\vec{b}}\left\|\mathbf{A}\vec{x} - \vec{b}\right\|_\infty, \left\|\frac{1}{\mathbf{A}\vec{x}_0 - \vec{b}}\right\|_\infty\right\}$.*

*Proof:* Given any interior point $\vec{x}_0$ such that $\mathbf{A}\vec{x}_0 > \vec{b}$ in [17] we showed how to find a weight $\vec{w}$ such that $\left\|\vec{w}\right\|_1 \leq 3d$ and for any $\vec{h} \in \mathbb{R}^d$ we have

$$\max_{i\in[n]}\left|\frac{[\mathbf{A}\vec{h}]_i}{[\mathbf{A}\vec{x}_* - \vec{b}]_i}\right| \leq 3\left\|\vec{h}\right\|_{\nabla^2 p(\vec{x}_*)} \tag{VII.4}$$

where $p(\vec{x}) = -\sum_{i=1}^{n} w_i \ln \left[\mathbf{A}\vec{x} - \vec{b}\right]_i$ and $\vec{x}_* = \arg\min_{\mathbf{A}\vec{y} \geq \vec{b}} p(\vec{y})$. Furthermore, we showed how to find $\vec{x} \in \mathbb{R}^d$ such that

$$\sum_{i=1}^{n} w_i \frac{[\mathbf{A}(\vec{x} - \vec{x}_*)]_i^2}{(\mathbf{A}\vec{x}_* - \vec{b})_i^2} \leq \frac{1}{100}\,. \tag{VII.5}$$

As discussed in Theorem 23, this can be done in time $\tilde{O}(\sqrt{d}\left(\mathrm{nnz}(\mathbf{A}) + d^2\right)\log(U))$. Consequently, we can use Lemma 28 with $\gamma = 3$ and $\left\|\vec{w}\right\|_1 = 3d$ and this gives us an ellipsoid $E$ such that $\vec{x}_* + \frac{1}{3}E \subset P \subset \vec{x}_* + 9dE$ where $E = \{\vec{h} \in \mathbb{R}^d \,:\, \left\|\vec{h}\right\|_{\nabla^2 p(\vec{x}_*)} \leq 1\}$.

Now, we show how to approximate $E$ using $\vec{x}$. (VII.5) shows that that $\left\|\vec{x} - \vec{x}_*\right\|_{\nabla^2 p(\vec{x}_*)} \leq \frac{1}{10}$, therefore, we have $\vec{x} - \vec{x}_* \in \frac{1}{10}E$ and hence

$$\vec{x} + \left(\frac{1}{3} - \frac{1}{10}\right)E \subset \vec{x}_* + \frac{1}{3}E \subset P \subset \vec{x}_* + 9dE \subset \vec{x} + \left(9d + \frac{1}{10}\right)E.$$

Now, using (VII.5) and (VII.4), we have $\left|\frac{\mathbf{A}(\vec{x} - \vec{x}_*)}{\mathbf{A}\vec{x}_* - \vec{b}}\right|_i \leq \frac{3}{10}$, we have

$$\frac{1}{(1 + \frac{3}{10})^2}\nabla^2 p(\vec{x}) \preceq \nabla^2 p(\vec{x}_*) = \mathbf{A}^T\mathbf{S}_*^{-1}\mathbf{W}\mathbf{S}_*^{-1}\mathbf{A} \preceq \frac{1}{(1 - \frac{3}{10})^2}\nabla^2 p(\vec{x})\,.$$

Therefore, we have $\vec{x} + \frac{\frac{1}{3} - \frac{1}{10}}{1 + \frac{3}{10}}E' \subset P \subset \vec{x} + \frac{9d + \frac{1}{10}}{1 - \frac{3}{10}}E'$ where $E' = \{\vec{h} \in \mathbb{R}^d \,:\, \left\|\vec{h}\right\|_{\nabla^2 p(\vec{x})} \leq 1\}$. After rescaling the ellipsoid, we have the result. ∎

### D. Multicommodity Flow

Here we show how our algorithm can be used to improve the running time for solving multicommodity flow. Note that the result presented here is meant primarily to illustrate our approach, we believe it can be further improved using the techniques in [10], [18].

For simplicity, we focus on the maximum concurrent flow problem. In this problem, we are given a graph $G = (V, E)$ $k$ source sink pairs $(s_i, t_i) \in \mathbb{R}^{V \times V}$ , and capacities $\vec{c} \in \mathbb{R}^E$ and wish to compute the maximum $\alpha \in \mathbb{R}$ such that we can simultaneously for all $i \in [k]$ route $\alpha$ unit of flow $f_i \in \mathbb{R}^E$ between $s_i$ and $t_i$ while maintaining the capacity constraint $\sum_{i=1}^{k} |f_i(e)| \leq c(e)$ for all $e \in E$. There are no combinatorial algorithms known and there are multiple algorithms to compute a $(1 - \epsilon)$ optimal flow in time polynomial in $|E|$, $|V|$ and $\epsilon^{-1}$ [6], [11], [8], [23]. In this regime, the fastest algorithm for directed graphs takes $O((|E| + k)|V|\epsilon^{-2}\log U)$ [23] and the fastest algorithm for undirected graphs takes $\tilde{O}(|E|^{1+o(1)}k^2\epsilon^{-2}\log^{O(1)} U)$ [13] where $U$ is the maximum capacity. For linear convergence, the previous best algorithm is a specialized interior point method that takes time $O(\sqrt{|E|k}|V|^2k^2\log(\epsilon^{-1}))$ [10]. To solve the problem, we use the following linear program

$$\begin{aligned}
\max \quad & \alpha & & \\
g(e) \quad & = & & \textstyle\sum f_i(e) \text{ for all edges } e, \\
\sum_{v \in V} f_i(u, v) \quad & = & & 0 \text{ for all vertices } u \notin \{s_i, t_i\}, \\
\sum_{v \in V} f_i(s_i, v) \quad & = & & \alpha \text{ for all } i, \\
c(e) \geq \quad & f_i(e), \; g(e) & \geq 0 \quad & \text{for all edges } e.
\end{aligned}$$

Note that there are $O(k|E|)$ variables, $O(k|V| + |E|)$ equality constraints, and $O(k|E|)$ non-zeroes in the constraint matrix. Furthermore, it is easy to find an initial point by computing a shortest path for each $s_i$ and $t_i$ and sending a small amount of flow along that path. Also, given any almost feasible flow, one can make it feasible by scaling and send excess flow at every vertex back to $s_i$ along some spanning tree. Therefore, Theorem 23 gives an algorithm that takes $\tilde{O}(\sqrt{|E| + k|V|}\left(k|E| + (|E| + k|V|)^2\right)\log(U/\epsilon)) = \tilde{O}\left((|E| + k|V|)^{2.5}\log(U/\epsilon)\right)$ time. Note that this is faster than the previous best algorithm when $k \geq (|E|/|V|)^{0.8}$.

### E. Minimum Cost Flow

The minimum cost flow problem and the more specific, maximum flow problem, are two of the most well studied problems in combinatorial optimization [31]. Many techniques have been developed, yet, our result matches the fastest algorithm for solving these problems on dense graphs [18] without using any combinatorial structure of this problem, in particular, Laplacian solvers. We emphasize that this result is not a running time improvement, rather just a demonstration of the power of our result and an interesting statement efficient on maximum flow algorithms.

**Corollary 30.** *There is an $\tilde{O}(|V|^{2.5}\log^{O(1)}(U))$ time algorithm to compute an exact minimum cost maximum flow for weighted directed graphs with $|V|$ vertices, $|E|$ edges and integer capacities and integer cost at most $U$.*

*Proof:* The proof is same as [18, ArXiv v2, Thm 34,35] except that we use Theorem 23 to solve the linear program. The proof essentially writes the minimum cost flow problem into a linear program with an explicit interior point and shows how to round an approximately optimal solution to a vertex of the polytope. To perform this rounding, we need to a fractional solution with error less than $O(\frac{1}{\mathsf{poly}(|V|U)})$ and which yields the $\log(U)$ term in the running time. ■

### F. Convex Problems

Many problems in convex optimization can be efficiently reduced to the problem of finding a point in a convex set $K$ given a separation oracle. Recall that given a point $\vec{x}$, the separation oracle either outputs that $\vec{x}$ is in $K$ or outputs a separating hyperplane that separates the input point $\vec{x}$ and the convex set $K$. In [20], they showed that how to make use of our fast algorithms for inverse maintenance problem under the $K$ stability assumption to obtain the following improved running time for this fundamental problem:

**Theorem 31** ([20])**.** *Given a non-empty convex set $K \subseteq \mathbb{R}^d$ that is contained in a box of radius $R$, i.e. $\max_{x\in K}\|x\|_\infty \leq R$. We are also given a separation oracle for $K$ that takes $O(\mathcal{T})$ time for each call. For any $0 < \epsilon < R$, we can either finds $\vec{x} \in K$ or proves that $K$ does not contains a ball with radius $\epsilon$ in time $O(d\mathcal{T}\log(dR/\epsilon) + d^3\log^{O(1)}(dR/\epsilon))$.*

*Remark* 32. [20] uses Theorem 21 to solve the linear systems involved in their cutting plane method. However, their inverse maintenance problem satisfies the $\ell^2$ stability assumption with 1 rows addition and removal. Furthermore, the $\mathbf{A}$ involved has only $O(d)$ many rows. Therefore, we believe a simple variant of [33] may also suffice for that paper.

## VIII. Open Problem: Sampling from a Polytope

Sampling a random point in convex sets is a fundamental problem in convex geometry with numerous applications in optimization, counting, learning and rounding [34]. Here consider a typical case where the convex set is a polytope that is explicitly given as $\{\vec{x} \in \mathbb{R}^d : \mathbf{A}\vec{x} \geq \vec{b}\}$ for $\mathbf{A} \in \mathbb{R}^{n\times d}$. The current fastest algorithm for this setting is Hit-and-Run [22] and Dikin walk [9].

Given an initial random point in the set, Hit-and-Run takes $O^*(d^3)$ iterations and each iteration takes time $O^*(\mathrm{nnz}(A))$ while Dikin walk takes $O^*(nd)$ iterations and each iteration takes time $O^*(nd^{\omega-1})$ where the $O^*$ notation omits the dependence on error parameters and logarithmic terms. Each iteration of Dikin walk is expensive because it solves a linear system to obtain the next point and computes determinants to implement an importance sampling scheme. The linear systems can be solved in amortized cost $O^*(\mathrm{nnz}(\mathbf{A})+d^2)$ by the inverse maintenance machinery presented in this paper. Unfortunately it is not known how to use this machinery to efficiently compute the determinant to sufficient accuracy to suffice for this method.

We leave it as an open problem whether it is possible to circumvent this issue and improve the running time of a method like the Dikin walk. In an older version of this paper, we mistakenly claimed an improved running time for Dikin walk by noting solely the improved running time for linear system solving and ignoring the determinant computation. We thank Hariharan Narayanan for pointing out this mistake.

## IX. Acknowledgments

We thank Yan Kit Chim, Andreea Gane, and Jonathan A. Kelner for many helpful conversations. This work was partially supported by NSF awards 0843915 and 1111109, NSF Graduate Research Fellowship (grant no. 1122374). Part of this work was done while both authors were visiting the Simons Institute for the Theory of Computing, UC Berkeley.

## REFERENCES


[1] Hui Han Chin, Aleksander Madry, Gary L Miller, and Richard Peng. Runtime guarantees for regression problems. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 269–282. ACM, 2013.

[2] Kenneth L Clarkson and David P Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 81–90. ACM, 2013.

[3] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. *CoRR*, abs/1408.5099, 2014.

[4] Michael B Cohen and Richard Peng. Lp row sampling by lewis weights. *arXiv preprint arXiv:1412.0588*, 2014.

[5] Petros Drineas, Michael W Mahoney, and S Muthukrishnan. Subspace sampling and relative-error matrix approximation: Column-based methods. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 316–326. Springer, 2006.

[6] Lisa K Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[7] François Le Gall. Powers of tensors and fast matrix multiplication. *arXiv preprint arXiv:1401.7714*, 2014.

[8] Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[9] Ravindran Kannan and Hariharan Narayanan. Random walks on polytopes and an affine interior point method for linear programming. *Mathematics of Operations Research*, 37(1):1–20, 2012.

[10] Sanjiv Kapoor and Pravin M Vaidya. Speeding up karmarkar's algorithm for multicommodity flows. *Mathematical programming*, 73(1):111–127, 1996.

[11] George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)*, 4(1):13, 2008.

[12] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[13] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *SODA*, pages 217–226. SIAM, 2014.

[14] Leonid G Khachiyan. Rounding of polytopes in the real number model of computation. *Mathematics of Operations Research*, 21(2):307–320, 1996.

[15] Piyush Kumar and E Alper Yildirim. Minimum-volume enclosing ellipsoids and core sets. *Journal of Optimization Theory and Applications*, 126(1):1–21, 2005.

[16] Béatrice Laurent and Pascal Massart. Adaptive estimation of a quadratic functional by model selection. *Annals of Statistics*, pages 1302–1338, 2000.

[17] Yin Tat Lee and Aaron Sidford. Path finding i: Solving linear programs with \˜ o (sqrt(rank)) linear system solves. *arXiv preprint arXiv:1312.6677*, 2013.

[18] Yin Tat Lee and Aaron Sidford. Path finding ii: An\˜ o (m sqrt (n)) algorithm for the minimum cost flow problem. *arXiv preprint arXiv:1312.6713*, 2013.

[19] Yin Tat Lee and Aaron Sidford. Path-finding methods for linear programming : Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 18-21 October, 2014, Philadelphia, PA, USA*, pages 424–433, 2014.

[20] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*. IEEE, 2015.

[21] Mu Li, Gary L Miller, and Richard Peng. Iterative row sampling. 2012.

[22] László Lovász and Santosh Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 57–68. IEEE, 2006.

[23] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 121–130. ACM, 2010.

[24] Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011.

[25] Michael W Mahoney, Petros Drineas, Malik Magdon-Ismail, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. In *ICML*, 2012.

[26] Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. *arXiv preprint arXiv:1211.1002*, 2012.

[27] Yu Nesterov and Arkadi Nemirovskiy. *Self-concordant functions and polynomial-time methods in convex programming*. USSR Academy of Sciences, Central Economic & Mathematic Institute, 1989.

[28] Yu Nesterov and A Nemirovsky. Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem. *SIAM Journal on Optimization*, 1(4):548–564, 1991.

[29] Yurii Nesterov and Arkadii Semenovich Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Society for Industrial and Applied Mathematics, 1994.

[30] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, 40(1-3):59–93, 1988.

[31] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[32] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[33] Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 332–337. IEEE, 1989.

[34] Santosh S Vempala. Recent progress and open problems in algorithmic convex geometry. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 8. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[35] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898. ACM, 2012.