Strongly Polynomial Frame Scaling to High Precision *

Daniel Dadush† Akshay Ramachandran‡

November 6, 2023

Abstract

The frame scaling problem is: given vectors $U := \{u_1, ..., u_n\} \subseteq \mathbb{R}^d$, marginals $c \in \mathbb{R}^n_{++}$, and precision $\varepsilon > 0$, find left and right scalings $L \in \mathbb{R}^{d\times d}, r \in \mathbb{R}^n_{++}$ such that $(v_1, \dots, v_n) := (Lu_1r_1, ..., Lu_nr_n)$ simultaneously satisfies $\sum_{i=1}^n v_i v_i^\mathsf{T} = I_d$ and $\|v_j\|_2^2 = c_j, \forall j \in [n]$, up to error ε. This problem has appeared in a variety of fields throughout linear algebra and computer science. In this work, we give a strongly polynomial algorithm for frame scaling with $\log(1/\varepsilon)$ convergence. This answers a question of Diakonikolas, Tzamos and Kane (STOC 2023), who gave the first strongly polynomial randomized algorithm with poly$(1/\varepsilon)$ convergence for Forster transformation, the special case $c = \frac{d}{n}1_n$. Our algorithm is deterministic, applies for general marginals $c \in \mathbb{R}^n_{++}$, and requires $O(n^3 \log(n/\varepsilon))$ iterations as compared to the $O(n^5 d^{11}/\varepsilon^5)$ iterations of DTK. By lifting the framework of Linial, Samorodnitsky and Wigderson (Combinatorica 2000) for matrix scaling to the frame setting, we are able to simplify both the algorithm and analysis. Our main technical contribution is to generalize the potential analysis of LSW to the frame setting and compute an update step in strongly polynomial time that achieves geometric progress in each iteration. In fact, we can adapt our results to give an improved analysis of strongly polynomial matrix scaling, reducing the $O(n^5 \log(n/\varepsilon))$ iteration bound of LSW to $O(n^3 \log(n/\varepsilon))$. Additionally, we give a bound on the size of approximate scaling solutions, which involves condition measure $\bar{\chi}$ studied in the linear programming literature, and may be of independent interest.

1 Introduction

In this work, we study the following problem on vectors:

Definition 1.1. (Frame Scaling) *Given input $U := (u_1, ..., u_n) \in \mathbb{R}^{d\times n}$, marginals $c \in \mathbb{R}^n_{++}$, and precision $\varepsilon > 0$, find a* **left scaling** *$L \in \mathbb{R}^{d\times d}$ and a* **right scaling** *$r \in \mathbb{R}^n_{++}$, such that the vectors $v_j := Lu_jr_j, j \in [n]$, are in ε-approximate (I_d, c)-position:*

$$\Big\| \sum_{j=1}^n v_j v_j^\mathsf{T} - I_d \Big\|_F^2 + \sum_{j=1}^n \Big(\|v_j\|_2^2 - c_j \Big)^2 \le \varepsilon^2.$$

We say $V := (v_1, \dots, v_n) \in \mathbb{R}^{d\times n}$ is (I_d, c)-scaled if $\varepsilon = 0$ in the above.

Scaling to approximate (I_d, c)-position can be thought of as a way to regularize a set of vectors. The first error term is small when the vectors are globally balanced in the sense that $\|V^\mathsf{T} x\|_2 \approx \|x\|_2$ for all directions $x \in \mathbb{R}^d$; and, for example for the special case $c = \frac{d}{n}1_n$, the second error term is small when all the vectors have nearly the same norm. This problem has appeared in a variety of areas including algebraic geometry [42], functional analysis [7], coding theory and signal processing [29], [41], and communication complexity [5]. The question of existence of frame scalings has been asked and answered many times in the literature:

Theorem 1.1. ([7, 42]) *Input $U \in \mathbb{R}^{d\times n}$ can be scaled to ε-approximate (I_d, c)-position for any $\varepsilon > 0$ iff*

$$\forall T \subseteq [n] : \quad \langle c, 1_T \rangle \le \mathrm{rk}(U_T) \qquad \text{and} \qquad \langle c, 1_n \rangle = d.$$

We say (U, c) is feasible if this is the case, and otherwise we say it is infeasible.

*The full version of the paper can be accessed at `https://arxiv.org/abs/XXX.XXXX` (will have link in a few days!)

†Centrum Wiskunde and Informatica.

‡Centrum Wiskunde and Informatica.

Some previous proofs of this existence statement use algebraic or compactness arguments, and therefore are non-constructive. More recently, there has been significant focus on algorithms for computing frame scalings. These are especially useful for data analysis or signal processing.

Our work is greatly influenced by two recent directions (scaling and learning theory), and we discuss these connections in more detail below.

The first is a recent line of work on a class of related problems such as operator scaling [20], [26] and Brascamp-Lieb constants [25]. These problems fall into the so-called scaling framework, with connections to algebraic geometry and invariant theory (see [31] for a detailed exposition). In particular, there has been a great deal of progress in new tractable algorithms for many problems in this framework (including frame scaling), which exploit the underlying algebraic and geometric structure of scalings.

The second is the recent application of frame scaling to problems in learning theory. In this context, the special case of $c := \frac{d}{n}1_n$ is known as the Forster transform, as it was originally studied by Forster [5] in the context of communication complexity lower bounds. This has been used as a robust method to regularize a dataset in order to speed up various downstream learning tasks [6], [8], [14], [9].

Many of the known algorithms for frame scaling exploit a convex formulation of this problem and then apply standard optimization techniques, such as Ellipsoid, gradient descent, or interior point methods [6], [32], [13], [8]. The standard analyses of these off-the-shelf methods as applied in these algorithms have weakly polynomial guarantees, in that the number of iterations depends on the bit complexity of the input.

A related problem is *matrix scaling problem*: given a non-negative matrix $A \in \mathbb{R}_+^{m\times n}$, desired row and column sums $(r, c) \in \mathbb{R}_{++}^{m+n}$ and a tolerance $\varepsilon > 0$, the goal is to compute positive diagonal matrices $L \in \mathrm{diag}_{++}(m), R \in \mathrm{diag}_{++}(n)$ such that the row and column sums $(RA^\mathsf{T}L1_m, LAR1_n)$ of the rescaled matrix LAR are within distance ε of (r, c). To make the analogy to frame scaling, for input $U \in \mathbb{R}^{d\times n}$, the column sum condition corresponds to the column norm condition on U, and the row sum condition corresponds to the isotropy condition. In classical work, Linial, Samrodnitsky, and Wigderson [1] gave a strongly polynomial $\mathrm{poly}(n, m)\log(n/\varepsilon)$-time algorithm for matrix scaling. Given the similarity with frame scaling, a natural question is whether a strongly polynomial algorithm also exists for frame scaling. In very recent work, Diakonikolas, Tzamos and Kane [9] gave a strongly polynomial $\mathrm{poly}(n, d, 1/\varepsilon)$-time algorithm for computing Forster transformations, which is a special case of frame scaling with $c = \frac{d}{n}1_n$. They applied this result to give the first strongly polynomial halfspace learner for certain noise models. The running time of their algorithm has inverse polynomial dependence on the error parameter ε, so they asked whether it is possible to extend this strongly polynomial result to the setting of exponentially small ε. In this work, we answer this question in the affirmative by giving a strongly polynomial algorithm for the frame scaling problem with arbitrary $c \in \mathbb{R}_+^n$ that has $\log(1/\varepsilon)$ runtime dependence.

Before presenting our main result, we discuss how our scalings are represented in strongly polynomial time. First note that both the isotropy and the norm condition are easy to satisfy individually simply by normalizing. This means that for any left scaling L, there is a simple way to compute a right scaling satisfying the norm condition, namely $r_j^2 := c_j/\|Lu_j\|_2^2$. Similarly, for any right scaling matrix $R = \mathrm{diag}(r_1, \dots, r_n)$ (the diagonal matrix with diagonal r), there is a natural corresponding left scaling $L := (UR^2U^\mathsf{T})^{-1/2}$ satisfying the isotropy condition. Unfortunately, it is not possible to take square roots in the strongly polynomial model. Therefore, in our algorithm, we will maintain the square of the right scaling $z := (r_1^2, \dots, r_n^2) \in \mathbb{R}_{++}^n$ while leaving the left scaling $L := (UZU^\mathsf{T})^{-1/2}$ implicit. We now present our main result.

Theorem 1.2. (Main Theorem (Informal)) *Given input $U \in \mathbb{R}^{d\times n}$, marginals $c \in \mathbb{R}_{++}^n$, and precision $\varepsilon > 0$, there is a deterministic strongly polynomial time algorithm, which in $O(n^3\log(n/\varepsilon)$ iterations, each requiring $\mathrm{poly}(n, d)$ arithmetic operations, either outputs the squared right scaling $z \in \mathbb{R}_{++}^n$ of an ε-approximate frame scaling solution, or outputs a certificate of infeasibility $T \subseteq [n]$ satisfying $\langle c, 1_T\rangle > \mathrm{rk}(U_T)$.*

As mentioned above, this answers the open question of [9] regarding the dependence on ε and also resolves the general marginal case. As added benefits, our algorithm is both deterministic and substantially simpler than that of [9]. The main reason for the latter is that [9] works with left scalings directly, which are matrices, whereas we work directly with (squared) right scalings, which are just positive vectors. Using right scalings, we are able to lift a potential function used in [1] from the matrix to the frame scaling setting, for which it is relatively simple to argue geometric potential function decrease. Even for large $\varepsilon > 0$, our algorithm is somewhat more efficient than that of [9]. In terms of iterations, after each of which the potential drops, we require $O(n^3\log(n/\varepsilon))$ instead of $O(n^5d^{11}/\varepsilon^5)$. For both algorithms, each iteration requires $\mathrm{poly}(n, d)$ arithmetic operations, which we have not attempted to optimize. The formal complexity of each iteration is left to the main body, along with a detailed comparison with prior work (see Theorem 2.1).

Interestingly our potential function analysis is tighter than that of [1] for matrix scaling. As a consequence, we show that using a slightly optimized update step, one can decrease the number of iterations needed by the [1] matrix scaling algorithm from $O(n^5 \log(n/\varepsilon))$ to $O(n^3 \log(n/\varepsilon))$ (see Section 5).

In the rest of the introduction, we give a high-level overview of the techniques (Section 1.1), a detailed comparison between the different algorithms (Section 1.2), and a discussion of (often overlooked) subtleties in the definition of strongly polynomial algorithms versus weakly polynomial algorithms.

1.1 Techniques In this subsection, we give technical overview of our improved algorithm for frame scaling. Our high-level strategy is essentially the same as that of [1] for matrix scaling. We iteratively update the scaling so that the norm square of the error decreases sufficiently. Each update step is a uniform scaling of a subset of columns, and the goal is to find a step-size making sufficient progress. How precisely to choose the update direction and step-size is the main technical work. The algorithm of [9] for Forster transform is similar in spirit, but the update step and analysis both involve eigenspaces and so are more complicated; we defer more detailed comparison to Section 1.2.

We begin by helpful simplification: to find an ε-approximate scaling according to Definition 1.1, it is enough to compute $z \in \mathbb{R}^n_{++}$ such that $\|\tau^U(z) - c\|_2^2 \le \varepsilon^2$ for

$$\tau_j^U(z) := z_j u_j^\mathsf{T} (UZU^\mathsf{T})^{-1} u_j.$$

This is equivalent to original frame scaling problem: applying the left and right scalings $L := (UZU^\mathsf{T})^{-1/2}, R := \sqrt{Z}$, we get that $V := LUR$ automatically satisfies the isotropy condition $VV^\mathsf{T} = I_d$ and has squared column norms $\{\|v_j\|_2^2 = \tau_j^U(z)\}_{j\in[n]}$. The quantity $\tau_j^U(z)$, $j \in [n]$, is the *leverage score* of the jth row of $\sqrt{Z}U^\mathsf{T}$, where $Z := \operatorname{diag}(z_1, \ldots, z_n)$, an extremely well studied quantity in numerical linear algebra which indicates the "importance" of a row. Note that $\tau^U(z)$ can be computed in strongly polynomial time as it involves just matrix inversions and multiplications. This is a key simplification as we only have to maintain a vector $z \in \mathbb{R}^n_{++}$ instead of a matrix. This is the first reason that we choose to work with the squares of right scalings and leave the left scaling implicit.

Now we can focus on the error $\tau^U(z) - c$, which is just a vector. In order to decrease $\|\tau^U(z) - c\|_2^2$ in each iteration, we follow the combinatorial approach of [1] and uniformly scale up a subset of columns with large "margin". Explicitly, we sort the error vector $\tau_1^U(z) - c_1 \le ... \le \tau_n^U(z) - c_n$, and choose $T = [k]$ to be the prefix set maximizing margin $2\gamma := (\tau_{k+1}^U(z) - c_{k+1}) - (\tau_k^U(z) - c_k)$. In Proposition 2.1, we show this maximum margin satisfies $\gamma^2 \ge \|\tau^U(z) - c\|_2^2/(2n^3)$. From here, our main goal will be to compute a new iterate $z' \in \mathbb{R}^n_{++}$ satisfying $\|\tau^U(z) - c\|_2^2 - \|\tau^U(z') - c\|_2^2 \gtrsim \gamma^2$, which decreases the squared error by a $(1 - \Omega(1/n^3))$ multiplicative factor, and in turn gives our $O(n^3 \log(n/\epsilon))$ iteration bound.

The next iterate is of the form $z' \leftarrow z \circ (1_{\overline{T}} + \alpha 1_T)$ for some $\alpha \ge 1$, where $\circ$ denotes the entry-wise product of vectors and $1_T \in \{0,1\}^n$ is the indicator of T. It is straightforward to show that this scaling increases all $\tau|_T$ and decreases all $\tau|_{\overline{T}}$, the leverages scores indexed by T and $\bar{T} := [n] \setminus T$ respectively. The key of our potential analysis is Lemma 2.1, where we show that the decrease in error $\|\tau^U(z) - c\|_2^2$ can be lower bounded in terms of the total amount that the leverage scores increase and the initial margin γ. Formally, we define the proxy function $h(\alpha)$ to be the sum of leverage scores $\tau|_T$ when scaling up $z \to z \circ (1_{\overline{T}} + \alpha 1_T)$. And in Lemma 2.1, we show that the error $\|\tau^U(z) - c\|_2^2$ decreases additively by $2\gamma(h(\alpha) - h(1))$ as long as $h(\alpha) - h(1) \le \gamma$.

In words, we show that as long as there is still a gap between $\tau|_T$ and $\tau|_{\overline{T}}$, then the error $\|\tau^U(z) - c\|_2^2$ is decreasing at a rate proportional to margin γ and the difference in leverage scores $\tau|_T$.

If we could find an update α such that $\gamma/5 \le h(\alpha) - h(1) \le \gamma$, then this analysis would show geometric progress in each iteration, since we have already argued γ^2 is a polynomial fraction of the potential $\|\tau^U(z) - c\|_2^2$. Following a similar argument as [1], we can show that such an update always exists in the feasible case. Specifically, in Proposition 2.2 we show $\lim_{\alpha\to\infty} h(\alpha) = \operatorname{rk}(U_T)$, whereas the margin γ of T effectively shows $h(1) \le \langle c, 1_T \rangle - \gamma$. Using feasibility $\langle c, 1_T \rangle \le \operatorname{rk}(U_T)$ and the fact that h is increasing, we must eventually have an increase $h(\alpha) - h(1) \ge \gamma/2$. In the contrapositive, this shows that a good update does not exist only when the subset T gives a certificate of infeasibility, $\langle c, 1_T \rangle > \operatorname{rk}(U_T)$ according to Theorem 1.1.

The rest of the algorithm, actually computing an update in the feasible case, is the main technical contribution of this work. Formally, we want an update satisfying $\gamma/5 \le h(\alpha) - h(1) \le \gamma$, which is essentially an approximate root-finding problem. In our setting, the function h is non-increasing and concave, as we show in Corollary 2.1, so we are able to use the classical Newton-Dinkelbach method, along with a refined Bregman divergence-based analysis similar to [38], in order to efficiently compute the update.

We note that the concavity of $h(\alpha)$ is another helpful property that is a consequence of our choice of representation as squares of right scalings $z \in \mathbb{R}^n_{++}$.

The running time of the Newton-Dinkelbach method applied to our function h is proportional to $O(\log(1 + \alpha_*/\alpha_0))$, where $h(\alpha_*) = h(1) + \gamma/5$ and α_0 is our initial guess. We emphasize that this is a special property of our function h and our desire to just compute an update making sufficient progress; in general, the analysis is more complicated (see [38]).

We rewrite the proxy function as

$$h(\alpha) = \mathrm{Tr}[\alpha U_T Z_T U_T^\mathsf{T}(UZU^\mathsf{T} + (\alpha-1)U_T Z_T U_T^\mathsf{T})^{-1}] = \sum_{i=1}^{d} \frac{\alpha\mu_i}{1+(\alpha-1)\mu_i},$$

where $1 \geq \mu_1 \geq \cdots \geq \mu_d \geq 0$ are the eigenvalues of $U_T Z_T U_T^\mathsf{T}(UZU^\mathsf{T})^{-1}$. The eigenvalues are in the range $[0,1]$ as the matrix is similar to a principal submatrix of a projection matrix, and the number of non-zero eigenvalues is precisely $\mathrm{rk}(U_T)$. Note that we do not have access to μ, but only to evaluations of $h(\alpha)$ and $h'(\alpha)$. In Lemma 2.3, we are able to use this implicit information to show that the solution lies in a small multiplicative range, either close to 1 or close to γ/μ_s where $\mu_s := \sum_{i:\mu_i<1/2} \mu_i$ is the sum of small eigenvalues.

Unfortunately, we cannot compute μ_s in strongly polynomial time, but we only need a polynomial approximation of this quantity in order to run Newton-Dinkelbach efficiently. By some preprocessing, we can in fact reduce to the setting where there is a large gap between the small and large eigenvalues. In order to compute the spectral sum μ_s, our goal is to approximate the subspace of large eigenvectors. Under the gapped assumption, all the vectors are close to the large eigenspace, so we can approximate this subspace by the span of a column subset. To find a good column subset, we apply a classical combinatorial algorithm of Knuth [2] which finds a subset with large determinant, and adapt the analysis to show that this gives a sufficiently accurate (but very weak) eigendecomposition in our setting. Using the resulting initial guess with the Newton-Dinkelbach method allows us to efficiently compute the desired update.

The final piece of the algorithm is a regularization step that we use to maintain bounded bit complexity. A similar idea was also used in [9] for the strongly polynomial Forster transform. But once again, the fact that we only maintain the square of the right scaling $z \in \mathbb{R}^n_{++}$ instead of a matrix greatly simplifies our regularization procedure while allowing us to give a more refined bound. Formally, in Theorem 4.1, we show that we can always reduce the condition number $\log \frac{z_{\max}}{z_{\min}}$ by shrinking the (multiplicative) gaps between the entries of z while maintaining small error. In fact, we are able to relate the required magnitude and bit complexity of the scaling to the $\bar{\chi}$ condition measure of the frame U. This is an extremely well-studied concept in the linear programming literature (see e.g. [37]), and importantly is a much more refined condition measure than just the bit complexity of the frame. We believe this relation and our regularization bounds are of independent interest.

1.2 Relation to Previous Work

In this subsection, we compare our algorithm and analysis to the previous strongly polynomial scaling algorithms of [1] and [9]. As mentioned in Section 1.1, the high-level iterative approach is similar, so we focus on the differences in implementation and analysis that lead to our improvements.

We first recall the main components of the algorithm:

1. We first reduce from finding a left and right scaling $L \in \mathbb{R}^{d\times d}$ and $r \in \mathbb{R}^n_{++}$, to just the square of a right scaling $z = (r_1^2, ..., r_n^2) \in \mathbb{R}^n_{++}$.

2. We use the norm squared of the error $\|c - \tau^U(z)\|_2^2$ as our potential function, and attempt to decrease it by a $(1 - O(1/n^3))$ multiplicative factor.

3. In each iteration, we choose a "subset" T with large "margin" $\gamma > 0$ to scale up uniformly by a factor $\alpha \geq 1$. That is, $z \to z \circ (1_{\overline{T}} + \alpha 1_T)$.

4. We relate the improvement to the proxy function $h_T^U(\alpha)$, the sum of the leverage scores inside T, and find a step-size $\alpha \geq 1$ satisfying $h_T(\alpha) \in [\gamma/5, \gamma]$ to derive the desired geometric decrease in potential.

5. We perform a regularization step to keep multiplicative range $\max_i z_i / \min_i z_i$ of the scaling uniformly bounded. This enables us to maintain polynomial bit complexity.

In the remainder, we will discuss each of the above steps in more detail, specifically how they compare across the different strongly polynomial algorithms, and how our new techniques give improved convergence.

Representation of Scalings: All three algorithms reduce to just one-sided scaling. In the matrix setting, for each right scaling $R \in \text{diag}_{++}(n)$, there is a unique left scaling normalizing the rows, namely $L_{ii} := r_i/(AR1_n)_i$. Therefore, the algorithm of [1] maintains a right scaling while implicitly choosing the left scaling that matches the row condition.

In the frame setting, there are two inequivalent ways to perform this reduction: any right scaling $R \in \text{diag}_{++}(n)$ induces a unique isotropic left scaling $L := (UR^2U^\mathsf{T})^{-1/2}$; and similarly, any left scaling $L \in \mathbb{R}^{d\times d}$ induces a unique right scaling satisfying the norm conditions, $r_j^2 := c_j/\|Lu_j\|_2^2$. Our algorithm uses the first approach, whereas the algorithm of [9] applies the second.

This ends up being a key reason that our algorithm and analysis is quite a bit simpler. As an example, our representation is a positive vector, whereas the algorithm of [9] requires keeping track of the left scaling matrix. And the spectral properties of the left scaling become crucial in their algorithm (e.g. condition number and singular vectors), for which they require a much more complicated set of procedures, such as a strongly polynomial approximate eigendecomposition.

Potential Function: For the potential function, both our work and [1] measure the norm square of the error of the column marginals, whereas the potential function in [9] measures the error of the isotropy condition $VV^\mathsf{T} - I_d$, which is a matrix quantity. This difference is another source of difficulty for the Forster transform algorithm of [9], as updates no longer just increase or decrease norms, but affect the whole error matrix, and in particular its eigenvalues, in more complicated ways.

Update Direction: Both our work and [1] choose to uniformly scale up a set of columns with large margin as defined in Section 1.1. On the other hand, the potential function of [9] is a matrix quantity, and therefore the relevant notion of "gap" is with respect to its eigenvalues. Assuming an exact eigendecomposition method, the analogous update step would be to scale up the eigenspace corresponding to the small errors. While an exact eigendecomposition cannot be performed in strongly polynomial time, they developed a randomized power method based algorithm to compute a very accurate approximation of the small eigenspace. In our algorithm, we only require a $\text{poly}(n,d)$ factor approximation of the sum of the small eigen values (i.e., of size at most $1/2$), a significantly simpler spectral quantity. The fact that we only require a very coarse approximation allows us to a use a simple deterministic method for this purpose.

Update Step-Size: This is the simplest in the matrix scaling setting of [1], as the step-size can be computed explicitly in terms of the input matrix and current scalings. In Section 5, we show how to adapt our proxy function analysis to give an improved update step that makes quadratically more progress in each step and reduces the number of iterations by the same factor.

In [9], the update is quite complicated for a few reasons: because the error is a matrix quantity, it is harder to control how scaling up an eigenspace affects the error $\|VV^\mathsf{T} - I_d\|_F^2$. Specifically, there is an off-diagonal term in Lemma 3.3 of [9] which detracts from the progress for large step sizes. For this reason, the algorithm in [9] breaks into two cases depending on how the off-diagonal term grows in terms of the scaling. In both cases, the algorithm chooses a fixed step-size that is guaranteed to make $\text{poly}(\varepsilon/nd)$ progress. This is one of the two main bottlenecks for the $\text{poly}(1/\varepsilon)$ runtime, as it is difficult to make geometric progress in the presence of the off-diagonal term. The other main bottleneck is the strongly polynomial approximate eigendecomposition procedure, which is an important and novel technical contribution of [9]. We are able to use our proxy function analysis and our weak eigendecomposition procedure to overcome both of these hurdles, which allows us to have $\log(1/\varepsilon)$ convergence.

Controlling Bit Size: The algorithm in [1] does not use a regularization step, and instead relies on the fact that the magnitude of the scalings at most squares in each iteration. While squaring can double the bit length, the algorithm only requires the top $O(\log(n/\varepsilon))$ most significant bits of each number (scaling or matrix entry) when computing the updates. This is because the algorithm only requires $(1 + \text{poly}(\varepsilon/nd))$ multiplicative approximations of partial row and column sums, and all of these quantities consist only of non-negative numbers. This allows them to use a floating point representation of the entries and scalings, keeping track of only the high order bits, and keeping the size of the bit-representations under control.

On the other hand, the frame scaling setting requires performing matrix operations (pseudo-inverses, determinants, projections, rank computations) up to varying levels of accuracy, which are much more sensitive to small errors than approximating sums of non-negative numbers as in matrix scaling. Therefore, both our algorithm and that of [9] involve an additional regularization step at the end to control the magnitude and bit complexity of scalings. And here again we benefit from the fact that our scaling is just a vector of numbers, whereas the left scaling maintained in [9] is a matrix. In particular, the regularization procedure in [9] is an iterative method involving approximate eigendecompositions and projections of point sets to subspaces, whereas ours can be performed using just arithmetic operations. Furthermore, we are able to relate the required bit complexity to the $\bar{\chi}$ condition measure of the input, which gives a much refined bound than just bit size.

1.3 Strong and Weak Polynomial Time The main result in this work is a faster strongly polynomial algorithm for frame scaling. In this subsection, we will more precisely define the computational model used in this work, as well as some subtleties that require clarification.

The "genuinely polynomial" model was first described by Meggido [35], and can be seen as a blend of the real and bit model: for input given as n real numbers, a strongly polynomial algorithm can perform $\text{poly}(n)$ exact arithmetic and comparison operations; further, if the input has bit complexity b, then the algorithm must have polynomial space complexity, in the sense that intermediate quantities must remain $\text{poly}(n, b)$ bit complexity. Note that the algorithm does not have knowledge of the bit complexity.

This work studies strongly polynomial algorithms for ε-approximate frame scaling with $\log(1/\varepsilon)$ convergence. Formally, our algorithm performs $\text{poly}(n, \log(1/\varepsilon))$ exact arithmetic and comparison operations in the real model. Unfortunately, in order to maintain polynomial space complexity, we require a regularization procedure that rounds intermediate quantities to some precision depending on the bit complexity of the input frame. This is not strongly polynomial in the strictest sense of the definition above, as it is not oblivious of the bit representation.

But this is a subtle obstacle that occurs in many previous algorithms that claim to be strongly polynomial. We discuss these examples in more detail in Section 6. To deal with this issue, we propose a slightly modified computational model that seems to capture the essential components of the strong polynomial requirement. Essentially, the algorithm may request that arithmetic operations be performed either exactly or approximately. In the approximate case, the operations are performed up to error ε, where $\varepsilon = 1/2^{\text{poly}(n,b)}$ when the input has bit complexity b. Importantly, the algorithm does not know the bit complexity of the input, nor of the exact precision of the approximate computations, and so still remains oblivious of the bit representation in a meaningful sense. At a high level, the algorithm requests approximate arithmetic operations for computations that are "numerically stable" (i.e., that change continuously with small perturbations of the input), and exact operations for the remaining computations (e.g., computing the rank of matrix). Note that for standard arithmetic operations such as addition, division and multiplication, one can instantiate the approximate computations by performing the computations in exact arithmetic and then truncating their bit description to the desired precision. We give a more formal definition of this model in Section 6, along with a comparison with other natural computational models.

1.4 Notation We use $\mathbb{R}$, $\mathbb{R}_+$ and $\mathbb{R}_{++}$ to denote the reals, non-negative reals and positive reals respectively. For $z \in \mathbb{R}^n$ we use $Z := \text{diag}(z) \in \mathbb{R}^{n\times n}$. We use $\text{diag}(n)$ and $\text{diag}_{++}(n)$ to denote $n \times n$ diagonal matrices with real and positive real diagonal entries respectively. For vectors $\{u_1, ..., u_n\}$, we will sometimes abuse notation and let U be the matrix with columns $(u_1, ..., u_n)$. For univariate function $f(x)$ we use $\frac{d}{dx}$ to be the derivative with respect to x. We use $\circ$ for the entry-wise (Hadamard) product of vectors. That is, $(x \circ y)_i = x_i y_i$ for $x, y \in \mathbb{R}^n$. For set $T \subseteq [n]$ we use 1_T to be the indicator vector of a set, and 1_n to be the all-ones vector.

1.5 Organization The remainder of this work is structured as follows: in Section 2, we present our main iterative algorithm and high level convergence analysis for the frame scaling problem (Definition 1.1). In Section 3, we present and analyze a coarse eigendecomposition procedure that is used as a subroutine to properly initialize the Newton-Dinkelbach method used to compute the update in each iteration. In Section 4 we show a simple procedure to round scalings in each iteration in order to maintain bounded bit complexity. In Section 5 we sketch how our new techniques give an improved procedure for strongly polynomial matrix scaling. In Section 6 we give a more detailed discussion of the strongly polynomial model and how it relates to the algorithm presented here. In Section 7, we discuss an interpretation of our work in the context of interior point methods. In Section 8, we discuss how our frame scaling algorithm fits into the known applications for halfspace learning.

2 Iterative Algorithm and Analysis

In this section, we describe the main iterative algorithm used to solve the frame scaling problem with $\log(1/\varepsilon)$ convergence. The update in each iteration scales up a particular subset uniformly in order to make progress. In the rest of this section, we describe how to choose the subset and step-size.

We first observe that we can just keep track of the right scaling, as any fixed $r \in \mathbb{R}^n_{++}$ induces a unique $L := (UR^2U^\top)^{-1/2}$ such that $V := LUR$ is isotropic $VV^\top = I_d$. Therefore, in the rest of the work, we will leave the left scaling implicit. For technical reasons, we will keep track of the square of the right scaling $z \in \mathbb{R}^n_{++}$, so the implicit left scaling is $L := (UZU^\top)^{-1/2}$. With this reduction, we can focus on matching the norms of the columns.

Definition 2.1. *For frame $U \in \mathbb{R}^{d\times n}$, applying right scaling $z \in \mathbb{R}^n_{++}$ and transforming to isotropic position gives norms*

$$\tau^U_j(z) := z_j u_j^\top (UZU^\top)^{-1} u_j = \|(UZU^\top)^{-1/2} u_j \sqrt{z_j}\|_2^2.$$

We let $\tau^U(z) \in \mathbb{R}^n_+$ denote the vector of norms.

These induced norms are known as leverage scores corresponding to the rows of the matrix $\sqrt{Z}U^\top$ and are a well-studied linear algebraic parameter with many applications. Note that even though our left/right scalings involve square roots, the norms $\tau^U(z)$ can be computed exactly by applying a single matrix inversion, which is known to have a strongly polynomial time algorithm [23]. As we show in Section 8, this implicit representation of scalings is sufficient for the known learning theory applications.

The following simple property of leverage scores will be useful throughout our analysis.

Fact 2.1. *For any frame $U \in \mathbb{R}^{d\times n}$ and scaling $z \in \mathbb{R}^n_{++}$,*

$$\langle \tau^U(z), 1_n\rangle = \sum_{j\in[n]} z_j u_j^\top (UZU^\top)^{-1} u_j = \mathrm{Tr}[UZU^\top(UZU^\top)^{-1}] = \mathrm{Tr}[I_d] = d.$$

This implies any feasible input $c \in \mathbb{R}^n_{++}$ must satisfy $\langle c, 1_n\rangle = d$, and we will assume this in the remainder. Therefore, by restricting to the isotropic setting, the frame scaling problem simplifies to:

Definition 2.2. (Isotropic Frame Scaling Problem) *Given frame $U \in \mathbb{R}^{d\times n}$, marginals $c \in \mathbb{R}^n_{++}$ with $\langle c, 1_n\rangle = d$, and $\varepsilon > 0$,*

- *output scaling $z \in \mathbb{R}^n_{++}$ such that $\|\tau^U(z) - c\|_2^2 \le \varepsilon^2$*
- *or output a certificate of infeasibility: $T \subseteq [n]$ such that $\langle c, 1_T\rangle > \mathrm{rk}(U_T)$.*

The following is our main algorithm and convergence result.

```
Main(U, c, ε):
Input: Full row rank frame $U \in \mathbb{R}^{d\times n}$, marginals $c \in \mathbb{R}^n_{++}, \langle c, 1_n\rangle = d$, precision $\varepsilon > 0$;
Output: Scaling $z \in \mathbb{R}^n_{++}$ s.t. $\|\tau^U(z) - c\|_2^2 \le \varepsilon^2$, or certificate of infeasibility: $\langle c, 1_T\rangle > \mathrm{rk}(U_T)$;
while $\|\tau^U(z^{(t)}) - c\|_2^2 > \varepsilon^2$ do
  Sort $\tau^U_1(z^{(t)}) - c_1 \le ... \le \tau^U_n(z^{(t)}) - c_n$;
  Let $k := \arg\max_{j\in[n]} \tau^U_{j+1}(z^{(t)}) - \tau^U_j(z^{(t)})$ be the index with largest gap;
  Choose $T = [k]$ with margin $\gamma := (\tau^U_{k+1}(z^{(t)}) - \tau^U_k(z^{(t)}))/2$;
  if $\mathrm{rk}(U_T) < \langle c, 1_T\rangle$ then
    Output Infeasible;
  end if
  $\hat\alpha \leftarrow$ SubsetScaleUp$(U, z^{(t)}, T, \gamma)$ (Algorithm 2.2);
  $z^{(t+1)} \leftarrow z^{(t)} \circ (1_{\overline{T}} + \hat\alpha 1_T)$;
  $z^{(t+1)} \leftarrow$ Regularize$(z^{(t+1)}, \gamma/\mathrm{poly}(n,d))$ (Theorem 4.1);
  $t \leftarrow t + 1$;
end while
Output $z^{(t)}$;
```

Theorem 2.1. *Algorithm 2 either finds a certificate of infeasibility $\langle c, 1_T\rangle > \mathrm{rk}(U_T)$ or produces a scaling $z \in \mathbb{R}^n_{++}$ satisfying $\|\tau^U(z) - c\|_2^2 \le \varepsilon^2$ in at most $O(n^3 \log(n/\varepsilon))$ iterations. Each iteration can be implemented using $O(n \log n)$ comparisons, and $O(nd^2 \log n)$ matrix operations involving $d \times d$ matrices (multiplication, inverse, determinant), and $O(n)$ Gram-Schmidt operations on $d \times n$ matrices. Further, if the entries of U have bit complexity b, then all intermediate scalings have bit complexity $O(n(db + \log(nd/\varepsilon)))$.*

This can be compared to the previous algorithm given in [9] for the special case $c = \frac{d}{n}1_n$ (known as Forster transformation): this algorithm requires $O(n^5 d^{11}/\varepsilon^5)$ iterations, each of which are implemented using at least $O(nd + d^6 \log(d)/\varepsilon^2)$ matrix operations on $d \times d$ matrices.

Here $z \circ x$ denotes the Hadamard (entry-wise) product of vectors z and x, so in the update step we scale up the subset T by value $\hat{\alpha}$. We defer the proof to the full version. In the remainder of this section, we motivate each of the steps in Algorithm 2.

Since we are trying to decrease the error $\|\tau^U(z) - c\|_2^2$, a natural first attempt would be to try some kind of descent method. To satisfy the strongly polynomial requirement, we instead choose a more combinatorial update step, uniformly scaling up the coordinates in a set. The appropriate choice of set is the one maximizing "margin" defined below. A similar notion appears in both the original matrix scaling algorithm of [1] and the frame scaling algorithm of [9].

Definition 2.3. *For input ($U \in \mathbb{R}^{d\times n}, z \in \mathbb{R}^n_{++}, c \in \mathbb{R}^n_+$) the margin of $T \subseteq [n]$ is defined as the largest $\gamma \ge 0$ such that there exists a threshold $\nu \in \mathbb{R}$ satisfying*

$$\max_{j\in T}(\tau_j^U(z) - c_j) \le \nu - \gamma \le \nu + \gamma \le \min_{j\notin T}(\tau_j^U(z) - c_j).$$

We note that $\tau, c \in [0,1]^n$, which implies that the margin is at most $\|\tau - c\|_\infty/2 = 1$. The margin of a subset T controls how much progress we can make by scaling up T.

We will relate the decrease in error to the margin of T, and the next result shows that we can simply compute a set with large margin:

Proposition 2.1. *For input ($U \in \mathbb{R}^{d\times n}, z \in \mathbb{R}^n_{++}, c \in \mathbb{R}^n_+$), sorting $\tau^U(z) - c$ in non-decreasing order and taking $T \subseteq [n]$ to be the prefix set with largest margin gives*

$$\gamma^2 \ge \frac{1}{2n^3}\|\tau^U(z) - c\|_2^2.$$

Proof. We will use the property $\langle \tau^U(z), 1_n\rangle = d$ by Fact 2.1 and $\langle c, 1_n\rangle = d$ by assumption. Letting $x := \tau^U(z) - c$ for shorthand, we have

$$2n\|x\|_2^2 = \sum_{i,j\in[n]}(x_i - x_j)^2 \le n^2 \max_{i,j\in[n]}(x_i - x_j)^2 \le n^2(2n\gamma)^2 = 4n^4\gamma^2,$$

where the first step uses $\langle x, 1_n\rangle = 0$, and the fourth step uses that the max margin is γ so the maximum distance between two consecutive values is $\le 2\gamma$. Rearranging gives the result. ◻

In the following Section 2.1 we discuss how we use the margin to analyze the improvement of $\|\tau - c\|_2$ due to uniformly scaling up subset T. Then in Section 2.2 we present Algorithm 2.2 which allows us to compute the update making sufficient progress.

2.1 Update Requirement In this section we give explicit conditions for our chosen update step to make sufficient progress in decreasing the error $\|\tau - c\|_2^2$. We also show that these conditions are not satisfied only when the input is infeasible, in which case the set T provides a certificate of infeasibility.

Our plan is to show that our uniform scaling of the coordinates z_T decreases the margin γ, which leads to a decrease in the error. We first observe that the uniform scaling has a simple effect on the leverage scores.

Fact 2.2. *$\tau_j^U(z \circ (1_{\overline{T}} + \alpha 1_T))$ is increasing in α for $j \in T$, and decreasing in α for $j \notin T$.*

This clearly implies that the margin is decreasing. We want to use this to show the error is also decreasing, so we define the following natural proxy function as a way to measure progress:

Definition 2.4. *For frame $U \in \mathbb{R}^{d\times n}$ and scaling $z \in \mathbb{R}^n_{++}$, the following function is used to measure the effect of uniformly scaling up $T \subseteq [n]$:*

$$h_T^{U,z}(\alpha) := \sum_{j\in T} \tau_j^U(z \circ (1_{\overline{T}} + \alpha 1_T)) = \operatorname{Tr}[\alpha U_T Z_T U_T^{\mathsf{T}} (U_{\overline{T}} Z_{\overline{T}} U_{\overline{T}}^{\mathsf{T}} + \alpha U_T Z_T U_T^{\mathsf{T}})^{-1}]$$

With these definitions in hand, we give a structural lemma showing that the update step decreases the error proportional to the margin of the set and the proxy function h. For the proof, see the full version. This simple lemma is the key to our improved iteration bound.

Lemma 2.1. *For input ($U \in \mathbb{R}^{d\times n}, z \in \mathbb{R}^n_{++}, c \in \mathbb{R}^n_+$) let $T \subseteq [n]$ have margin γ according to Definition 2.3 and let $h := h_T^{U,z}$ be the progress function given in Definition 2.4. Then for any α satisfying $0 \le h(\alpha) - h(1) \le \gamma$, the scaling $z' := z \circ (1_{\overline{T}} + \alpha 1_T)$ results in*

$$\|\tau^U(z) - c\|_2^2 - \|\tau_j^U(z') - c\|_2^2 \ge 2\gamma(h(\alpha) - h(1)).$$

Note that by Proposition 2.1, we can always find a set with margin at least a polynomial fraction of the error $\|\tau - c\|_2$. Therefore if we could find $h(\alpha) - h(1) \ge \gamma/\operatorname{poly}(n, d)$ in each iteration, this would give $\log(1/\varepsilon)$ convergence. In the following Section 2.2, we show how to compute an update α satisfying the conditions of the above Lemma 2.1 and making $\Omega(\gamma)$ progress if possible.

Below we show that such an update always exists in the feasible case. For the proof, see the full version.

Proposition 2.2. *For progress function $h := h_T^{U,z}$ as in Definition 2.4,*

1. *h is an increasing function of α;*
2. *$\lim_{\alpha\to\infty} h(\alpha) = \operatorname{rk}(U_T)$;*
3. *$\langle c, 1_T\rangle - h(1) \ge \gamma$ where γ is the margin of set T according to Definition 2.3;*
4. *If $\operatorname{rk}(U_T) \ge \langle c, 1_T\rangle$, then for margin γ and any $\delta \in [0, \gamma)$, there is a finite solution $\alpha < \infty$ to the equation $h(\alpha) = h(1) + \delta$.*

Item (4) in the contrapositive implies that the only reason we cannot make sufficient progress $\Omega(\gamma)$ in the proxy function h is when the set T gives a certificate of infeasibility $\langle c, 1_T\rangle > \operatorname{rk}(U_T)$ according to Theorem 1.1. Combining this with the analysis in Lemma 2.1 shows that we can make $1/\operatorname{poly}(n)$ multiplicative progress in the error. In the following subsection, we show how to find such an update in strongly polynomial time.

2.2 Computing the Update The goal of this section is to compute an update $\hat{\alpha}$ for which we can apply Lemma 2.1 to get polynomial progress. Below we give the pseudocode and analysis for our update algorithm.

Theorem 2.2. *Given input (U, z, T, γ) that is feasible according to Proposition 2.2, Algorithm 2.2 runs in strongly polynomial time and finds $\hat{\alpha} \ge 1$ satisfying*

$$\gamma/5 \le h(\hat{\alpha}) - h(1) \le \gamma,$$

where $h := h_T^{U,z}$ according to Definition 2.4. Furthermore, the update satisfies $\hat{\alpha} \le 1 + O(\gamma)/\mu_{\min}$ where μ is the spectrum given in Lemma 2.2.

```
    **Update**$(U, z, T, \gamma)$:
    **Input**: $U \in \mathbb{R}^{d\times n}$, scaling $z$, $T \subseteq [n]$, margin $\gamma \in (0,1]$ satisfying $\sup_{\alpha\ge 1} h(\alpha) \ge h(1) + \gamma$;
    **Output**: $\hat{\alpha} \ge 1$ s.t. $\gamma/5 \le h(\hat{\alpha}) - h(1) \le \gamma$ for $h := h_T^{U,z}$ (Definition 2.4);
  **if** $h'(1) \ge \gamma/4$ **then**
    $\alpha_0 \leftarrow 1$;
  **else**
    $\tilde{\mu} \leftarrow$ Approx-Small-Eigen-Sum$(U, z, T)$ (Algorithm 3);
    $\alpha_0 \leftarrow 1 + \frac{\gamma}{2\bar{\mu}}$;
  **end if**
  Output ND$(h, \alpha_0, b' := h(1) + \gamma/4, b := h(1) + \gamma)$ (Algorithm 2.2);
```

We will use structural properties of the proxy function to compute our update, so for this purpose we rewrite h as a sum of simple constituent functions.

LEMMA 2.2. *For frame $U \in \mathbb{R}^{d\times n}$, scaling $z \in \mathbb{R}^n_{++}$, and set $T \subseteq [n]$, let $\mu_1 \geq ... \geq \mu_d$ be the eigenvalues of the matrix $(UZU^\mathsf{T})^{-1}U_T Z_T U_T^\mathsf{T}$. Then the function $h := h_T^{U,z}$ given in Definition 2.4 can be rewritten as*

$$h(\alpha) = \sum_{i=1}^{d} \frac{\alpha\mu_i}{1-\mu_i+\alpha\mu_i}, \qquad h(\alpha) - h(1) = \sum_{i=1}^{d} \frac{(\alpha-1)\mu_i(1-\mu_i)}{1+(\alpha-1)\mu_i}, \qquad h'(\alpha) = \sum_{i=1}^{d} \frac{\mu_i(1-\mu_i)}{(1+(\alpha-1)\mu_i)^2}.$$

If we had access to the values $\{\mu_i\}_{i\in[d]}$, then we could compute this piece-wise linear approximation $g(\alpha) \approx h(\alpha) - h(1)$ and solve for the correct value $g(\alpha) = \Omega(\gamma)$. This is the approach we take in the matrix scaling setting.

In the frame setting, we only have implicit access to the spectrum, so one approach would be to approximate the eigenvalues. It turns out that we can bypass this procedure and use a more implicit method to compute the solution. Our main tool will be the Newton-Dinkelbach method, which can be applied to compute approximate solutions to the functional equation $h(\alpha) - h(1) = \gamma$ when h is increasing and concave.

COROLLARY 2.1. *In the setting of Lemma 2.2, the proxy function h is increasing and concave.*

Proof. This follows simply from Lemma 2.2 by noting h is a sum of functions of the form $\frac{\alpha\mu}{1-\mu+\alpha\mu}$, which can be easily verified to be increasing and concave for $\mu \in [0,1]$. ◻

This motivates the Newton Dinkelbach method to find the solution satisfying the conditions of Lemma 2.1.

```
  **ND**$(f, \alpha_0, b', b)$:
  **Input**: Differentiable and increasing concave function $f : \mathbb{R} \to \mathbb{R}$, lower bound $b'$, upper bound $b$ satisfying $\sup_\alpha f(\alpha) \geq b$ and $b' < b$, starting guess $f(\alpha_0) \leq b$;

  **Output**: $\hat{\alpha}$ satisfying $f(\hat{\alpha}) \in [b', b]$;
  $t \leftarrow 0$;
  **while** $f(\alpha_t) < b'$ **do**
    $\alpha_{t+1} := \alpha_t + \frac{b-f(\alpha_t)}{f'(\alpha_t)}$;
    $t \leftarrow t+1$;
  **end while**
  Output $\alpha_t$;
```

The convergence of this method is well-known, and we used a refined iteration bound using Bregman divergence arguments similar to the work in [38]. For the proof, see the full version.

THEOREM 2.3. *Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable increasing concave function and let $b' < b$, and let $\alpha_* \in \mathbb{R}$ satisfy $f(\alpha_*) = b'$. Let α_0 be an initial guess satisfying $f(\alpha_0) \leq b$, and let $\varepsilon \in (0,1)$ satisfy $\varepsilon(b - f(\alpha_0)) \leq b - b'$. If $b' \leq f(\alpha_0) \leq b$, the algorithm returns α_0 directly. Otherwise if $f(\alpha_0) < b'$, the Newton-Dinkelbach method in Algorithm 2.2 applied with starting guess α_0 outputs $\hat{\alpha}$ satisfying $f(\hat{\alpha}) \in [b', b]$ in at most T iterations, where*

$$T := 1 + \left\lceil \log_{\frac{1}{1-\varepsilon}} \left(\max\left\{1, \frac{D_f(\alpha_*|\alpha_0)}{b-b'}\right\}\right)\right\rceil,$$

and $D_f(\beta|\alpha) := f'(\alpha)(\beta-\alpha) + f(\alpha) - f(\beta)$ is known as the Bregman divergence of f.

We will apply the method with $f = h^T_{U,z}$ to reach target $b', b = h(1) + \Omega(\gamma)$. The above analysis gives an explicit expression bounding the number of iterations of Newton-Dinkelbach that we can use to analyze Algorithm 2.2. In our setting, we will have somewhat explicit control on the function values, since we are searching for some progress $h(\alpha) - h(1) \geq \Omega(\gamma)$. We show a starting guess that is close to the solution.

Lemma 2.3. *Consider input (U, z, T) with progress function $h := h^T_{U,z}$ and gap $\gamma \in (0, 1]$. Assume $\lim_{\alpha\to\infty} h(\alpha) \geq h(1) + \gamma$, and let α_* satisfy $h(\alpha_*) = h(1) + \gamma/5$.*

1. *If $h'(1) \geq \gamma/4$, then $\alpha_* \leq 4$.*
2. *Otherwise, if $h'(1) < \gamma/4$, let $\mu_s := \sum_{i:\mu_i<1/2}$. Then $\mu_s > 0$ and $\alpha_* \leq 1 + \frac{\gamma}{\mu_s}$. More generally, for any $\alpha := 1 + \delta/\mu_s$ with $\delta \in [0, 1]$:*
$$\frac{\delta}{4} \leq h(\alpha) - h(1) \leq \frac{\gamma}{2} + \delta.$$

Using these bounds and the Newton-Dinkelbach method, we have reduced the problem of finding an update satisfying $h(\alpha) - h(1) \gtrsim \gamma$, to just computing an approximation for $\mu_s := \sum_{i:\mu_i<1/2} \mu_i$. Furthermore, as shown in the proof of the second item above, we have actually reduced to the case when there is a large gap between the eigenvalues $\{\mu_i \geq 1/2\}$ and $\{\mu_i < 1/2\}$. In the following Section 3, we show how to compute an approximation of μ_s in this gapped setting. The following are the guarantees:

Theorem 2.4. *Algorithm 3 runs in strongly polynomial time and outputs $\tilde{\mu}$ satisfying*

$$\sum_{i:\mu_i<1/2} \mu_i \leq \tilde{\mu} \leq (1 + 8nd^2) \sum_{i:\mu_i<1/2} \mu_i.$$

Combining the pieces above allows us to prove the guarantees of the update algorithm (see full version).

In the following section we show how to achieve the approximation guarantee required for the starting guess.

3 Guessing Starting Point

In this section, we present the guessing algorithm in 2.2 and its approximation guarantee:

```
    **Approx-Small-Eigen-Sum**$(U, z, T)$:
    **Input**: $U \in \mathbb{R}^{d\times n}, z \in \mathbb{R}^n_{++}, T \subseteq [n]$ with $\mu := \mathrm{spec}(U_T Z_T U_T^\top (UZU^\top)^{-1})$, $\sum_{i=1}^d \mu_i(1-\mu_i) < 1/4$;
    **Output**:$\tilde{\mu}$ satisfying $\sum_{i:\mu_i<1/2} \mu_i \leq \tilde{\mu} \leq (1 + 8nd^2) \sum_{i:\mu_i<1/2} \mu_i$.
$p \leftarrow \lfloor \mathrm{Tr}[U_T Z_T U_T^\top (UZU^\top)^{-1}] \rceil$;
**if** $p = \mathrm{rk}(V_T)$ **then**
    Output 0;
**else if** $p = 0$ **then**
    Output $\mathrm{Tr}[U_T Z_T U_T^\top (UZU^\top)^{-1}]$;
**else**
    $D \leftarrow \mathrm{DetLocalOpt}(U, z, T, p)$ (Algorithm 3);
    Output $\mathrm{Tr}[(I_d - U_D(U_D^\top (UZU^\top)^{-1} U_D)^{-1} U_D^\top (UZU^\top)^{-1}) U_T Z_T U_T^\top (UZU^\top)^{-1}]$;
**end if**
```

Theorem 3.1. (Restatement of Theorem 2.4) *Algorithm 3 outputs value $\tilde{\mu}$ satisfying*

$$\sum_{i:\mu_i<1/2} \mu_i \leq \tilde{\mu} \leq (1 + 8nd^2) \sum_{i:\mu_i<1/2} \mu_i,$$

where $\mu := \mathrm{spec}(U_T Z_T U_T^\top (UZU^\top)^{-1})$. Furthermore, the algorithm runs in strongly polynomial time and uses $O(nd^2 \log n)$ matrix computations involving $d \times d$ matrices (multiplication, determinant, inverse).

Throughout this section we will use shorthand $V := (UZU^\top)^{-1/2} UZ^{1/2}$. Note that $VV^\top = I_d$, $V^\top V$ is the orthogonal projection onto $\mathrm{im}(V^\top) = \mathrm{im}(\sqrt{Z}U^\top)$, and

$$\mathrm{spec}(V_T V_T^\top) = \mathrm{spec}((UZU^\top)^{-1/2} U_T Z_T U_T^\top (UZU^\top)^{-1/2}) = \mathrm{spec}(U_T Z_T U_T^\top (UZU^\top)^{-1}) = \mu.$$

Our goal is to compute the sum of eigenvalues $\mu_s := \sum_{i:\mu_i<1/2} \mu_i$ for $V_T V_T^\top$ in strongly polynomial time, where $\mu = \mathrm{spec}(V_T^\top V_T)$ as defined in Lemma 2.2. The following characterization of spectral projections gives some intuition for our guessing algorithm.

Theorem 3.2. (Ky-Fan) *For any symmetric matrix with eigendecomposition $X = \sum_{i=1}^d \lambda_i e_i e_i^\intercal$ and $\lambda \in \mathbb{R}^d$ in non-increasing order,*

$$\sum_{i=1}^k \lambda_i(X) = \sum_{i=1}^k \langle e_i, X e_i \rangle = \max_{\mathrm{rk}(P)=k} \mathrm{Tr}[PX],$$

where the max runs over all orthogonal projections of rank k.

Letting $X := V_T V_T^\intercal$, this tells us that we can approximate the spectral sum μ_s if we can guess the number $p := |\{i : \mu_i \geq 1/2\}|$ and then compute a projector $P \approx P_p$ which approximates the projector onto the top p eigenvalues. The conditions $V_T V_T^\intercal \preceq VV^\intercal = I_d$ and $\sum_{i=1}^d \mu_i(1-\mu_i) < 1/4$ guarantee that all of the eigenvalues are either close to 0 or close to 1, so $\mathrm{Tr}[V_T V_T^\intercal] \approx p$.

Furthermore, in the gapped setting, the vectors $v_{j\in T}$ are all close in Euclidean distance to the large eigenspace. Therefore we could hope that the span of a "representative" column subset gives a good approximation of the large eigenspace.

Formally, we want to choose a subset D such that all vectors $v_{j\in T}$ are close to $\mathrm{im}(V_D)$ in Euclidean distance, and we can use the determinant as a proxy for this property. Therefore, we will compute an approximate local optimum for subdeterminants, and use this as our "representative" set.

Definition 3.1. *Given input $\Pi \in \mathbb{R}^{m\times m}$, $D \subseteq [n]$ is a β-approximate local optimum for subdeterminant if*

$$\forall i \in D, j \notin D : \qquad \det(\Pi_{D-i+j,D-i+j}) \leq \beta \det(\Pi_{D,D}),$$

where $\Pi_{S,S}$ denotes the $S \times S$ principal submatrix.

The following classical algorithm of Knuth [2] allows us to compute the local optimum in strongly polynomial time. The guarantee follows, and we direct the reader to the full version for the proof.

```
DetLocalOpt($U, z, T, p$):
Input: $U \in \mathbb{R}^{d\times n}, z \in \mathbb{R}^n_{++}, T \subseteq [n]$, $0 < p < \mathrm{rk}(U_T)$, $\mathrm{Tr}[U_T Z_T U_T^\intercal (UZU^\intercal)^{-1}] \geq p - 1/2$;
Output: $\beta = 2$-local maximizer for determinant $D \in \binom{T}{p}$ according to Definition 3.1;
for $k = 1, ...p$ do
    $G \leftarrow \arg\max_{i\in T-G} \det(U_{G+i}^\intercal (UZU^\intercal)^{-1} U_{G+i} Z_{G+i})$;
end for
while True do
    $(i,j) \leftarrow \arg\max_{i\in D, j\in T-D} \det(U_{D-i+j}^\intercal (UZU^\intercal)^{-1} U_{D-i+j} Z_{D-i+j})$;
    if $\det(U_{D-i+j}^\intercal (UZU^\intercal)^{-1} U_{D-i+j} Z_{D-i+j}) \leq 2\det(U_D^\intercal (UZU^\intercal)^{-1} U_D Z_D)$ then
        Output $D$;
    end if
    $D \leftarrow D \triangle \{i,j\}$;
end while
```

Proposition 3.1. *The Greedy and Local Search Algorithm 3 applied to input (U, z, T, p) satisfying $p := \lfloor \mathrm{Tr}[U_T Z_T U_T^\intercal (UZU^\intercal)^{-1}] \rceil$ computes $\beta = 2$-approximate local maximum according to Definition 3.1. This can be implemented using $O(nd + nd^2 \log n)$ determinant computations on $d \times d$ sub-matrices of $U_T^\intercal (UZU^\intercal)^{-1} U_T Z_T$.*

In the remainder of this section we will analyze $V := (UZU^\intercal)^{-1/2} U Z^{1/2}$. This is so that we can deal with symmetric matrices and orthogonal projection. Note that $VV^\intercal = I_d$ and $\mathrm{spec}(V_T V_T^\intercal) = \mu$. Let eigendecomposition $V_T V_T^\intercal = \sum_{i=1}^d \mu_i e_i e_i^\intercal$, then we collect $E_b := [e_1, ..., e_p]$ be the matrix of top-p eigenvectors and E_s the remaining.

The following proposition then shows that the local optimum gives a good approximation of the quantity μ_s.

Proposition 3.2. *Let (U, z, T, p) be as in Algorithm 3 and $D \in \binom{T}{p}$ be a $\beta = 2$-local maximum for the subdeterminant according to Definition 3.1. For $V = (UZU^\intercal)^{-1/2} U Z^{1/2}$ as defined above, with $\mu = \mathrm{spec}(V_T V_T^\intercal)$,*

$$\sum_{i=p+1}^d \mu_i \leq \|(I_d - P_D) V_T\|_F^2 = \mathrm{Tr}[(I_d - P_D) V_T V_T^\intercal] \leq (1 + 8nd^2) \sum_{i=p+1}^d \mu_i$$

where $P_D := V_D (V_D^\intercal V_D)^{-1} V_D^\intercal$ is the orthogonal projection onto $\mathrm{im}(V_D)$.

Combining the analysis in the above two steps gives the proof of Theorem 3.1.

4 Regularization

In this section we explain the regularization step in Algorithm 2 that allows us to maintain bounded scalings. In particular, we will show that the algorithm is strongly polynomial by showing that the bit complexity of all scaling iterates can be bounded by $\text{poly}(n, b, \log(1/\varepsilon))$, where b is the bit complexity of the input vectors $U \in \mathbb{R}^{d\times n}$. We first show how to control the magnitude or condition number of the scaling iterates. Then we show how this implies strongly polynomial bit complexity by a simple rounding procedure. Finally, we discuss the relation to previous work.

We begin by presenting a condition measure of frames.

Definition 4.1. *For input $U \in \mathbb{R}^{d\times n}$, let*

$$\bar{\chi}_T(U) := \|((UU^\top)^{-1/2}U_T)^+\|_{\text{op}}, \qquad \bar{\chi}(U) := \max_{T\subseteq[n]} \bar{\chi}_T(U),$$

$$\rho_T(U) := \|((UU^\top)^{-1/2}U_T)^+\|_{\text{op}}^2 - 1, \qquad \rho(U) := \max_{T\subseteq[n]} \rho_T(U).$$

Note that these condition measures depend only on $\text{im}(U^\top)$, as $U^\top(UU^\top)^{-1/2}$ is an orthonormal basis for this subspace. We point the reader to [37] for other interpretations of these condition measures, as well as a more detailed bibliography of the relation to linear programming. This definition will be crucial in analyzing how the scalings change both in the update and regularization procedures. First, recall that Lemma 2.3 shows that the update in a single iteration is related to $\mu = \text{spec}((U_T Z_T U_T^\top)(UZU^\top)^{-1})$. It turns out that the update can be controlled in terms of the current scaling $z \in \mathbb{R}^n_{++}$ and the condition measure $\rho(U)$. In the next lemma, we show how to relate the spectrum μ to the condition measure $\rho(V)$ for frame $V := U\sqrt{Z}$. Further, the quantity ρ changes in a simple way under diagonal scalings. This will allow us to relate both the update and regularization to the condition measure of the underlying frame $U \in \mathbb{R}^{d\times n}$, independent of our current scaling $z \in \mathbb{R}^n_{++}$.

Fact 4.1. *For $V \in \mathbb{R}^{d\times n}$ and column subset $T \subseteq [n]$, let $\lambda_{\min}$ denote the smallest non-zero eigenvalue. Then*

$$(\lambda_{\min}(V_T^\top(VV^\top)^{-1}V_T))^{-1} = \|(V_T^\top(VV^\top)^{-1}V_T)^+\|_{\text{op}} = 1 + \rho_T(V).$$

We will use the above expression to bound the size of the update step for current iterate $V := U\sqrt{Z}$. In the next lemma, we show how this quantity changes under column scalings.

Lemma 4.1. *For frame $U \in \mathbb{R}^{d\times n}$ and scaling $z \in \mathbb{R}^n_{++}$,*

$$\forall T \subseteq [n] : \quad \rho_T(U\sqrt{Z}) \le \frac{\max_{j\notin T} z_j}{\min_{j\in T} z_j} \rho_T(U).$$

We can show that $\rho(U)$ is a singly exponential function of dimension and bit complexity of U. But in fact, the main message of this section is that ρ is a more refined complexity measure than bit complexity. For example if $A \in \mathbb{R}^{V\times E}$ is the node-edge incidence matrix of a (directed) graph, then A has entries of constant bit complexity, but it can be shown that $\rho(A) \le \text{poly}(|V|)$, which is significantly better than the naive $2^{\text{poly}(|V|)}$ bound derived just using bit complexity. As a further benefit, ρ is continuous and therefore is robust to perturbation.

Using these definitions, we can prove a bound on the growth in magnitude of scaling z in a single iteration.

Proposition 4.1. *For input U and scaling $z^{(t)}$, line 8 of Algorithm 2 produces update $z^{(t+1)}$ satisfying*

$$\|\log z^{(t+1)}\|_\infty \le 2\|\log z^{(t)}\|_\infty + \log\rho(U) + O(1).$$

This result tells us that if the iterate $z^{(t)}$ is bounded, then the update $z^{(t+1)}$ will also be bounded. Of course, applying this for many iterations could cause the magnitude to blow up. Therefore, in the remainder of this section we regularize our scalings so that the magnitude is bounded by a function of the measure ρ. Finally, we use a simple rounding procedure to maintain bounded bit complexity of scalings throughout the algorithm.

THEOREM 4.1. *Given frame $U \in \mathbb{R}^{d\times n}$, scaling $z \in \mathbb{R}^n_{++}$, and precision $0 < \delta \leq 1$, there is $\hat{z} \in \mathbb{R}^n_{++}$ such that*

$$\|\tau^U(z) - \tau^U(\hat{z})\|_1 \leq 2nd\delta \qquad \text{and} \qquad \|\log \hat{z}\|_\infty \leq n\log(\rho(U)/\delta).$$

Actually, it is well known that the quantity $\overline{\chi}$ (and therefore ρ) is NP-hard to approximate even to simply exponential accuracy [3]. But note that the procedure described in the proof above does not require the true value of ρ, and instead evaluates $\rho_T(U)$ for polynomially many sets $T \subseteq [n]$. In the full version, we show how to turn the above Theorem 4.1 into a strongly polynomial algorithm by using simple overestimates for ρ_T.

In order to bound the bit complexity, we first show that leverage scores are robust to small multiplicative perturbations of scalings.

LEMMA 4.2. *For input $U \in \mathbb{R}^{d\times n}$ and scaling $z \in \mathbb{R}^n_{++}$, if $\hat{z} \in (1 \pm \delta)z$ for $0 \leq \delta < 1/2$, then*

$$\tau^U(\hat{z}) \in (1 \pm 3\delta)\tau^U(z).$$

Proof. This follows by a straightforward Taylor approximation

$$\tau^U_j(\hat{z}) = \hat{z}_j u_j^\mathsf{T}(U\hat{Z}U^\mathsf{T})^{-1}u_j \in (1 \pm \delta)z_j u_j^\mathsf{T}((1 \pm \delta)UZU^\mathsf{T})^{-1}u_j \in (1 \pm 3\delta)\tau^U_j(z).$$

where in the second step we used $\hat{z}_j \in (1 \pm \delta)z_j$ and $\hat{Z} \in (1 \pm \delta)Z$ which implies $U\hat{Z}U^\mathsf{T} \in (1 \pm \delta)UZU^\mathsf{T}$ and therefore $(U\hat{Z}U^\mathsf{T})^{-1} \in (1 \pm \delta)^{-1}(UZU^\mathsf{T})$, and in the final step we used $|\frac{1+\delta}{1-\delta} - 1| \leq 3\delta$ for $0 < \delta < 1/2$. ◻

The following well-known result allows us bound ρ in terms of bit complexity.

THEOREM 4.2. (THEOREM 6 IN [10]) *For $U \in \mathbb{R}^{d\times n}$ with each entry having bit complexity b,*

$$\log \overline{\chi}(U) \leq O(d(b + \log d) + \log n), \qquad \log \rho(U) \leq O(d(b + \log d) + \log n).$$

As a consequence, we can maintain bounded bit complexity for all intermediate scalings. We note that the procedure below is the only time we require the 'rounding' oracle described in Section 6.

COROLLARY 4.1. *Given any frame $U \in \mathbb{R}^{d\times n}$ of bit complexity b, for any scaling $z \in \mathbb{R}^n_{++}$ and precision $0 < \delta < 1/2$, there is $\hat{z} \in \mathbb{R}^n_{++}$ with (entry-wise) bit complexity $O(n(db + \log n + \log(1/\delta)))$ that satisfies $\|\tau^U(z) - \tau^U(\hat{z})\|_1 \leq O(nd\delta)$. Further, $\hat{z}$ can be computed in strongly polynomial time, involving $O(n \log n)$ comparisons and $O(n)$ matrix operations on $d \times d$ and $d \times n$ matrices (Gram-Schmidt, inverse, multiplication).*

In the remainder of this section we will compare our regularization Theorem 4.1 to similar results in the literature on algorithms for frame scaling.

The strongly polynomial algorithm of [9] for Forster transformation (the special case of Definition 1.1 with $c = \frac{d}{n}1_n$) maintains left scaling $L \in \mathbb{R}^{d\times d}$ while keeping the right scaling implicit. The regularization result given in Theorem 5.1 of [9] is therefore much more complicated, as they need to show how to round the left scaling matrix $L \in \mathbb{R}^{d\times d}$ to bounded bit complexity while maintaining small error with respect to the frame scaling problem Definition 1.1.

All other results come from weakly polynomial algorithms from frame scaling. The line of works [6], [13], [34], [33], [32] all study a convex formulation for frame scaling and show that, for any desired precision $\delta > 0$, there exists a scaling achieving optimality gap δ with magnitude bounded by a function of $1/\delta$. This is then combined with off-the-shelf optimization methods to give a weakly polynomial algorithm to solve the convex formulation.

In our work, we require an algorithm to produce a bounded scaling, but with the condition that the marginals are close instead of small optimality gap. For this reason, we cannot apply the previous results as those scaling bounds are usually existential. We believe Theorem 4.1 is of independent interest, since it relates the bound to the refined ρ parameter.

5 Improvement of Matrix Scaling

In this section we show how are techniques give a faster strongly polynomial algorithm for the matrix scaling problem. In this setting, we are given $A \in \mathbb{R}^{m\times n}_+$ and want to find diagonal scalings $L \in \text{diag}_{++}(m), R \in \text{diag}_{++}(n)$ such that $B := LAR$ satisfies $B1_n = r, B^\mathsf{T}1_m = c$. Formally:

Definition 5.1. (Matrix Scaling Problem) *Given input $A \in \mathbb{R}_+^{m\times n}$, let $(r(A), c(A)) \in \mathbb{R}^m \times \mathbb{R}^n$ be the row and column sums. Explicitly,*

$$r_i(A) := \sum_{j\in[n]} A_{ij}, \qquad c_j(A) := \sum_{i\in[m]} A_{ij}.$$

Given desired marginals $(r, c) \in \mathbb{R}_+^m \times \mathbb{R}_+^n$ and precision $\varepsilon > 0$, find diagonal scalings $L \in \mathrm{diag}_{++}(m), R \in \mathrm{diag}_{++}(n)$ such that A is ε-approximately (r, c)-scaled:

$$\|r(A) - r\|_2^2 + \|c(A) - c\|_2^2 \le \varepsilon^2.$$

In [1], the authors give the first strongly polynomial algorithm for this problem. By adapting our frame scaling analysis to this setting, we are able to reduce the number of iterations from $O(n^5 \log(n/\varepsilon))$ as given in [1] to $O(n^3 \log(n/\varepsilon))$ iterations, matching our Main Theorem 2.1. Further, each iteration can still be performed in nearly linear time, so this gives a quadratic improvement in runtime over the algorithm in [1]. In order to deal with the polynomial bit complexity requirement, the work of [1] does not use a rounding step, but instead maintains quantities in floating point representation and argues that the exponent remains of polynomial bit size. We show that we can also adapt our regularization procedure from Section 4 to the matrix setting, giving a new stronger bound on the size and bit complexity of scalings required for ε-approximate matrix scaling. This also allows us to avoid the complications that arise from merging floating point precision with the real model as discussed in Section 6.

Our main result is as follows:

Theorem 5.1. *Given input $A \in \mathbb{R}_+^{m\times n}$ with desired marginals $(r, c) \in \mathbb{R}_{++}^m \times \mathbb{R}_{++}^n$ satisfying $s := \langle r, 1_m\rangle = \langle c, 1_n\rangle$, and precision $\varepsilon > 0$, there is a strongly polynomial algorithm to produce ε-approximate scalings or a certificate of infeasibility. This algorithm takes $O(n^3 \log(sn/\varepsilon))$ iterations, each of which require linear $\tilde{O}(nnz(A))$ arithmetic operations (here $nnz(A)$ is the number of non-zero entries of A, and the intermediate scalings all have bit complexity bounded by $\mathrm{poly}(n, m, b) \log(1/\varepsilon)$.*

We note that the algorithm follows the same outline as our main algorithm 2 for frame scaling, but the implementation of each step is much simpler since we have explicit access to the input. We leave the proof to the full version.

6 Strongly Polynomial Models of Computation

In this section, we discuss the precise sense in which our algorithm is strongly polynomial. We will also place this model in context with other commonly used computational models and give motivation for its use. Some of the below descriptions are inspired by the recent survey of Srivastava [19] on the complexity of eigendecomposition.

1. **Real Model**: the input is given as n real numbers; the algorithm is allowed oracle access to exact arithmetic gates, which take constant time; here, a polynomial time algorithm is one that makes $\mathrm{poly}(n)$ calls to the arithmetic oracle. This model is studied in complexity theory and pure mathematics.

2. **Bit model**: the input is given as n numbers each consisting of b bits; the algorithm is able to perform exact arithmetic operations, but the cost is the number of bit operations; since bit size can grow with each operations, the algorithm is allowed the freedom to round arbitrarily; here, a polynomial time algorithm performs $\mathrm{poly}(n, b)$ bit operations. Note that this automatically enforces a $\mathrm{poly}(n, b)$ space constraint on all intermediate quantities. This is the most prominent computational model used to define tractable problems from the perspective of theoretical computer science.

3. **Finite precision model**: the input is given as n numbers in floating point with fixed machine precision b bits; the algorithm is allowed arithmetic operations but they are only performed approximately up to adversarial multiplicative error that depends on the machine precision; here once again a polynomial time algorithm performs $\mathrm{poly}(n, b)$ bit operations, but because all intermediate quantities remain at fixed precision, this is equivalent to performing $\mathrm{poly}(n, b)$ arithmetic operations. This is the dominant model used in numerical analysis.

As described in Section 1.3, the strongly polynomial model is a blend of the real and bit models: arithmetic operations are performed using exact arithmetic and only take constant time, but the bit complexity of intermediate quantities still needs to remain polynomially bounded in the input complexity.

The prototypical strongly polynomial algorithm in linear algebra is the computation of the determinant. In practice, this is performed in the finite precision model by applying Gaussian elimination to reduce to row-echelon form and computing an approximation of the determinant that depends on machine precision. But this naive algorithm is not strongly polynomial, as it is well-known that Gaussian elimination with exact arithmetic and arbitrary pivots could lead to an exponential size blow up in the bit complexity [22]. In [23], Edmonds gave a simple Schur-complement style algorithm that computes the determinant in the real model. Furthermore, all intermediate quantities are $k \times k$ subdeterminants of the original matrix for $k \in [n]$, so have polynomial bit complexity in the original input. This algorithm therefore satisfies all strongly polynomial requirements.

Smale [24] famously conjectured the existence of strongly polynomial algorithms for general linear programs. Several such algorithms have been developed for special classes of LP's, such as flow problems on graphs, but the general case is still open. In [37], the authors build an interior point framework for LP whose complexity depends only on the constraint matrix. This gives a unified strongly polynomial algorithm for many known important cases of LP.

The algorithms we consider in this paper almost fit into this model, but require slightly extra power. Specifically, we consider the stronger version of strong polynomial as defined in section 1.3 of [30] where we are given an additional 'rounding oracle': given input $x \in \mathbb{R}, b \in \mathbb{N}$, output x rounded to the nearest multiple of 2^{-b}. Note that this is not oblivious to the bit representation of the input, and so does not fit into the real model interpretation of strongly polynomial. We require this extra power only in our regularization step in Section 4, and otherwise we only require exact arithmetic operations. This difficulty is encountered in many other other strongly polynomial algorithms, such as the original matrix scaling algorithm of [1]. We believe it may be possible to avoid this rounding step by maintaining scalings as small complexity combinations of some 'canonical scalings', and we leave this as an interesting open problem.

7 Application to Interior Point Methods

In this section we discuss the relation of our work to algorithms for solving linear programs. We begin by recalling the setup of linear programs and path following interior point methods. Then we give a brief description of more sophisticated interior point methods that require are able to decrease the number of iterations required by 'shortcutting' certain segments of the path. This naturally motivates the lifting map, which is used crucially throughout the work of [37]. Finally, we discuss the relation of our frame scaling problem to the lifting map, and show that our guessing algorithm 3 can be re-interpreted as giving a subspace with good properties for each IPM update step. Importantly our algorithm is strongly polynomial and does not require square roots for this operation.

The problem we study is finding the exact optimal solution to a linear program in standard form:

$$\min_{x\geq 0} c^{\mathsf{T}}x \qquad\qquad \max_{s\geq 0} b^{\mathsf{T}}y$$
$$Ax = b; \qquad\qquad c = A^{\mathsf{T}}y + s.$$

For any feasible starting point (x_0, y_0, s_0), the set of feasible updates satisfy

$$Ax = b = Ax_0 \implies x - x_0 \in \ker(A), \qquad s - A^{\mathsf{T}}y = c = s_0 - A^{\mathsf{T}}y_0 \implies s - s_0 \in \operatorname{im}(A^{\mathsf{T}}).$$

This makes clear the importance of the two subspaces $\ker(A), \operatorname{im}(A^{\mathsf{T}})$.

Interior point methods solve regularized versions of the above LP's, iteratively decreasing the regularization parameter and computing updates. The optimizers to the regularized problem are parametrized by $\mu > 0$ and form the 'central path':

$$Ax(\mu) = b, \quad A^{\mathsf{T}}y(\mu) + s(\mu) = c, \quad x(\mu), s(\mu) > 0, \qquad x(\mu) \circ s(\mu) = \mu 1_n.$$

Standard path following methods attempt to iteratively decrease the parameter μ at a geometric rate, achieving $\log(1/\varepsilon)$ convergence in terms of distance to optimality. This is weakly polynomial, and further, the true optimal solution is really only achieved at $\mu = 0$, so to compute this solution an additional combinatorial rounding step is required.

In order to improve runtime or iteration complexity of path-following methods, we could try to shortcut the central path. For intuition, assume that we had the optimal partition $B \cup N = [n]$ (basis and non-basic variables), such that $x^*_N \equiv 0, s^*_B \equiv 0$ for optimal solution (x^*, s^*). Starting at any feasible (x, s), if we focus on these coordinates and consider subspaces $(\pi_N(\ker(A)), \pi_B(\operatorname{im}(A^\mathsf{T})))$ where π_S is the projection onto coordinates $S \subseteq [n]$, then the following least squares problems

$$\min_{\delta_x \in P_N \ker(A)} \|x_N + \delta_x\|_2^2, \qquad \min_{\delta_s \in P_B \operatorname{im}(A^\mathsf{T})} \|s_B + \delta_s\|_2^2,$$

have optimum value 0. If we then figured out how to lift the partial solutions $(\delta_x, \delta_s) \to (\Delta x, \Delta s) \in (\ker(A), \operatorname{im}(A^\mathsf{T}))$, then we necessarily arrive at an optimal solution, as $(x + \Delta x, s + \Delta s)$ is feasible and satisfies complementary slackness.

This motivates the following definition of lifting map:

Definition 7.1. (Def 2.2 in [37]) *For subspace $W \subseteq \mathbb{R}^n$ and partition $B \cup N = [n]$, the lifting map for coordinates N is $L_N^W : \pi_N(W) \to W$ defined as*

$$L_N^W(y) = \arg\min\{\|w\|_2 \quad | \quad w \in W, w|_N = y\}.$$

We also define $\ell_N^W(y) \in \mathbb{R}^B$ to be the restriction of $L_N^W(y)$ to the B coordinates.

Starting with the work of Vavasis and Ye [11], the use of these subspace have been a crucial component of strongly polynomial IPMs for linear programming. Specifically, they allow for IPMs that take very long steps. In [11], they only use $V = W$ always, so the only choice was that of the partition (B, N). In the recent work of Allamigeon et al [12], which provided an IPM which for a given partition (B, N), they designed IPM steps that required the use of cheap lifting subspace V of maximum dimension (note that $V = \{0\}$ is always a possible choice), which was computed by one of two possible approximate singular value decomposition algorithm. Unfortunately, neither algorithm provably fit the strongly poly model, the first could not be shown to be numerically stable and the second required the computation of square roots (and randomization). Building on the techniques in Section 3, we give a deterministic, numerically stable algorithm for computing the desired subspace that does not require square roots.

As we cannot compute square-roots in strongly polynomial time, we will instead output unscaled subspaces $S \subseteq W$. To analyze the lifting costs of these subspaces, we will use the following weighted norms:

Definition 7.2. (Rescaled Norm) *Given $z \in \mathbb{R}^n_{++}$, we define the z-norm for $x \in \mathbb{R}^n$ as $\|x\|_z^2 := \sum_{j=1}^n z_i x_j^2$.*

The achievable values of ν will be related to the singular values of lifting map $\ell_N^{\sqrt{Z}W}$, where singular values are defined as follows.

Definition 7.3. (Singular Values) *A linear operator $T : U \to V$, where $U \subseteq \mathbb{R}^m$ and $V \subseteq \mathbb{R}^n$ are linear subspaces, admits a singular values decomposition $T = \sum_{i=1}^k \sigma_i v_i u_i^\mathsf{T}$, $k = \operatorname{rk}(T)$, where $v_1, \dots, v_k \in V$ and $u_k, \dots, u_k \in U$ are orthonormal vectors in their respective subspaces and $\sigma_1 \geq \cdots \geq \sigma_k > 0$. We define $\sigma(T) := (\sigma_1, \dots, \sigma_k)$ to be the vector of singulars values of T listed in non-increasing order. By convention, we define $\sigma_i = 0$ for $i > \operatorname{rk}(T)$. For a matrix $M \in \mathbb{R}^{n \times m}$, we define its singular to be those of the induced linear operator from $\mathbb{R}^m$ to $\mathbb{R}^n$.*

The main result of this section gives a strongly polynomial algorithm to compute cheap lifting subspaces, whose lifting cost and dimension are expressed in terms of the singular values of the lifting map.

Theorem 7.1. *For $U \in \mathbb{R}^{d \times n}$, $W := \operatorname{im}(U^\mathsf{T})$, $z \in \mathbb{R}^n_{++}$, and $B \cup N = [n]$ be a partition, let $\sigma := \sigma(\ell_N^{\sqrt{Z}W})$ be the singular values of the lifting map $\ell_N^{\sqrt{Z}W}$, as in Definition 7.1, and let $r = \operatorname{rk}(U_N)$. Then, there exists a strongly polynomial time algorithm on input U, z, N, B as above, and threshold ν, $0 < \nu \leq \frac{1}{16rn}$, that outputs either:*

1. *"not polarized", in which case $\exists i \in [r] : \nu/r < \sigma_i^2 < r/\nu$; or*
2. *$Y \in \mathbb{R}^{n \times p}$ with $\operatorname{rk}(Y) = p := |\{i \in [r] : \sigma_i \leq 1\}|$ and $S' := \operatorname{im}(Y) \subseteq W$ satisfying*

$$\forall y \in \operatorname{im}(Y) : \quad \|y_B\|_z^2 \leq 2\sigma_{\leq 1}^2 \|y_N\|_z^2$$

where $\sigma_{\leq 1}^2 := \max\{\sigma_i^2 : i \in [r], \sigma_i \leq 1\} \leq 2\nu$.

8 Applications to Learning Theory

In this section we discuss two previous works that apply frame scaling in the context of machine learning. In particular, we will show how our new algorithm for frame scaling can be used to improve the runtime of the learning algorithms of [9] and [8].

8.1 Halfspace Learning [9] We first discuss the application of frame scaling to halfspace learning. We begin by formally stating the problem. In [9], the goal is to learn an unknown affine halfspace defined by normal vector $w \in \mathbb{R}^d$. The algorithm gets labeled samples from an unknown distribution $(x, y) \sim \mathcal{D}$, where $x \in \mathbb{R}^d, y = \text{sign}(\langle w, x\rangle)$. And the main result is as follows:

THEOREM 8.1. (THEOREM 1.6 IN [9]) *Given unknown halfspace defined by normal vector $w \in \mathbb{R}^{d+1}$ and unknown distribution $\mathcal{D}$ as above, there is a strongly polynomial algorithm that, for any $\delta > 0$, takes* $\text{poly}(d/\delta)$ *samples from $\mathcal{D}$ and with high probability outputs a hypothesis $f : \mathbb{R}^d \to \{\pm 1\}$ satisfying*

$$Pr_{(x,y)\sim\mathcal{D}}[f(x) \neq y] \leq \delta.$$

They are also able to extend this result to the inhomogeneous setting, where the halfspace has threshold $\{x \mid \langle w, x\rangle \geq t\}$, as well certain noisy models, where the labels only agree with the halfspace on some large fraction of samples (see Theorem 1.8 in [9]). Our new frame scaling algorithm can also be applied in the simpler homogeneous setting with exact labels, and the ideas can be extended in a straightforward way.

Frame scaling is helpful in this setting for two key reasons: (1) points with large "margin" can be easily classified; (2) for point sets in approximate $(I_d, c = \frac{d}{n}1_n)$-position, a large fraction of points have large "margin". These two observations are formalized in Lemma 7.2 and 7.3 in [9] respectively. These technical lemmas can also be appropriately modified to work with our implicit frame scaling representation, and we leave the proof to the full version.

We first show that the large "margin" classifier can be modified to work in strongly polynomial time for arbitrary inner product. We will apply this for inner product $\langle u, v\rangle_Q := u^{\mathsf{T}}(UZU^{\mathsf{T}})^{-1}v$ for our frame $U \in \mathbb{R}^{d\times n}$ and scaling $z \in \mathbb{R}^n$. This follows the "Improved Perceptron" algorithm given in Dunagan and Vempala [43].

LEMMA 8.1. (LEMMA 7.2 IN [9] (PERCEPTRON)) *Consider n labeled examples $\{(u_j, y_j) \in \mathbb{R}^d \times \{\pm1\}\}_{j\in[n]}$, such that there is an unknown linear halfspace satisfying $\forall j \in [n] : y_j = sign(\langle w, u_j\rangle_Q)$ for some inner product $\langle\cdot,\cdot\rangle_Q$. Then for any $\gamma \in (0, 1)$ and initial guess v_0 satisfying $\langle v_0, w\rangle_Q > 0$, there is an algorithm requiring* $\log_{1-\gamma^2}(\frac{\langle v_0,w\rangle_Q^2}{\|v_0\|_Q^2\|w\|_Q^2})$ *iterations that outputs guess v_T such that*

$$\langle v_T, u_j\rangle_Q^2 \geq \gamma^2\|v_T\|_Q^2\|u_j\|_Q^2 \implies sign(\langle v_T, u_j\rangle) = y_j.$$

Next, if the points are in approximate radial isotropic position, then a large fraction of the points will have large margin.

LEMMA 8.2. (LEMMA 7.3 IN [9]) *Consider frame $U \in \mathbb{R}^{d\times n}$ with scaling $z \in \mathbb{R}^n_{++}$ is in ε-approximate $(I_d, c = \frac{d}{n}1_n)$-position according to Definition 1.1 with $\varepsilon \leq \frac{d}{2n}$, and inner product $\langle u, v\rangle_Q := u^{\mathsf{T}}(UZU^{\mathsf{T}})^{-1}v$. Then for any $w \in \mathbb{R}^d$,*

$$T := \left\{j \in [n] : \frac{\langle w, u_j\rangle_Q^2}{\|w\|_Q^2\|u_j\|_Q^2} \geq \frac{1}{4d}\right\} \implies \quad \frac{|T|}{n} \geq \frac{1}{5d}.$$

Therefore, we can use our improved frame scaling algorithm in Theorem 2.1 to improve the runtime of the strongly polynomial improper halfspace learner given in [9]. In particular, we note that our implicit representation of the left and right scaling for approximate radial isotropic position can still be used with the perceptron algorithm to learn large margin points.

8.2 Point Location [8] In this subsection, we discuss the application of frame scaling to point location problem studied in [8]. Here the input is a fixed set of n points and we are given query access to an unknown halfspace in the form $x \to \text{sign}(\langle w, x\rangle)$. The goal is to learn the label of all n points using as few queries as possible.

In the following, we discuss how frame scaling is used for this problem in [8]. The first observation is quite similar to that of [9]: if the points are in approximate (I_d, c)-position according to Definition 1.1, then there is a simple way to learn the label of a large fraction of points. The technical statement is as follows:

LEMMA 8.3. (LEMMA 4.6 IN [8] (INFORMAL)) *Let S be a set of n labeled examples $(x, y) \in \mathbb{R}d \times \{\pm 1\}$, such that there is an unknown linear halfspace satisfying $\forall (x, y) \in S : y = sign(\langle w, x\rangle)$. Further, assume the input is in approximate (I_d, c)-position according to Definition 1.1. Then there is an algorithm "IsoLearn" with the guarantee: for some integer $k \in [d]$, the algorithm makes $\tilde{O}(k)$ queries and correctly classifies a set of points $T \subseteq [n]$ with weight $\langle c, 1_T\rangle \geq k$.*

In other words, we can learn a large weighted fraction of points according to the weighting c. In order to learn all the labels, this is combined with a boosting procedure, which iteratively increases the weighting of points whose label we have not yet learned (described in Algorithm 5 in [8]). Therefore, it is important that we can compute frame scalings for arbitrary weighting $c \in \mathbb{R}^n_{++}$ as given in Definition 1.1.

Using our improved frame scaling result in Theorem 2.1, we are able to give a strongly polynomial time procedure that can be used to place points in approximate (I_d, c)-position or give a certificate of infeasibility. Recall that we keep track of the square of the right scaling and leave the left scaling is implicit. Following the ideas in the previous subsection, this is sufficient if we switch to an appropriate inner product that depends on our scalings.

There are a few steps in the learning algorithms given in this work that are not strictly strongly polynomial, as the main goal is to give optimal query complexity for this problem. In particular, we are required to query the halfspace on linear combinations of our points in approximate (I_d, c)-position. We can once again simulate the left scaling by changing the inner product $\langle x, x'\rangle - Q := x^\mathsf{T}(UZU^\mathsf{T})^{-1}x'$; but again for linear combinations we will require square root operations. Further, a key technique in this work is the notion of margin oracle, which requires queries on random Gaussian linear combinations of points. Both of these are not strongly polynomial, and we leave to future work the question of whether these issues can be avoided.

References

[1] Linial, N., Samorodnitsky, A. & Wigderson, A. A deterministic strongly polynomial algorithm for matrix scaling and approximate permanents. *Combinatorica.* **20**, 545-568 (2000)

[2] Knuth, D. Semi-optimal Bases for Linear Dependencies. *Linear And Multilinear Algebra.* **17** (1985)

[3] Tuncel, L. Approximating the complexity measure of Vavasis-Ye algorithm is NP-hard. *Mathematical Programming.* (1999)

[4] Kachiyan, L. On the complexity of Approximating Extremal Determinants in Matrices. *Journal Of Complexity.* **11** (1995)

[5] Forster, J. A linear lower bound on the unbounded error probabilistic communication complexity. *Journal Of Computer And System Sciences.* **65** (2002)

[6] Hardt, M. & Moitra, A. Algorithms and hardness for robust subspace recovery. *26th Annual Conference On Learning Theory (COLT).* (2013)

[7] Barthe, F. On a reverse form of the Brascamp-Lieb inequality. *Inventiones Mathematicae.* **134** (1998)

[8] Hopkins, M., Kane, D., Lovett, S. & Mahajan, G. Point location and active learning: Learning halfspaces almost optimally. *61st IEEE Annual Symposium On Foundations Of Computer Science (FOCS).* (2020)

[9] Diakonikolas, I., Tzamos, C. & Kane, D. A strongly polynomial algorithm for approximate forster transforms and its application to halfspace learning. *Proceedings Of The 55th Annual ACM Symposium On Theory Of Computing.* pp. 1741-1754 (2023)

[10] Vavasis, S. & Ye, Y. Condition numbers for polyhedra with real number data. *Operations Research Letters.* (1995)

[11] Vavasis, S. & Ye, Y. A primal-dual interior point method whose running time depends only on the constraint matrix. *Mathematical Programming.* (1996)

[12] X, P, B., S, G. & M, J. Log-barrier interior point methods are not strongly polynomial. *SIAM Journal On Applied Algebra And Geometry.* (2018)

[13] Artstein-Avidan, S., Kaplan, H. & Sharir, M. On Radial Isotropic Position: Theory and Algorithms. *ArXiv Preprint ArXiv:2005.04918.* (2020)

[14] Diakonikolas, I., Kane, D. & Tzamos, C. Forster decomposition and learning halfspaces with noise. *2021 Conference On Neural Information Processing Systems (NeurIPS).* (2021)

[15] Allen-Zhu, Z., Garg, A., Li, Y., Oliveira, R. & Wigderson, A. Operator Scaling via Geodesically Convex Optimization, Invariant Theory and Polynomial Identity Testing. *Proceedings Of The 50th Annual ACM SIGACT Symposium On Theory Of Computing (STOC).* (2018)

[16] Cohen, M., Madry, A., Tsipras, D. & Vladu, A. Matrix Scaling and Balancing via Box Constrained Newton's Method and Interior Point Methods. *2017 IEEE Symposium On Foundations Of Computer Science (FOCS).* (2017)

[17] Idel, M. A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps. *ArXiv Preprint ArXiv:1609.06349.* (2016)

[18] Allen-Zhu, Z., Li, Y., Oliveira, R. & Wigderson, A. Much Faster Algorithms for Matrix Scaling. *2017 IEEE 58th Annual Symposium On Foundations Of Computer Science (FOCS).* (2017)

[19] Srivastava, N. The Complexity of Diagonalization. *Proceedings Of The International Symposium On Symbolic And Algebraic Computation (ISSAC).* (2023)

[20] Garg, A., Gurvits, L., Oliveira, R. & Wigderson, A. A deterministic polynomial time algorithm for non-commutative rational identity testing. *ArXiv Preprint ArXiv:1511.03730.* (2015)

[21] Cohen, M., Madry, A., Tsipras, D. & Vladu, A. Matrix Scaling and Balancing via Box Constrained Newton's Method and Interior Point Methods. *2017 IEEE Symposium On Foundations Of Computer Science (FOCS).* (2017)

[22] Fang, X. & Havas, G. On the Worst-case Complexity of Integer Gaussian Elimination. *Proceedings Of The International Symposium On Symbolic And Algebraic Computation (ISSAC).* (1997)

[23] Edmonds, J. Systems of Distinct Representatives and Linear Algebra. *Journal Of Research Of The National Bureau Of Standards.* (1967)

[24] Smale, S. Mathematical problems for the next century. *The Mathematical Intelligencer.* (1998)

[25] Garg, A., Gurvits, L., Oliveira, R. & Wigderson, A. Algorithmic and optimization aspects of Brascamp-Lieb inequalities, via Operator Scaling. *Geometric And Functional Analysis.* **28** (2018)

[26] Bürgisser, P., Franks, C., Garg, A., Oliveira, R., Walter, M. & Wigderson, A. Efficient algorithms for tensor scaling, quantum marginals, and moment polytopes. *2018 IEEE Symposium On Foundations Of Computer Science (FOCS).* (2018)

[27] Kempf, G. & Ness, L. The length of vectors in representation spaces. *Algebraic Geometry.* pp. 233-243 (1979)

[28] Bhatia, R. Positive Definite Matrices. (Princeton University Press,2007)

[29] Casazza, Peter G., Kutyniok, Gitta. Finite Frames: Theory and Applications. (Birkhauser Basel, 2013)

[30] Grötschel, M., Lovász, L. & Schrijver, A. Geometric Algorithms and Combinatorial Optimization. (Springer,1993)

[31] Bürgisser, P., Franks, C., Garg, A., Oliveira, R., Walter, M. & Wigderson, A. Towards a theory of non-commutative optimization: geodesic 1st and 2nd order methods for moment maps and polytopes. *2019 IEEE 60th Annual Symposium On Foundations Of Computer Science (FOCS).* pp. 845-861 (2019)

[32] Burgisser, P., Li, Y., Nieuwboer, H. & Walter, M. Interior-point methods for unconstrained geometric programming and scaling problems. *ArXiv Preprint ArXiv:2008.12110.* (2020)

[33] Straszak, D. & Vishnoi, N. Maximum entropy distributions: Bit complexity and stability. *Proceedings Of The Thirty-Second Conference On Learning Theory (COLT).* (2019)

[34] Singh, M. & Vishnoi, N. Entropy, optimization and counting. *Proceedings Of The Forty-Sixth Annual Symposium On Theory Of Computing (STOC).* (2014)

[35] Megiddo, N. Towards a genuinely polynomial algorithm for linear programming. *SIAM Journal On Computing.* **12** (1983)

[36] Tardos, E. A strongly polynomial minimum cost circulation algorithm. *Combinatorica.* **5** (1985)

[37] Dadush, D., Huiberts, S., Natura, B. & Vegh, L. A scaling-invariant algorithm for linear programming whose running time depends only on the constraint matrix. *Proceedings Of The 52nd Annual ACM Symposium On Theory Of Computing (STOC).* (2020)

[38] Dadush, D., Koh, Z., Natura, B. & Végh, L. An Accelerated Newton-Dinkelbach Method and Its Application to Two Variables per Inequality Systems. *29th Annual European Symposium On Algorithms (ESA 2021).* **204** pp. 36:1-36:15 (2021), https://drops.dagstuhl.de/opus/volltexte/2021/14617

[39] Dinkelbach, W. On nonlinear fractional programming. *Management Science.* **13** (1967)

[40] Radzik, T. Newton's method for fractional combinatorial optimization. *Proceedings Of The 33rd Annual Symposium On Foundations Of Computer Science (FOCS).* (1992)

[41] Holmes, R. & Paulsen, V. Optimal frames for erasures. *Linear Algebra And Its Applications.* (2004)

[42] Gelfand, I., Goresky, R., MacPherson, R. & Serganova, V. Combinatorial geometries, convex polyhedra, and schubert cells. *Advances In Mathematics.* **63** (1987)

[43] Dunagan, J. & Vempala, S. A simple polynomial-time rescaling algorithm for solving linear programs. *Proceedings Of The 36th Annual ACM SIGACT Symposium On Theory Of Computing (STOC).* (2004)