Acceleration Meets Inverse Maintenance: Faster ℓ_∞-Regression

Deeksha Adil
ETH Zürich, Switzerland

Shunhua Jiang
Columbia University, New York, NY, USA

Rasmus Kyng
ETH Zürich, Switzerland

Abstract

We propose a randomized multiplicative weight update (MWU) algorithm for ℓ_∞ regression that runs in $\widetilde{O}(n^{2+1/22.5}\text{poly}(1/\epsilon))$ time when $\omega = 2 + o(1)$, improving upon the previous best $\widetilde{O}(n^{2+1/18}\text{poly}\log(1/\epsilon))$ runtime in the low-accuracy regime. Our algorithm combines state-of-the-art inverse maintenance data structures with acceleration. In order to do so, we propose a novel acceleration scheme for MWU that exhibits *stability* and *robustness*, which are required for the efficient implementations of the inverse maintenance data structures.

We also design a faster *deterministic* MWU algorithm that runs in $\widetilde{O}(n^{2+1/12}\text{poly}(1/\epsilon))$ time when $\omega = 2 + o(1)$, improving upon the previous best $\widetilde{O}(n^{2+1/6}\text{poly}\log(1/\epsilon))$ runtime in the low-accuracy regime. We achieve this by showing a novel stability result that goes beyond previously known works based on interior point methods (IPMs).

Our work is the first to use acceleration and inverse maintenance together efficiently, finally making the two most important building blocks of modern structured convex optimization compatible.

2012 ACM Subject Classification Theory of computation → Continuous optimization; Theory of computation → Data structures design and analysis; Theory of computation → Sketching and sampling

Keywords and phrases Regression, Inverse Maintenance, Multiplicative Weights Update

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.5

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/pdf/2409.20030`

Funding *Deeksha Adil*: Supported by Dr. Max Rössler, the Walter Haefner Foundation and the ETH Zürich Foundation.
Shunhua Jiang: Supported by NSF grants (CCF-1617955, CCF-1740833, CCF-2008733), NSF CAREER award CCF-1844887, and by Google PhD fellowship.
Rasmus Kyng: The research leading to these results has received funding from grant no. 200021 204787 and from the starting grant “A New Paradigm for Flow and Cut Algorithms” (no. TMSGI2_218022) of the Swiss National Science Foundation.

1 Introduction

In this paper, we study the ℓ_∞-regression problem. Given $\epsilon > 0$, a matrix $\boldsymbol{C} \in \mathbb{R}^{n\times d}$ and vector $\boldsymbol{d} \in \mathbb{R}^n$, $d \le n$, we want to find $\widetilde{\boldsymbol{x}} \in \mathbb{R}^d$ such that,

$$\|\boldsymbol{C}\widetilde{\boldsymbol{x}} - \boldsymbol{d}\|_\infty \le (1+\epsilon) \min_{\boldsymbol{x}\in\mathbb{R}^d} \|\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d}\|_\infty. \quad (1)$$

Some of the popular approaches to obtaining fast algorithms for ℓ_∞-regression include using multiplicative weight update (MWU) routines [7, 4, 11, 10, 2, 13, 1], gradient descent [31, 18] and other ways to optimize a softmax function [9, 32, 1], and using interior point methods

© Deeksha Adil, Shunhua Jiang, and Rasmus Kyng;
licensed under Creative Commons License CC-BY 4.0

(IPM) [17, 30, 29]. Interior point methods can find a high-accuracy solution, i.e., an ϵ-approximate solution in $\widetilde{O}(\sqrt{n}\log(1/\epsilon))$[1] linear system solves, whereas most of the other methods are low accuracy solvers, i.e., their running time scales as $\mathrm{poly}(1/\epsilon)$. Naively using gradient descent or MWU requires $O(\sqrt{n}\cdot\mathrm{poly}(1/\epsilon))$ linear system solves. Multiplicative weight update based approaches can be accelerated via a technique called *width reduction* to converge in $O(n^{1/3}\cdot\mathrm{poly}(1/\epsilon))$ linear system solves [11, 10, 2, 13, 1, 3]. Several acceleration techniques have also been developed to improve the iteration complexity of other low-accuracy regression algorithms [26, 8, 9, 32, 1].

To get an overall fast runtime, apart from improving the iteration complexity, a useful approach is to reduce the per-iteration cost. This can be done using *inverse maintenance*, which reduces the cost via *lazy-update* schemes. Notions of inverse maintenance appear in the very first interior point methods, [17, 28], but the modern form was introduced by Vaidya [33]. There have been many important developments in inverse maintenance algorithms since then, and state-of-the-art algorithms use both linear algebraic data structures and dimensionality reduction routines, such as sketching [6]. The improvements in runtimes of interior point methods including the state-of-the-art algorithms depend heavily on these developments in inverse maintenance routines [20, 12, 5, 16, 22].

1.1 Our Results

For simplicity, in the discussion of our results and prior work on this problem, we focus on the case $\omega = 2 + o(1)$ – but our full technical theorems give results for all ω. In the low-accuracy regime of $\varepsilon = 1/\mathrm{polylog(n)}$ the state-of-the-art running time for ℓ_∞-regression is $\widetilde{O}(n^{2+1/18})$, obtained via the randomized algorithm of [16], and $\widetilde{O}(n^{2+1/6})$ for deterministic algorithms via [5]. Both these algorithms in fact obtain high-accuracy solutions, and they use inverse maintenance, but no acceleration. In this work, we push the running time further in the low-accuracy regime by combining the state-of-the-art inverse maintenance techniques of these results with new multiplicative weight methods which allow us to perform acceleration, yielding running times of $\widetilde{O}(n^{2+1/22.5}\,\mathrm{poly}(\epsilon^{-1}))$ with randomization and $\widetilde{O}(n^{2+1/12}\,\mathrm{poly}(\epsilon^{-1}))$ without.

Our first result is a deterministic algorithm that combines acceleration and lazy inverse updates in a novel, more sophisticated way, and achieves a running time of $\widetilde{O}(n^{2+1/12}\,\mathrm{poly}(\epsilon^{-1}))$. This improves on deterministic state-of-the-art $\widetilde{O}(n^{2+1/6}\log(\epsilon^{-1}))$ [5] in the low-accuracy regime. The key to this result is a new notion of ℓ_3-*stability* which is tailored to the accelerated MWU.

▶ **Theorem 1** (Deterministic algorithm). *There is a deterministic algorithm that solves Problem* (1) *in* $\widetilde{O}(n^{2+1/12}\,\mathrm{poly}(\epsilon^{-1}))$ *time when* $\omega = 2 + o(1)$. *This algorithm converges in* $\widetilde{O}\left(n^{1/3}\,\mathrm{poly}(\epsilon^{-1})\right)$ *iterations.*

Our main result is our randomized algorithm with running time $\widetilde{O}(n^{2+1/22.5}\,\mathrm{poly}(\epsilon^{-1}))$.

▶ **Theorem 2** (Randomized algorithm). *There is a randomized algorithm that solves Problem* (1) *in* $\widetilde{O}(n^{2+1/22.5}\,\mathrm{poly}(\epsilon^{-1}))$ *time when* $\omega = 2 + o(1)$. *This algorithm converges in* $\widetilde{O}\left(n^{1/2.5}\,\mathrm{poly}(\epsilon^{-1})\right)$ *iterations.*

To obtain this result, we introduce the first MWU which can combine all three key techniques for ℓ_∞-regression: (a) acceleration, (b) lazy inverse updates, and (c) sketching.

[1] We use $\widetilde{O}(\cdot)$ to hide $\mathrm{poly}\log n$ factors, and we use $\widetilde{O}_\epsilon(\cdot)$ to additionally hide $\mathrm{poly}(\epsilon^{-1})$ factors.

Thus, we give the optimization approach method which is able to efficiently combine these three key techniques of structured convex optimization. This is likely an essential building block toward $n^{2+o(1)}$ optimization for many objectives. If, some day, acceleration is achieved for linear programming, an equivalent integration will be necessary for optimal algorithms in this context. Before describing our new approach, we first review existing techniques for fast ℓ_∞-regression.

1.2 Background: The Ingredients of Fast ℓ_∞-Regression Methods

Both MWUs and IPMs that solve ℓ_∞-regression methods rely on a sequence of calls to ℓ_2-oracles, i.e. a subroutine that solves an ℓ_2-minimization problem, or equivalently, solves a linear equation. In order to solve the ℓ_∞-regression problem (1), a standard MWU approach repeatedly solves a sequence of ℓ_2-oracle problems of the form

$$\boldsymbol{x}^{(i)} = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} \sum_e r_e^{(i)} (\boldsymbol{C}\boldsymbol{x} - \boldsymbol{d})_e^2 \tag{2}$$

where the weights $\{r_e^{(i)}\}$ are chosen by the MWU depending on the magnitude of previous iterates.

Inverse maintenance via stability and robustness

The ℓ_2-oracles of MWUs and IPMs can be implemented by applying the inverse of a matrix, and inverse maintenance can be used to solve the sequence of ℓ_2-oracle calls faster than simply performing a full matrix inversion or linear equation solve on each call. Two key phenomena drive inverse maintenance: *stability* and *robustness*. *Stability* is the property that the inputs to the ℓ_2-oracle only change slowly. In the MWU case, this means the weights $\{r_e^{(i)}\}$ change slowly. We say an optimizer is *robust* if it can make progress using answers from ℓ_2-oracles with somewhat inaccurate inputs. The combination of stability and robustness is especially powerful. Together, these properties ensure that we can delay making small coordinate updates to inputs until they build up to a large cumulative update, and that we only get few large cumulative updates, enabling the use of coordinate-sparse update techniques. This approach of batching together small updates is known as *lazy* inverse updating. Obtaining further speed-ups using sketching also crucially relies on robustness. Because of robustness, we can afford to use sketching to estimate $\boldsymbol{x}^{(i)}$, as long as our estimates allow sufficiently accurate updates to the weights $\{r_e^{(i)}\}$.

The IPM of [12] first achieved a running time of $\widetilde{O}(n^{2+1/6} + n^\omega)$ by introducing a method with excellent stability and robustness, which in turn allowed them to implement a powerful inverse maintenance approach using lazy updates and sketching. Later, [5] showed that the same running time can be obtained deterministically using only lazy updates, and finally [16] gave an improved running time of $\widetilde{O}(n^{2+1/18} + n^\omega)$ using both lazy updates and sketching. The approach of [16] can be thought of as a two-level inverse maintenance, and the use of the randomized sketching techniques is crucial for them to efficiently implement the *query* operation of this data structure. It remains open if there exists any deterministic IPM that can run faster than $\widetilde{O}(n^{2+1/6} + n^\omega)$.

Acceleration via width-reduction

In oracle-based optimization, there is a long history of developing *accelerated* methods, which reduce the iteration count compared to more basic approaches. This can be traced back to accelerated solvers for quadratic objectives [19, 15] and first-order acceleration for gradient

Lipschitz functions ([27] and earlier works by Nemirovski). Christiano et al. [11] developed an acceleration method for multiplicative weight methods that reduces the iteration count for solving ℓ_∞ regression with ℓ_2-oracles from $\widetilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ to $\widetilde{O}(n^{1/3} \cdot \text{poly}(1/\epsilon))$. An alternative approach to acceleration for ℓ_∞-regression can be obtained via the methods of Monteiro and Svaiter [26], and has also been a major research topic, but is beyond the scope of our discussion. For simplicity of our remaining discussion, we ignore ϵ dependencies. A rough outline of the MWU acceleration approach of [11] is as follows: The MWU solves a sequence of ℓ_2-oracle problems returning iterates $\boldsymbol{x}^{(i)}$. If we scale the problem so that $\|\boldsymbol{C}\boldsymbol{x}^\star - \boldsymbol{d}\|_\infty \leq 1$, then weights ensure that (a) in each iteration, $\|\boldsymbol{C}\boldsymbol{x}^{(i)} - \boldsymbol{d}\|_\infty \lesssim \sqrt{n}$ and (b) after $T = \widetilde{O}(\sqrt{n})$ iterations, $\widetilde{\boldsymbol{x}} = \frac{1}{T}\sum_i \boldsymbol{x}^{(i)}$ has $\|\boldsymbol{C}\widetilde{\boldsymbol{x}} - \boldsymbol{d}\|_\infty \leq 1 + \epsilon$. [11] made an important modification: if in some iteration we have $\|\boldsymbol{C}\boldsymbol{x}^{(i)} - \boldsymbol{d}\|_\infty \geq \rho \approx n^{1/3}$, then instead of using $\boldsymbol{x}^{(i)}$, we will adjust the weights $\{r_e^{(i)}\}$ in order to reduce the value of $\|\boldsymbol{C}\boldsymbol{x}^{(i')} - \boldsymbol{d}\|_\infty$ for future iterates $\boldsymbol{x}^{(i')}$. Using this method, an approximately optimal $\widetilde{\boldsymbol{x}} = \frac{1}{T}\sum_i \boldsymbol{x}^{(i)}$ can be found in $T = \widetilde{O}(n^{1/3})$ iterations. The parameter ρ measures the ℓ_∞-norm $\|\boldsymbol{C}\boldsymbol{x}^{(i)} - \boldsymbol{d}\|_\infty$ of each iterate, sometimes known as the *width*, and the weight-adjustment steps of Christiano et al. are hence known as *width reduction steps*. When the oracle width can be reduced in this way, we will say our method is *width-reducible*. This acceleration has never been developed for ℓ_∞-regression in the high-accuracy regime (i.e. running times that scale as polylog$(1/\epsilon)$), and whether this is possible is one of the major open questions in convex optimization.

Weight monotonicity in MWUs: an obstacle to sketching

Many MWU methods are designed to have an important property, which we call *weight monotonicity*. Concretely, in [11] and many other MWUs, the oracle weights $\{\boldsymbol{r}_e^{(i)}\}$ are only growing. This often simplifies analyses greatly, and helps establish other properties including stability, robustness, and width-reducibility. Referring back to our oracle queries introduced above in (2), let us define $\widetilde{\boldsymbol{x}}^{(i)} = \frac{1}{T}\sum_{j\leq i} \boldsymbol{x}^{(j)}$. Weight monotonicity arises because we choose the weights based on an overestimate of $|(\boldsymbol{C}\widetilde{\boldsymbol{x}}^{(i)} - \boldsymbol{d})_e|$ given by $\gamma_i = \frac{1}{T}\sum_{j\leq i} |(\boldsymbol{C}\boldsymbol{x}^{(j)} - \boldsymbol{d})_e|$. In particular, choosing $\boldsymbol{r}_e^{(i)} = \exp(\alpha\gamma_i)$ for some scaling factor α will ensure the weights only grow. As we will discuss later, *weight monotonicity* seems inherently incompatible with sketching, and thus we will need to develop a non-monotone MWU. Prior work by Madry [24, 25] introduced non-monotone weights in a highly specialized IPM for unit-capacity maximum flow. This IPM of Madry has MWU-like properties and allows for some acceleration. The method has other drawbacks including low stability and robustness, but nonetheless inspired some of our design choices.

Prior inverse maintenance with acceleration

We are aware of a single prior work which combined lazy inverse updates with an accelerated MWU to obtain a running time of $\widetilde{O}(n^{2+1/3} + n^\omega)$ for ℓ_p-regression [2]. This approach is relatively naive, falling short of the $\widetilde{O}(n^{2+1/6} + n^\omega)$ running time which can be achieved using only lazy inverse updates.

1.3 Discussion of Techniques

The crucial algorithmic techniques we rely on for speeding up ℓ_∞-regression are (a) acceleration, (b) lazy inverse updates, and (c) sketching. We can view each of these techniques as being enabled by different properties of the overall optimization approach. Our approach to acceleration is enabled by *width-reducibility*, while lazy updates require *stability* and

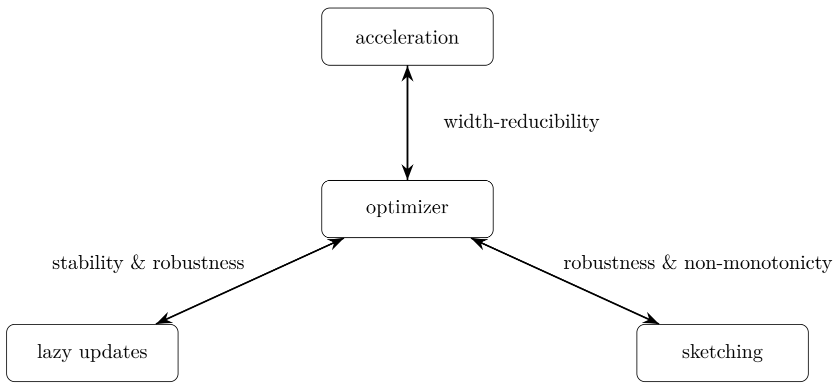

Figure 1 Algorithmic techniques and their requirements on our optimizer.

robustness, and finally sketching requires *robustness* and *non-monotonicity*. This means we need to develop an MWU which simultaneously exhibits all these properties, i.e. it must be stable, robust, non-monotone, and width-reducible. In Figure 1, we summarize how our algorithmic techniques impose different requirements on our optimization approach. Again, for simplicity, in the remaining discussion of our results and prior work on this problem, we focus on the case $\omega = 2 + o(1)$.

We first discuss how to combine stability, robustness, and width-reducibility in a *monotone* MWU, which leads to a comparatively simple, deterministic algorithm using acceleration and lazy inverse updates, but no sketching.

Stability and robustness of a monotone, width-reducible MWU

[2] showed how to obtain stability, robustness, and width-reducibility together, with a monotone MWU. However, this work only established a weak notion of stability and hence comparatively slow running time of $\widetilde{O}(n^{2+1/3})$. In contrast, one can show that by directly using stability and robustness in a monotone accelerated MWU, one can adapt the data structure approach of [5] to achieve a running time of $\widetilde{O}(n^{2+1/9})$, yielding a faster MWU.

Our first result Theorem 1 is based on the observation that monotone MWU also enables a new, stronger notion of stability, which we call ℓ_3-stability. This allows us to further reduce the number of lazy updates we make and lets us achieve a deterministic running time of $\widetilde{O}(n^{2+1/12})$.

Non-monotone MWU - a key ingredient for sketching

As we described above, it is relatively easy to improve the running time of low-accuracy ℓ_∞-regression among deterministic algorithms, by designing a monotone, robust, width-reducible MWU with a novel ℓ_3-stability.

To further accelerate the algorithm by using a two-level inverse maintenance data structure, we need to use randomized sketching techniques to efficiently implement the query operation, which is required in every iteration of the MWU algorithm. Unfortunately, weight monotonicity is in conflict with sketching, because monotonicity arises from ignoring cancellations in $(\boldsymbol{C}\widetilde{\boldsymbol{x}}^{(i)} - \boldsymbol{d})_e$ between different iterations.[2] In contrast, when using sketching, we want

[2] Recall that the final output of our MWU is the last averaged iterate $\widetilde{\boldsymbol{x}}^{(T)}$.

to crucially rely on cancellation between different iterations, as we sometimes overestimate $(\boldsymbol{C}\boldsymbol{x}^{(i)} - \boldsymbol{d})_e$ and sometimes underestimate it, but get it right on average. Because of this, we design an MWU with non-monotone weights. This in turn makes width-reducibility, robustness, and stability much harder to obtain.

To allow us to work with non-monotone weights and still obtain acceleration, we introduce a more delicate width-reduction scheme, inspired by [25]. We also provide a tighter analysis of the sketching technique (it was named coordinate-wise embedding by [21, 16]) that upper bounds its total noise across different iterations using martingale concentration inequalities. This tighter analysis is necessary to control the overall error introduced by the sketching technique in our MWU algorithm. We believe this tighter analysis could also provide a simpler analysis for the IPM results of [12, 16].

This new width-reduction approach in turn also requires us to estimate an ℓ_3-norm associated with each iterate $\boldsymbol{x}^{(i)}$, and to do this quickly, we need to employ new sketching tools. To implement this approach, we also need an additional heavy-hitter sketch that allows us to identify which weights to adjust during width reduction.

Stability and robustness of a non-monotone, width-reducible MWU

Stability and robustness are crucial when we want to use lazy updates and sketching for inverse maintenance. Standard techniques for acceleration by width-reduction are unstable in the context of *non-monotone* MWU. Thus, to combine stability, width-reduction, and non-monotonicity, we have to further change our width-reduction strategy.

A central challenge is that width-reducibility is inherently in tension with the other properties. To simultaneously achieve stability and width-reducibility, we introduce a new and rather different approach to width-reduction, which we call *stable width-reduction.* This approach is more conservative than existing methods, and uses smaller width-reduction steps to achieve stability.

Combining width-reducibility with robustness is also difficult. Width-reduction relies on identifying too-large entries of the oracle outputs and making adjustments to the corresponding weights. But, robustness requires us to operate with inaccurate weights. We want to allow for weights that are inaccurate up to a factor $(1 \pm 1/\operatorname{polylog}(n))$, and this is enough to completely change which oracle outputs are too large. In fact, we do not achieve general robust, but instead show that our method is robustness to (1) the errors induced by our specific lazy update scheme and (2) the errors induced by sketching.

Future perspectives

It remains open to design any algorithm for low-accuracy ℓ_∞ regression beyond $\widetilde{O}(n^{2+1/22.5})$ when $\omega = 2 + o(1)$. We remark that if it were possible to use ℓ_3-stability with the two-level data structure and an algorithm that converges in $n^{1/3}$ iterations, then we would achieve a runtime of $\widetilde{O}(n^{2+1/48})$. However, the current techniques for inverse maintenance and acceleration are not sufficient to achieve $\widetilde{O}(n^{2+o(1)} + n^{\omega})$, which we believe would require substantially new techniques. On the other hand, even obtaining slight improvements in the runtime would require more robust acceleration and inverse maintenance frameworks which would be of independent interest.

In this paper, we analyze our algorithms in the RealRAM model. Establishing a similar analysis in finite precision arithmetic is an interesting open problem. Inverse maintenance-based IPM with finite precision arithmetic was studied by [14].

We have demonstrated that acceleration techniques for MWU can be efficiently combined with inverse maintenance methods. For linear programming, no similar acceleration techniques exist and it is a major open problem to design these or rule out the possibility in various computational models. If acceleration can be achieved for linear programming, deploying it in conjunction with inverse maintenance will likely require techniques similar to those we introduce in this work.

2 Technical overview

2.1 Deterministic MWU Algorithm via One-Level Inverse Maintenance

MWU methods reduce ℓ_∞-regression problems to a sequence of ℓ_2-minimization problems, which can be solved by solving systems of linear equations – or equivalently, applying the inverse of some matrix. More concretely, an MWU for finding approximate solutions to $\min_{\boldsymbol{x}\in\mathbb{R}^d} \|\boldsymbol{C}\boldsymbol{x}-\boldsymbol{d}\|_\infty$ requires us to repeatedly solve problems of the form

$$\min_{\Delta\in\mathbb{R}^d} \sum_e \boldsymbol{r}_e^{(i)}(\boldsymbol{C}\Delta-\boldsymbol{d})_e^2$$

across iterations $i=1,\dots,T$. The exact solution to these minimization problems is given by

$$\Delta^{(i)} = (\boldsymbol{C}^\top \boldsymbol{R}^{(i)}\boldsymbol{C})^{-1}\boldsymbol{C}^\top \boldsymbol{R}^{(i)}\boldsymbol{d}.$$

The multiplicative weight update method iteratively updates the weights using $\Delta^{(i)}$ and "penalizes" the coordinates e that have large $|\boldsymbol{C}\Delta^{(i)}-\boldsymbol{d}|_e$ by increasing their weights $\boldsymbol{r}_e^{(i+1)}$ in the next iteration. In the end the method outputs $\boldsymbol{x}=\sum_{i=1}^T \Delta^{(i)}/T$ as the approximate ℓ_∞ minimizer.

The cost of each iteration is dominated by the time required to solve the corresponding system of linear equations for $\Delta^{(i)}$ – or equivalently, applying the inverse of some matrix. If solving this sequence of systems of linear equations can be done faster than naively solving each system separately, then we can speed up the cost per iteration of the MWU algorithm, and hence make the algorithm faster. A similar problem of solving a sequence of systems of linear equations was studied for the IPM algorithms [12, 5, 16], and they achieved speed-ups by using lazy updates with *inverse maintenance* data structures. They could use lazy updates because the IPM algorithm satisfies a stability guarantee and a robustness guarantee. More precisely, (1) IPMs satisfy an ℓ_2-stability guarantee that the ℓ_2-norm of the relative changes between two iterations is bounded, i.e., $\|\frac{\boldsymbol{r}^{(i+1)}-\boldsymbol{r}^{(i)}}{\boldsymbol{r}^{(i)}}\|_2^2 \le O(1)$. (2) IPMs are still correct if the system of linear equations is solved with coordinate-wise approximate weights $\overline{\boldsymbol{r}} \approx_\delta \boldsymbol{r}$ for some $\delta>0$.

As it turns out, the *monotone* MWU algorithm is also inherently stable and robust, even with acceleration. We can therefore use coordinate-wise approximate weights $\overline{\boldsymbol{r}}^{(i)} \approx_\delta \boldsymbol{r}^{(i)}$ in each iteration, and only update $\overline{\boldsymbol{r}}_e^{(i)}$ when it differs from $\boldsymbol{r}_e^{(i)}$ by more than δ. This ensures that the approximate weights $\overline{\boldsymbol{r}}^{(i)}$ undergoes low-rank updates. We present a robust version of the known *accelerated* multiplicative weights update method for ℓ_∞-regression from [11, 10] below, where when solving the system of linear equations for $\Delta^{(i)}$ we use the approximate weights $\overline{\boldsymbol{r}}^{(i)}$.

▶ **Theorem 3** ([10]). *Let $0<\epsilon<1/2$ and $0\le\delta\le\epsilon/6$. Algorithm 1 returns $\widehat{\boldsymbol{x}}$ such that $\|\boldsymbol{C}\widehat{\boldsymbol{x}}-\boldsymbol{d}\|_\infty \le 1+O(\epsilon)$ in $\widetilde{O}(n^{1/3}\epsilon^{-7/3})$ iterations. Each iteration solves a linear system as specified in Line 9 of the algorithm.*

Algorithm 1 Monotone Width Reduced MWU Algorithm.

```
1: procedure MWU-SOLVER($\epsilon, \boldsymbol{C}, \boldsymbol{d}$)
2:     $\boldsymbol{w}^{(0,0)} \leftarrow 1_n, \quad \boldsymbol{x}^{(0)} \leftarrow 0_d$
3:     $\tau \leftarrow \Theta\left(\frac{n^{\frac{1}{3}}}{\epsilon^{\frac{1}{3}}} \log \frac{n}{\Psi_0}\right), \alpha \leftarrow \Theta\left(n^{-\frac{1}{2}+\eta}\epsilon^{\frac{1}{3}}\left(\log \frac{n}{\Psi_0}\right)^{-1}\right), \eta \leftarrow \frac{1}{6}$
4:     $T \leftarrow \alpha^{-1}\epsilon^{-2}\log n$
5:     $i \leftarrow 0, k \leftarrow 0$
6:     while $i < T$ do
7:         $\boldsymbol{r}_e^{(i,k)} \leftarrow \boldsymbol{w}_e^{(i,k)} + \frac{\epsilon}{n}\|\boldsymbol{w}^{(i,k)}\|_1$
8:         $\overline{\boldsymbol{r}}^{(i,k)} \leftarrow$ SELECTVECTOR($\boldsymbol{r}^{(i,k)}, i+k, \delta$)        ▷ $\overline{\boldsymbol{r}} \approx_\delta \boldsymbol{r}$
9:         $\Delta^{(i,k)} \leftarrow \arg\min_{\Delta \in \mathbb{R}^d} \sum_e \overline{\boldsymbol{r}}_e^{(i,k)}(\boldsymbol{C}\Delta - \boldsymbol{d})_e^2$        ▷ $\Delta = (\boldsymbol{C}^\top \overline{\boldsymbol{R}}^{(i,k)} \boldsymbol{C})^{-1}\boldsymbol{C}^\top \overline{\boldsymbol{R}}^{(i,k)} \boldsymbol{d}$
10:        if $\left\|\boldsymbol{C}\Delta^{(i,k)} - \boldsymbol{d}\right\|_\infty \leq \tau$ then        ▷ primal step
11:            $\boldsymbol{w}^{(i+1,k)} \leftarrow \boldsymbol{w}^{(i,k)}(1 + \epsilon\alpha|\boldsymbol{C}\Delta^{(i,k)} - \boldsymbol{d}|)$
12:            $\boldsymbol{x}^{(i+1)} \leftarrow \boldsymbol{x}^{(i)} + \Delta^{(i,k)}$
13:            $i \leftarrow i + 1$
14:        else
15:            For all coordinates $e$ with $|\boldsymbol{C}\Delta^{(i,k)} - \boldsymbol{d}|_e \geq \tau$        ▷ width reduction step
16:                $\boldsymbol{w}_e^{(i,k+1)} \leftarrow (1+\epsilon)\boldsymbol{w}_e^{(i,k)} + \frac{\epsilon^2}{n}\|\boldsymbol{w}^{(i,k)}\|_1$
17:                $k \leftarrow k + 1$
18:    return $\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x}^{(T)}}{T}$
```

In fact, we can prove that this algorithm satisfies an even stronger stability guarantee – a quantitatively strong type of ℓ_3-stability, namely

$$\sum_{i=1}^{T} \left\| \frac{\boldsymbol{r}^{(i+1)} - \boldsymbol{r}^{(i)}}{\boldsymbol{r}^{(i)}} \right\|_3^3 \leq O(n^{1/3}).$$

The ℓ_3-stability guarantee allows for the following lazy-update scheme: for every ℓ, in every 2^ℓ iterations perform an update of size $O(2^{3\ell})$ to $\overline{\boldsymbol{r}}^{(i)}$.

Together with the one-level inverse maintenance of [6], this improves upon the previous best deterministic algorithm for low-accuracy ℓ_∞ regression that runs in $O(n^\omega + n^{2+1/6})$. We present a simplified version of the data structure below, and the formal version tailored to our application can be found in the full version.

▶ **Lemma 4** (One-level inverse maintenance, (Informal) Theorem 4.1 of [6])**.** *There is a data structure that supports the following two operations to maintain the inverse of an $n \times n$ matrix M:*

- **Reset***: Reset M^{-1} to $(M + \Delta)^{-1}$, where Δ has k_0 non-zero entries. This operation can be done in $O(\mathcal{T}_{\text{mat}}(n, n, k_0))$ time.*[3]
- **Query***: Output the vector $(M + \Delta)^{-1} \cdot v$ using the maintained M^{-1} and $M^{-1}v$, where Δ has at most n^{a_0} non-zero entries. This operation can be done in $O(n^{\omega a_0} + n^{1+a_0})$ time.*

Runtime when $\omega = 2$

For simplicity, we only show the runtime of our algorithm when $\omega = 2$ in this section and omit polylogarithmic factors. Let us choose the parameter $a_0 = 3/4$, so that we perform a reset

[3] $\mathcal{T}_{\text{mat}}(n, r, m)$ denotes the time complexity of multiplying an $n \times r$ matrix with an $r \times m$ matrix.

operation whenever we accumulate more than $n^{a_0} = n^{3/4}$ updates to $\overline{\boldsymbol{r}}$. From our low-rank update scheme under the ℓ_3 stability guarantee, this only happens in every $n^{1/4}$ iterations. So we perform a reset operation with cost $O(n^2)$ (since $\omega = 2$) in every $O(n^{1/4})$ iterations, and over the total $O(n^{1/3})$ iterations this gives a total reset time of $O(n^{2-1/4} \cdot n^{1/3}) = O(n^{2+1/12})$.

We perform a query operation in every iteration with cost $O(n^{2a_0} + n^{1+a_0}) = O(n^{1+3/4})$. Over all $O(n^{1/3})$ iterations this gives a total query time of $O(n^{1+3/4} \cdot n^{1/3}) = O(n^{2+1/12})$. Therefore, the total runtime is the sum of the reset time and the query time, which is $O(n^{2+1/12})$ as claimed in Theorem 1.

2.2 Randomized MWU Algorithm via Two-Level Inverse Maintenance

To further improve the runtime of the algorithm, we will use the following, more efficient two-level inverse maintenance data structure.

▶ **Lemma 5** (Two-level inverse maintenance, (Informal) Theorem 4.2 of [6]). *There is a data structure that supports the following three operations to explicitly maintain the inverse of an $n \times n$ matrix M. The algorithm achieves the goal via explicitly maintaining the inverse of an $n \times n$ matrix M_0 and implicitly maintaining the inverse of another $n \times n$ matrix M_1 that differs from M_0 on at most n^{a_0} entries, and the true matrix M always differ from M_1 on at most n^{a_1} entries where $a_1 \leq a_0$:*

- **Reset***: Reset M_0^{-1} to $(M_0 + \Delta_0)^{-1}$, where Δ_0 has k_0 non-zero entries. This operation can be done in $\mathcal{T}_{\text{mat}}(n, n, k_0)$ time.*
- **Partial reset***: Implicitly reset M_1^{-1} to $(M_1 + \Delta_1)^{-1}$, where Δ_1 has k_1 non-zero entries. This operation can be done in $\mathcal{T}_{\text{mat}}(n, n^{a_0}, k_1)$ time.*
- **Query***: Output ℓ entries of the vector $M^{-1} \cdot v$ using the maintained M_0^{-1}, M_1^{-1} (implicitly). This operation can be done in $\mathcal{T}_{\text{mat}}(n^{a_0}, n^{a_1}, \max\{n^{a_1}, \ell\})$ time.*

The total runtime of the above data structure is the sum of its reset, partial reset, and query times. Let us now compare the query times of this two-level data structure with the one-level version. Observe that, the query time of the one-level data structure is n^{1+a_0} and that of the two-level data structure is better than n^{1+a_0} only if $\ell = o(n)$. In other words, we get an improvement via the two-level data structure only if we have an algorithm that does not require querying the entire maintained vector $M^{-1}v$.

So far, such an improvement via the two-level data structure has only been utilized, although in a complicated way, in the work of [16] where they give a fast algorithm for linear programming by using the data structure within the robust interior point method framework and querying a *sketch* of the vector at every iteration. It is still an open problem if one can achieve their runtime of $\approx n^{2+1/18}$ via a deterministic algorithm and it is conjectured that improving the runtime either requires an improved data structure or, a more sophisticated "dimension reduction technique" to work with the algorithm.

Sketching and non-monotone MWU

Similar to [16], in our work we also query a sketch of the maintained vector in every iteration. More precisely, in each iteration we use a random matrix $\boldsymbol{S} \in \mathbb{R}^{n^{1/2+\eta} \times n}$ where η is the acceleration that we get, i.e., the total number of iterations is $O(n^{1/2-\eta})$, and we compute an approximate step $\boldsymbol{S}^\top \cdot \boldsymbol{S} \cdot (\boldsymbol{C}^\top \Delta^{(i,k)} - \boldsymbol{d})$. Using the coordinate-wise embedding guarantee of the random matrix $\boldsymbol{S}$, we can ensure that for each coordinate we have

$$\left(\boldsymbol{S}^\top \boldsymbol{S}(\boldsymbol{C}^\top \Delta^{(i,k)} - \boldsymbol{d})\right)_e \approx (\boldsymbol{C}^\top \Delta^{(i,k)} - \boldsymbol{d})_e.$$

We now require to change Line 11 of Algorithm 1 to update the weights by

$$\boldsymbol{w}^{(i+1,k)} \leftarrow \boldsymbol{w}^{(i,k)}\Big(1 + \epsilon\alpha \cdot \boldsymbol{S}^\top \boldsymbol{S}(\boldsymbol{C}\Delta^{(i,k)} - \boldsymbol{d})\Big).$$

Note that we lose monotonicity of the weights with this new primal step. We have to use this non-monotone update because the absolute values $|\boldsymbol{S}^\top \boldsymbol{S}(\boldsymbol{C}^\top\Delta^{(i,k)} - \boldsymbol{d})|$ would result in an error that is around the standard deviation of the estimator in every update of $\boldsymbol{w}^{(i,k)}$'s, and this would add up over iterates. Since the entire analysis of the MWU methods depends on tracking potentials which are functions of the weights, we would incur a large error. To circumvent this issue we require a version of the MWU method where the weights are not updated monotonically, and the random noise introduced by the sketching matrix $\boldsymbol{S}$ can cancel out with each other across different coordinates e and across different iterations i.

Monotonicity is crucial in accelerating MWU methods and it is non-trivial to achieve accelerated rates without it. A few works in graph algorithms have been successful in obtaining accelerated rates without monotonicity [25, 23] for specific algorithms. In this paper, we extend the algorithm of [25] to regression and obtain an algorithm with non-monotone updates that also converges in $n^{1/3}$ iterations and is robust (please refer to the full version for the complete algorithm and analysis).

Interior point methods directly control the solution quality of the last iterate. In contrast, MWU algorithms only measure the quality of the average of the primal iterates $\Delta^{(i,k)}$. As a result, our bound on the final solution requires a new MWU analysis that can handle cancellations between iterates of the errors arising from using sketching. We achieve this by developing a tighter analysis that upper bounds the sum of the sketching error over multiple iterations:

$$\sum_{i=0}^{t}\left(\Big(\boldsymbol{S}^\top \boldsymbol{S}(\boldsymbol{C}\Delta^{(i)} - \boldsymbol{d})\Big)_e - (\boldsymbol{C}\Delta^{(i)} - \boldsymbol{d})_e\right) \lesssim \frac{\sqrt{nt}}{\sqrt{b}}.$$

We prove this bound using Freedman's concentration bound for martingales. We also believe this tighter analysis can simplify the sketching analysis for the previous IPM papers [12, 21, 16].

Stability and robustness of non-monotone MWU

The non-monotone MWU with standard width reduction steps is neither stable nor satisfies a low-rank update per iteration. We propose a new width reduction step that satisfies a low-rank update scheme which is sufficient for our data structure. Our steps, however, do not satisfy ℓ_2 stability, which is a sufficient condition for the low-rank update scheme. Instead of increasing all weights by a factor of $(1+\epsilon)$ as in Line 16 of Algorithm 1, our new width reduction step increases a carefully selected set of weights. As a result, we can ensure that whenever we increase a large set of weights, we also increase the potential by a lot, so this event doesn't happen very often. This helps ensure that weight updates from width-reduction steps occur on a similar "schedule" to weight updates from our primal update steps, and it allows us to efficiently handle both in the inverse maintenance data structure (please refer to Algorithm 3 for the complete algorithm and refer to the full version for the full analysis). To efficiently find the coordinates e to perform width reduction on, we use an additional *heavy-hitter* data structure to identify these Δ_e exactly. We can only afford to find $n^{1/2+\eta}$ such coordinates in each iteration. This restriction on the number of coordinates restricts us to set η to be 1/10, and our final iteration complexity is $n^{1/2-\eta} = n^{2/5}$ instead of $n^{1/3}$. The non-monotone algorithm also requires estimating a weighted ℓ_3-norm of $\widehat{\Delta}^{(i,k)}$'s for which we use an additional sketch from [34].

Unlike the width reduction steps, the primal steps are stable, and they satisfy the ℓ_2 stability,

$$\left\|\frac{\boldsymbol{r}^{(i+1)}-\boldsymbol{r}^{(i)}}{\boldsymbol{r}^{(i)}}\right\|_2^2 \leq O(n^{2\eta}).$$

Given the ℓ_2 stability guarantee, we again use coordinate-wise approximate weights $\overline{\boldsymbol{r}}^{(i)} \approx_\delta \boldsymbol{r}^{(i)}$ in each primal step, and only update $\overline{\boldsymbol{r}}_e^{(i)}$ to be $\boldsymbol{r}_e^{(i)}$ if it differs from $\boldsymbol{r}_e^{(i)}$ by more than δ. This again guarantees a low-rank update scheme for the primal steps: for every ℓ, in every 2^ℓ iterations we only perform an update of size $O(2^{2\ell} \cdot n^{2\eta})$ to $\overline{\boldsymbol{r}}^{(i)}$.

It is non-trivial to show that the accelerated non-monotone MWU is robust under such coordinate-wise approximations to the weights. This is because we do not update the weights in every primal step, and we lazily update them in future iterations. We use an amortization argument to show that we can still gain enough changes in the required potentials even when we defer some updates to the future. However, this means our accelerated non-monotone MWU is only robust under the specific approximate weights $\overline{\boldsymbol{r}}_e^{(i)}$ that are updated to be $\boldsymbol{r}_e^{(i)}$ whenever it differs too much from $\boldsymbol{r}_e^{(i)}$. We cannot guarantee robustness if in every iteration we choose an arbitrary coordinate-wise approximation unless we consider the unaccelerated algorithm, which was guaranteed in the IPM algorithms.

Runtime when $\omega = 2$

Finally, we sketch the time complexity of our non-monotone MWU algorithm using sketching when $\omega = 2$. For simplicity, we omit polylogarithmic factors. Using the two-level inverse maintenance data structure of Lemma 5, we perform a reset operation whenever we accumulate more than n^{a_0} updates to $\overline{\boldsymbol{r}}$, and by our low-rank update scheme under the ℓ_2 stability guarantee, this only happens in every $n^{a_0/2-\eta}$ iterations. Similarly, we perform a partial reset operation whenever we accumulate more than n^{a_1} updates to $\overline{\boldsymbol{r}}$, and this only happens in every $n^{a_1/2-\eta}$ iterations. Finally, note that our query time is bounded by $n^{a_0+a_1}$ since we always ensure that we query for at most $\ell = O(n^{1/2+\eta})$ coordinates in each iteration. So our total runtime over $T = n^{1/2-\eta}$ iterations is

$$\underbrace{T \cdot \frac{n^2}{n^{a_0/2-\eta}}}_{\text{reset}} + \underbrace{T \cdot \frac{n^{1+a_0}}{n^{a_1/2-\eta}}}_{\text{partial reset}} + \underbrace{T \cdot n^{a_0+a_1}}_{\text{reset}} = n^{2.5-a_0/2} + n^{1.5+a_0-a_1/2} + n^{0.5-\eta+a_0+a_1}.$$

Choosing the parameters $a_0 = 1 - \frac{1-2\eta}{9}$ and $a_1 = 1 - \frac{1-2\eta}{3}$, we have that the total runtime is bounded by $O(n^{2+1/18-\eta/9})$. Since we achieve an acceleration of $\eta = 1/10$ and $n^{1/2-\eta} = n^{2/5}$ iterations, this gives the claimed $O(n^{2+1/18-\eta/9}) = O(n^{2+1/22.5})$ time complexity of Theorem 2.

3 Fast Width-Reduced MWU Algorithms

In this section, we present the formal guarantees of our multiplicative weight update routines: a deterministic MWU algorithm with monotone weights (Algorithm 1) that is used in Theorem 1, and a randomized MWU algorithm with non-monotone weights and stable and robust steps (Algorithm 3) that is used in Theorem 2.

3.1 Lazy update procedure

We first present the SelectVector algorithm (Algorithm 2) from [22] that computes a coordinate-wise approximate vector $\overline{\boldsymbol{r}}$ of $\boldsymbol{r}$ such that $\overline{\boldsymbol{r}}$ undergoes small updates.

We remark that the only difference between our algorithm and that of [22] is in Line 11 where we only include a coordinate e in S if $\boldsymbol{w}_e$ is not being updated by a width reduction step between primal iterations $i-2^\ell$ and i. This is due to a minor technicality of dealing with the two kinds of steps, primal and width reduction, in Algorithm 3. In all our algorithms, if we toggle a coordinate e in a width reduction step, then we always update the "lazy" approximate vector $\overline{\boldsymbol{r}}_e$ to be the same as $\boldsymbol{r}_e$, so the guarantees of the SelectVector algorithm still hold under this change in Line 11.

Algorithm 2 Compute a coordinate-wise approximate vector that undergoes small updates [22].

```
1: procedure SelectVector($\boldsymbol{r}^{(i)}, i, \delta$)
2:         ▷ This procedure stores all previous $\boldsymbol{r}^{(0)}, \cdots, \boldsymbol{r}^{(i-1)}$, and the $\overline{\boldsymbol{r}}$ in the previous iteration
3:     if $i = 0$ then
4:         return $\overline{\boldsymbol{r}} \leftarrow \boldsymbol{r}^{(0)}$
5:     $S \leftarrow \emptyset$
6:     for $\ell = 0, 1, \cdots, \log n$ do
7:         if $i \equiv 0 \mod 2^\ell$ then
8:             if $\ell = \log n$ then
9:                 $S \leftarrow [n]$
10:            else
11:                $S \leftarrow S \cup \{e : |\ln(\frac{\boldsymbol{r}_e^{(i)}}{\boldsymbol{r}_e^{(i-2^\ell)}})| \geq \frac{\delta}{2\log n}$ and LastWidth$(i, e) \leq i - 2^\ell\}$
12:                    ▷ LastWidth$(i, e) \leq i$ is the last primal step during which a width reduction step updates $\boldsymbol{w}_e$
13:    $\overline{\boldsymbol{r}}_e \leftarrow \boldsymbol{r}_e^{(i)}$ for all $e \in S$
14:    return $\overline{\boldsymbol{r}}$
```

3.2 Monotone Multiplicative Weights Update Algorithm

We have already presented the convergence guarantees of Algorithm 1 in Theorem 3. We now add the stability guarantees that we use to prove the guarantees of our fast deterministic algorithm. The analysis of the algorithm and the stability guarantees can be found in the full version.

▶ **Lemma 6** (Stability bound of ℓ_3 norm over all primal iterations). *Let k_i denote the number of width reduction steps taken by the algorithm when the i^{th} primal step is being executed. Then over all T primal steps of Algorithm 1, we have*

$$\sum_{i=0}^{T-1} \sum_{e \in S_i} \left(\frac{\boldsymbol{r}_e^{(i+1,k_i)} - \boldsymbol{r}_e^{(i,k_i)}}{\boldsymbol{r}_e^{(i,k_i)}} \right)^3 \leq \widetilde{O}(\alpha^2 n) = \widetilde{O}(n^{1/3}\epsilon^{2/3}).$$

Here S_i is the set of coordinates e at primal iteration i such that $\boldsymbol{r}_e^{(i+1,k_i)} \geq \boldsymbol{r}_e^{(i,k_i)}(1+3\epsilon\alpha)$[4].

[4] We note that it is sufficient to consider these sets S_i's since any change that is smaller than the ones captured here can happen only $\widetilde{O}(1)$ times.

▶ **Lemma 7** (Stability bound of ℓ_3 norm over all width reduction iterations)**.** *Let i_k denote the number of primal steps taken before the execution of the k^{th} width reduction step. Then, over all K width reduction steps of Algorithm 1, we have*

$$\sum_{k=0}^{K-1}\left(\frac{\boldsymbol{r}_e^{(i_k,k+1)}-\boldsymbol{r}_e^{(i_k,k)}}{\boldsymbol{r}_e^{(i_k,k)}}\right)^3 \leq \widetilde{O}(n^{1/3}).$$

3.3 Algorithm with Non-Monotone Weights, Stability and Robustness

We now give our main algorithm which can be used with our two-level inverse maintenance data structure. Algorithm 3 updates the weights in a non-monotone way, and additionally has stable primal and width reduction steps. It is also compatible with sketching as required by the data structure. We can prove the following guarantees.

▶ **Theorem 8.** *For $\eta \leq 1/10$, with probability $1-1/n^3$, Algorithm 3 with inputs $\begin{bmatrix}\boldsymbol{C}\\-\boldsymbol{C}\end{bmatrix}$, $\begin{bmatrix}\boldsymbol{d}\\-\boldsymbol{d}\end{bmatrix}$, and ϵ finds $\widehat{\boldsymbol{x}} \in \mathbb{R}^n$ such that $\|\boldsymbol{C}\widehat{\boldsymbol{x}}-\boldsymbol{d}\|_\infty \leq 1+O(\epsilon)$ in at most $\widetilde{O}\left(n^{1/2-\eta}\epsilon^{-4}\right)$ iterations. Furthermore, the algorithm satisfies the following extra guarantees:*

1. *In the width reduction step of the algorithm, the algorithm only requires to find at most $\widetilde{O}\left(n^{1/2+\eta}\right)$ large coordinates per iteration.*
2. *The algorithm satisfies the following low-rank update scheme: There are at most $\frac{T+K}{2^\ell}$ number of iterations where $\overline{\boldsymbol{r}}$ receives an update of rank $\widetilde{O}_\epsilon(n^{1/5}2^{2\ell})$.*

In order to get Algorithm 3 we begin by extending the graph based algorithms of [25] to ℓ_∞-regression. A direct extension does not have stable width reduction steps. We therefore design a new set of width reduction steps which necessitates a new analysis for bounding the number of such steps. We then additionally add sketching to the primal steps to get the final algorithm. The full analysis of the algorithm can be found in the full version.

References

1 Deeksha Adil, Brian Bullins, and Sushant Sachdeva. Unifying width-reduced methods for quasi-self-concordant optimization. *Advances in Neural Information Processing Systems*, 34:19122–19133, 2021. URL: `https://proceedings.neurips.cc/paper/2021/hash/9f9e8cba3700df6a947a8cf91035ab84-Abstract.html`.

2 Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Iterative refinement for ℓ_p-norm regression. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1405–1424. SIAM, 2019.

3 Deeksha Adil, Rasmus Kyng, Richard Peng, and Sushant Sachdeva. Fast algorithms for ℓ_p-regression. *Journal of the ACM*, 71(5):1–45, 2024.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The Multiplicative Weights Update Method: A Meta-Algorithm and Applications. *Theory of Computing*, 8(6):121–164, May 2012. `doi:10.4086/toc.2012.v008a006`.

5 Jan Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020. `doi:10.1137/1.9781611975994.16`.

6 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 456–480. IEEE, 2019. `doi:10.1109/FOCS.2019.00036`.

Algorithm 3 Accelerated MWU algorithm with non-monotone weights and stable and robust steps.

```
procedure MWU-NonMonotoneRobust($\widetilde{C}, \widetilde{d}, \epsilon$)
    $w^{(0,0)} \leftarrow 1_{2n}, \quad \overline{r}^{(0,0)} \leftarrow r^{(0,0)} \leftarrow (1+\epsilon)1_{2n}, \quad x^{(0)} \leftarrow 0_d$
    $\alpha \leftarrow \widetilde{\Theta}(n^{-1/2+\eta}\epsilon)$
    $\tau \leftarrow \widetilde{\Theta}(n^{1/2+\eta}\epsilon^{-4}), \quad \rho \leftarrow \widetilde{\Theta}(n^{1/2-3\eta}\epsilon^{-2})$
    $T \leftarrow \alpha^{-1}\epsilon^{-2}\ln n$
    $i, k = 0$
    $b \leftarrow \widetilde{\Theta}(n^{1/2+\eta}\epsilon^{-2})$
    Let $S^{(0)}, S^{(1)}, \cdots, S^{(T-1)} \in \mathbb{R}^{b\times 2n}$ be random matrices where each entry is $+\frac{1}{\sqrt{b}}$ with probability 1/2 and $-\frac{1}{\sqrt{b}}$ with probability 1/2..
    while $i < T$ do
        $\Delta^{(i,k)} \leftarrow (\widetilde{C}^\top \overline{R}^{(i,k)} \widetilde{C})^{-1} \widetilde{C}^\top \overline{R}^{(i,k)} \widetilde{d}$    ▷ $\Delta^{(i,k)} = \arg\min_\Delta \sum_e \overline{r}_e^{(i,k)} (\widetilde{C}\Delta - \widetilde{d})_e^2$
        $u^{(i,k)} \leftarrow \widetilde{C}\Delta^{(i,k)} - \widetilde{d}$
        $\widehat{u}^{(i,k)} \leftarrow (\overline{R}^{(i,k)})^{-1/2} \cdot (S^{(i)})^\top S^{(i)} \cdot (\overline{R}^{(i,k)})^{1/2} u^{(i,k)}$
        $\Psi(\overline{r}^{(i,k)}) \leftarrow \sum_e \overline{r}_e^{(i,k)} (u_e^{(i,k)})^2$
        if $\sum_e \overline{r}_e^{(i,k)} |u_e^{(i,k)}|^3 \le C_3 \rho \Psi(\overline{r}^{(i,k)})$ then    ▷ primal step
            $w^{(i+1,k)} \leftarrow w^{(i,k)} \left(1 + \epsilon \overrightarrow{\alpha}^{(i,k)} \widehat{u}^{(i,k)}\right)$
            $\overrightarrow{\alpha}_e^{(i,k)} = \begin{cases} \alpha \cdot (1 + \epsilon\alpha \widehat{u}_e^{(i,k)}) & \text{if } \widehat{u}_e^{(i,k)} \ge 0 \\ \alpha / (1 - \epsilon\alpha \widehat{u}_e^{(i,k)}) & \text{else} \end{cases}$
            $r^{(i+1,k)} \leftarrow w^{(i+1,k)} + \frac{\epsilon}{2n} \sum_e w_e^{(i+1,k)}$
            $\overline{r}^{(i+1,k)} \leftarrow$ SelectVector($r^{(i+1,k)}, i+1, \delta$)    ▷ Algorithm 2
            $x^{(i+1)} \leftarrow x^{(i)} + \Delta^{(i,k)}$
            $i \leftarrow i + 1$
        else if $\sum_e \overline{r}_e^{(i,k)} |u_e^{(i,k)}|^3 \ge C_3^{-1} \rho \Psi(\overline{r}^{(i,k)})$ then    ▷ width reduction step
            Let $S$ be the set of coordinates $e$ such that $|u_e^{(i,k)}| \ge \rho/(2C_3)$
            $H \subseteq S$ be maximal subset such that $\sum_{e\in H} \overline{r}_e^{(i,k)} \le \tau^{-1} \Psi(\overline{r}^{(i,k)})$
            if $H \ne S$ then
                Pick any $\bar{e} \in S \setminus H$.
                For all $e \in H \cup \{\bar{e}\}$, $w_e^{(i,k+1)} \leftarrow (1+\epsilon) w_e^{(i,k)} + \frac{\epsilon^2}{2n} \Phi(w^{(i,k)})$
                $r^{(i,k+1)} \leftarrow w^{(i,k+1)} + \frac{\epsilon}{2n} \Phi(w^{(i,k+1)})$
                For all $e \in H \cup \{\bar{e}\}$, $\overline{r}_e^{(i,k+1)} \leftarrow r_e^{(i,k+1)}$
            else
                for $\zeta = \rho, 2\rho, 4\rho, \cdots, 2^{c_\rho}\rho$ do
                        ▷ $c_\rho$ is defined to be the smallest integer $c$ that satisfies $2^c \rho \ge \sqrt{n/\epsilon}$
                    Define the set $H_\zeta = \{e \in H \mid |\widetilde{C}\Delta^{(i,k)} - \widetilde{d}|_e \in [\zeta, 2\zeta)\}$.
                    If $\sum_{e\in H_\zeta} \overline{r}_e^{(i,k)} |\widetilde{C}\Delta^{(i,k)} - \widetilde{d}|_e^3 \ge \frac{\rho\Psi(\overline{r}^{(i,k)})}{\log(\frac{n}{\epsilon\rho})}$, set $\zeta^* \leftarrow \zeta$, and break.
                For all $e \in H_{\zeta^*}$, $w_e^{(i,k+1)} \leftarrow (1+\epsilon) w_e^{(i,k)} + \frac{\epsilon^2}{2n} \Phi(w^{(i,k)})$
                $r^{(i,k+1)} \leftarrow w^{(i,k+1)} + \frac{\epsilon}{2n} \Phi(w^{(i,k+1)})$
                For all $e \in H_{\zeta^*}$, $\overline{r}_e^{(i,k+1)} \leftarrow r_e^{(i,k+1)}$
            $k \leftarrow k + 1$
    return $x^{(T)}/T$
```

7 G. W. Brown and J. Von Neumann. 6. SOLUTIONS OF GAMES BY DIFFERENTIAL EQUATIONS. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games (AM-24), Volume I*, pages 73–80. Princeton University Press, December 1951. doi:10.1515/9781400881727-007.

8 Brian Bullins. Fast minimization of structured convex quartics. *arXiv preprint arXiv:1812.10349*, 2018. arXiv:1812.10349.

9 Yair Carmon, Arun Jambulapati, Qijia Jiang, Yujia Jin, Yin Tat Lee, Aaron Sidford, and Kevin Tian. Acceleration with a ball optimization oracle. *Advances in Neural Information Processing Systems*, 33:19052–19063, 2020.

10 Hui Han Chin, Aleksander Madry, Gary L Miller, and Richard Peng. Runtime guarantees for regression problems. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 269–282, 2013. doi:10.1145/2422436.2422469.

11 Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282, 2011. doi:10.1145/1993636.1993674.

12 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021. doi:10.1145/3424305.

13 Alina Ene and Adrian Vladu. Improved convergence for ℓ_{infty} and ℓ_1 regression via iteratively reweighted least squares. *Proceedings of Machine Learning Research*, 97, 2019.

14 Mehrdad Ghadiri, Richard Peng, and Santosh S Vempala. The bit complexity of efficient continuous optimization. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2059–2070. IEEE, 2023. doi:10.1109/FOCS57990.2023.00125.

15 M. R. Hestenes and E. Stiefel. On the convergence of the conjugate gradient method for singular liner operator equations. *J. Research Nat. Bur. Standards*, 49:409–436, 1952.

16 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 823–832, 2021. doi:10.1145/3406325.3451058.

17 Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 302–311, 1984. doi:10.1145/800057.808695.

18 Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014. doi:10.1137/1.9781611973402.16.

19 Cornelius Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49(1):33–53, 1952.

20 Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249. IEEE, 2015. doi:10.1109/FOCS.2015.23.

21 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory*, pages 2140–2157. PMLR, 2019. URL: http://proceedings.mlr.press/v99/lee19a.html.

22 Yin Tat Lee and Santosh S Vempala. Tutorial on the robust interior point method. *arXiv preprint arXiv:2108.04734*, 2021. arXiv:2108.04734.

23 Yang P Liu and Aaron Sidford. Faster energy maximization for faster maximum flow. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 803–814, 2020. doi:10.1145/3357713.3384247.

24 Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 253–262. IEEE, 2013. doi:10.1109/FOCS.2013.35.

25 Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016. doi:10.1109/FOCS.2016.70.

26 Renato D. C. Monteiro and B. F. Svaiter. An Accelerated Hybrid Proximal Extragradient Method for Convex Optimization and Its Implications to Second-Order Methods. *SIAM Journal on Optimization*, 23(2):1092–1125, January 2013. doi:10.1137/110833786.

27 Y Nesterov. A method for solving the convex programming problem with convergence rate o(1/k2). *Dokl Akad Nauk SSSR*, 269:543, 1983.

28 Yu E. Nesterov and Arkadii Nemirovskii. Self-concordant functions and polynomial-time methods in convex programming. *Report, Central Economic and Mathematic Institute, USSR Acad. Sci*, 1989.

29 Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994. doi:10.1137/1.9781611970791.

30 James Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988. doi:10.1007/BF01580724.

31 Jonah Sherman. Nearly maximum flows in nearly linear time. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 263–269. IEEE, 2013. doi:10.1109/FOCS.2013.36.

32 Aaron Sidford and Kevin Tian. Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 922–933. IEEE, 2018. doi:10.1109/FOCS.2018.00091.

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

34 David Woodruff and Qin Zhang. Subspace embeddings and\ell_p-regression using exponential random variables. In *Conference on Learning Theory*, pages 546–567. PMLR, 2013.