Sketching for Infeasible Interior-Point Methods

Candidate Number: 1054613
M.Sc. in Mathematical Sciences
University of Oxford

Trinity Term 2021

Abstract

At the heart of interior-point methods for linear programming is the solution of a least-squares type linear system in every iteration. A recently proposed preconditioning technique based on sketching allows employing iterative methods for the solution of this linear system. We simplified the original proofs on how this technique can be combined with an infeasible interior-point method to guarantee convergence and conducted novel experiments to verify the effectiveness of the preconditioners and compare the performance with classical direct solvers.

Word count: 7471

Contents

1. Overview

Optimisation problems are ubiquitous in science and industry. An important subclass of optimisation problems are linear programs (LPs), where the objective function as well as the constraints on the variables are linear. They take a primary role because of the extensive research on them and the existence of efficient programs to solve them optimally. Many problems can be modelled or approximated in this way, for example ℓ_1-regularized SVMs [Zhu+04] in Machine Learning, MAP inference of Markov random fields [MG11] in statistics and linear netlength minimisation [BV08] in Chip Design. As the problems grow more complex the number of variables increases, and we need more efficient algorithms for solving ever larger linear programs.

One of the first methods that came about for solving LPs was Dantzig's simplex method [Dan65], which is still in use today. For LPs the set of feasible points is a polytope and the simplex method can be thought of as walking through the vertices of this polytope while reducing the objective function at every step until the minimum is reached. It was later shown that LPs in n dimensions exist whose polytope has 2^n vertices and which are all visited by the simplex method so that the worst case complexity of the simplex method is exponential. After a theoretical breakthrough by Khachiyan's ellipsoid method [Kha79; Kha80], a polynomial time algorithm for LPs, which unfortunately is not competitive in practice, the Interior-Point Method (IPM) developed by Karmarkar [Kar84] was shown to converge in polynomial time while also being competitive in practice. Since then, many variants of IPMs have been developed. Some of them have better theoretical guarantees others are faster in practice.

The main bottleneck of IPMs is the solution of a linear system which determines the search direction in every step. Carefully reformulating this system, solving it can be broken down to solving the symmetric positive-definite linear system $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\Delta\boldsymbol{y} = \boldsymbol{p}$ for some vector $\boldsymbol{p}$. Here, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is the matrix of constraints, $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ is a diagonal matrix depending on the current iterates and the equation is called the *normal equation*. Typically, this normal equation is solved via direct methods using a Cholesky decomposition of $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$ [Wri97, p. 17], but this is exceedingly expensive if the matrix is large and dense. If the matrix is sparse, one can speed up the process [NP93] but $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$ can be dense even if $\boldsymbol{A}$ is sparse. Iterative methods that only rely on matrix-vector products can use the sparsity of $\boldsymbol{A}$ but converge only very slowly because $\boldsymbol{D}^2$ becomes more and more ill-conditioned as the algorithm progresses. Therefore, good preconditioners are crucial for employing iterative methods. Based on the work by Chowdhury et al. [Cho+20] we will show a way to effectively precondition this linear system in the special case when $\boldsymbol{A}$ is sparse and the number of constraints is much smaller than the number of variables, using tools from Randomized Linear Algebra (RLA).

In the last decade many advances in numerical linear algebra where achieved by incor-

porating randomness. We will focus on a RLA tool called sketching. Woodruff [Woo14] gives an overview on how sketching matrices lead to dimensionality reductions that can be applied to least-squares problems, low-rank approximation and graph-sparsification. The key idea is that a matrix $\boldsymbol{B} \in \mathbb{R}^{n \times m}$ with $m \ll n$ has an m-dimensional column space that forms a subspace of $\mathbb{R}^n$. This ambient dimension of n is unnecessarily large. For $w \in \mathbb{N}$ large enough there are ensembles of random $\mathbb{R}^n \to \mathbb{R}^w$ linear maps such that for any fixed m-dimensional subspace of $\mathbb{R}^n$ its geometry is approximately preserved by a randomly chosen map with high probability. These ensembles are called *oblivious subspace embeddings*. The size of w depends on m, the approximation tolerance and the failure probability but not on n, so we can significantly reduce dimensions at the cost of some approximation error and a failure probability. While matrices with properly normalized Gaussian entries form oblivious subspace embeddings [Woo14, Theorem 6], applying them to a vector in $\mathbb{R}^n$ is expensive as these matrices are dense. Ailon and Chazelle [AC06] developed so-called Fast-Johnson-Lindenstrauss-Transforms which enable fast matrix-vector multiplications. For sparse matrices $\boldsymbol{B}$ there exist sparse embedding matrices [Ach03; Coh16] to speed up forming the product.

In this work, we will assume that $\boldsymbol{A}$ is a sparse $m \times n$ matrix with $m \ll n$ and show that a decomposition of the product of a sparse embedding matrix and $\boldsymbol{DA}^T$ can serve as a preconditioner for the $\boldsymbol{AD}^2\boldsymbol{A}^T$. Chowdhury et al. [Cho+20] combined this idea for preconditioning with an iterative method to solve the normal equation and the convergence proof of Monteiro and O'Neal [MO03]. Monteiro and O'Neal [MO03] showed that a long-step infeasible primal-dual IPM converges in $O(n^2 \log(1/\varepsilon))$ iterations, where ε is the required accuracy, if the Newton direction is determined sufficiently accurate in every iteration. This allows us to bound the total number of operations by bounding the runtime of each iteration given that we want a high overall success probability.

The same idea of preconditioning using a decomposition of a reduced matrix was previously successfully applied in Blendenpik [AMT10] and LSRN [MSM14] to linear least-squares problems. Blendenpik [AMT10] uses a dimensionality reduction technique based on random row sampling after applying a transformation that aims to even out the importance of each row, and determines the preconditioner using a QR decomposition. The authors of LSRN [MSM14] decided to use a Gaussian sketch and an SVD of the sketched matrix as a preconditioner. Afterwards, both methods use an iterative least-squares solver to find the solution to the preconditioned problem. Both papers report that the generated preconditioners bring the condition number down to a small constant and provide a great speed-up compared to direct methods based on decompositions of the full-sized matrix.

The special case of LPs with sparse constraint matrices $\boldsymbol{A}$ and $m \ll n$ considered in this thesis is not just convenient because of the RLA results but also comes up in practice. In particular, optimizing ℓ_1-regularized SVMs [Zhu+04] with more features than data points and recovering sparse vectors in compressed sensing [YZ11] produce LP instances with these properties. We want to highlight a few previous papers on LPs in this setting. London et al. [Lon+18] devised a black box algorithm that speeds up any given algorithm for solving packing LPs (where each variable is restricted to lie between 0 and 1) if the number of variables is much larger than the number of constraints. This is

achieved by sampling a fraction of the variables randomly, solving the restricted problem optimally and setting each variable to 0 or 1 depending on the outcome. Brand et al. [Bra+20] employed RLA ideas and inverse maintenance in a very sophisticated way to achieve a theoretical running time of $\tilde{O}(nm + m^2)$ without assuming that $\boldsymbol{A}$ is sparse. Since the data structures, algorithms and proofs are all very technical and complicated, it is not established if efficient implementations of their ideas exist. Lastly, as already mentioned, [Cho+20] is the basis of this thesis. Their main results are described above and will be presented in detail in the following chapters.

This thesis is structured as follows. The notation for Interior-Point Methods and the specific long-step infeasible IPM used in [MO03] along with a convergence result will be presented in Chapter 2. Chapter 3 will be concerned with the concepts of sketching that are useful for us and how its approximation guarantees lead to good preconditioners. Compared to [Cho+20] we will show that a QR decomposition of the sketched matrix produces a preconditioner of the same quality as an SVD. Chapter 4 will contain all the proofs to show that under suitable conditions the preconditioning technique can be used to obtain a high accuracy LP solution using $O(n^2 \log(n)(m^3 + \text{nnz}(\boldsymbol{A})))$ operations with only a small failure probability. The proofs are adapted from Chowdhury et al. [Cho+20] and simplified to first establish the effectiveness of the preconditioner and the convergence result for the IPM separately and combine them afterwards. This way, the preconditioning technique can be integrated into convergence proofs for other IPMs more easily in the future. Lastly, we will incorporate the presented preconditioning technique into an existing IPM implementation based on [AA00] and evaluate its effectiveness in Chapter 5.

2. A Brief Introduction to IPMs for Linear Programming

2.1. Central Concepts

The standard form linear programming (LP) problem is given by

$$\min \boldsymbol{c}^T \boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \tag{P}$$

and its dual is

$$\max \boldsymbol{b}^T \boldsymbol{y} \quad \text{subject to} \quad \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c},\ \boldsymbol{s} \geq \boldsymbol{0} \tag{D}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$ are given and $\boldsymbol{x}, \boldsymbol{s} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ are the variables. A triple $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ is called *feasible* if it satisfies the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, $\boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}$ and $\boldsymbol{x}, \boldsymbol{s} \geq \boldsymbol{0}$. The two linear programs (P) and (D) are closely connected. For feasible $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ we always have $\boldsymbol{c}^T\boldsymbol{x} \geq \boldsymbol{b}^T\boldsymbol{y}$ and the joint solutions satisfy $\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{b}^T\boldsymbol{y}$. Therefore, if there are feasible points, both linear programs are feasible and bounded and solutions exist. Furthermore, as the optimal value of the objective function for both LPs is the same, it is upper bounded by $\boldsymbol{c}^T\boldsymbol{x}$ and lower bounded by $\boldsymbol{b}^T\boldsymbol{y}$ at any feasible point. $\boldsymbol{x}^T\boldsymbol{s} = \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{y}$ can thus be interpreted as a measure of how close to optimality a given feasible point is.

As already seen, necessary and sufficient optimality conditions are given by the feasibility conditions and $\boldsymbol{x}^T\boldsymbol{s} = \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{y} = 0$. Because the entries of $\boldsymbol{x}$ and $\boldsymbol{s}$ are both nonnegative, this is equivalent to the complementarity condition $\boldsymbol{x} \circ \boldsymbol{s} = \boldsymbol{0}$ requiring that for each $1 \leq i \leq n$ at least one of x_i and s_i is zero. Necessary and sufficient optimality conditions for both problems are therefore given by

$$\boldsymbol{r}_p = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0} \tag{2.1a}$$
$$\boldsymbol{r}_d = \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} = \boldsymbol{0} \tag{2.1b}$$
$$\boldsymbol{x}, \boldsymbol{s} \geq \boldsymbol{0} \tag{2.1c}$$
$$\boldsymbol{x} \circ \boldsymbol{s} = \boldsymbol{0} \tag{2.1d}$$

Here, $\boldsymbol{r}_p$ and $\boldsymbol{r}_d$ denote the primal and dual residuals and $\boldsymbol{r} = (\boldsymbol{r}_p, \boldsymbol{r}_d)$ denotes their concatenation as a vector in $\mathbb{R}^{m+n}$. We denote the set of feasible points with $\mathcal{F}$ and the set of solutions as $\mathcal{F}^*$. The feasible points which satisfy Equation (2.1c) strictly (all entries need to be positive) are called *strictly feasible* and the set of strictly feasible points is denoted by $\mathcal{F}^o$. Because we will focus on a so-called infeasible IPM it is also useful to define $\mathcal{G} := \mathbb{R}^n_{>0} \times \mathbb{R}^m \times \mathbb{R}^n_{>0}$ as the set of points that satisfy Equation (2.1c) strictly without requiring feasibility.

Assume strictly feasible points exist, then the unique solutions $(\boldsymbol{x}_\tau, \boldsymbol{y}_\tau, \boldsymbol{s}_\tau)$ of

$$\boldsymbol{A}\boldsymbol{x}_\tau - \boldsymbol{b} = \boldsymbol{0} \tag{2.2a}$$
$$\boldsymbol{A}^T\boldsymbol{y}_\tau + \boldsymbol{s}_\tau - \boldsymbol{c} = \boldsymbol{0} \tag{2.2b}$$
$$\boldsymbol{x}_\tau, \boldsymbol{s}_\tau > \boldsymbol{0} \tag{2.2c}$$
$$\boldsymbol{x}_\tau \circ \boldsymbol{s}_\tau = \tau\boldsymbol{1}. \tag{2.2d}$$

for $\tau > 0$ form a continuous path of strictly feasible points which is called the *central path*, see [Wri97, Theorem 2.8]. Moreover, the elements of the central path converge towards a solution as $\tau \to 0$.

Primal-dual IPMs now try to follow this path towards a solution by maintaining iterates in some neighbourhood of the central path and computing the search direction in each step as a Newton direction towards an element of the central path. In this thesis we focus on an IPM variant with good practical properties. Its neighbourhood is defined by

$$\mathcal{N}_{-\infty}(\gamma) = \left\{ (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{G} \;\middle|\; \boldsymbol{x} \circ \boldsymbol{s} \geq (1-\gamma)\mu\boldsymbol{1},\ \frac{\|\boldsymbol{r}\|_2}{\|\boldsymbol{r}^0\|_2} \leq \frac{\mu}{\mu^0} \right\}. \tag{2.3}$$

where $\mu = \boldsymbol{x}^T\boldsymbol{s}/n$, i.e. the average value of $x_i s_i$, and the parameter $\gamma \in (0, 1)$ controls the size of the neighbourhood. The condition $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{G}$ is shared by all IPMs. It keeps the $\boldsymbol{x}$ and $\boldsymbol{s}$ components of the iterates in the interior of the nonnegative orthant. The second condition qualifies this IPM as a *long-step* IPM as this one-sided ∞-norm bound allows for longer steps inside the neighbourhood compared to a two-sided 2-norm bound such as $\|\boldsymbol{x} \circ \boldsymbol{s} - \mu\boldsymbol{1}\|_2 \leq \gamma\mu$. Neighbourhoods involving those bounds are known as *short-step* IPMs and give better theoretical guarantees while having worse practical performance [Wri97, Chapter 5]. Lastly, the bound on the residuals (where $\boldsymbol{r}^0$ and μ^0 are defined by the starting point $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$) implies that we will allow infeasible iterates as long as their residuals decrease to zero at least as fast as μ does. Any algorithm that maintains iterates inside $\mathcal{N}_{-\infty}(\gamma)$ and is able to decrease the corresponding μ values to zero produces iterates which become arbitrarily close to optimality because $\|\boldsymbol{r}\|_2$ and $x_i s_i$ converge to zero.

2.2. The Normal Equation

The search direction in IPMs is determined by a Newton system that aims to find the root of

$$F(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \coloneqq \begin{pmatrix} \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c} \\ \boldsymbol{x} \circ \boldsymbol{s} - \sigma\mu\boldsymbol{1} \end{pmatrix} \tag{2.4}$$

where $\sigma \in (0, 1)$ is a parameter. In other words, we aim at an element further along the central path by choosing $\tau = \sigma\mu$. The search direction $(\Delta\boldsymbol{x}, \Delta\boldsymbol{y}, \Delta\boldsymbol{s})$ is therefore

determined by

$$\underbrace{\begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{pmatrix}}_{F'(\boldsymbol{x},\boldsymbol{y},\boldsymbol{s})} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \underbrace{\begin{pmatrix} -\boldsymbol{r}_p \\ -\boldsymbol{r}_d \\ -\boldsymbol{x} \circ \boldsymbol{s} + \sigma\mu\boldsymbol{1} \end{pmatrix}}_{-F(\boldsymbol{x},\boldsymbol{y},\boldsymbol{s})} \tag{2.5}$$

where $\boldsymbol{X} = \operatorname{diag}(\boldsymbol{x})$ and $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{s})$. Because $\boldsymbol{X}$ and $\boldsymbol{S}$ are diagonal they can be easily inverted, and we can simplify the linear system into smaller ones and a series of matrix-vector products. Equation (2.5) is equivalent to

$$\begin{pmatrix} \boldsymbol{X}^{-1}\boldsymbol{S} & \boldsymbol{A}^T \\ \boldsymbol{A} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{r}_d - \boldsymbol{s} + \sigma\mu\boldsymbol{X}^{-1}\boldsymbol{1} \\ -\boldsymbol{r}_p \end{pmatrix} \tag{2.6a}$$

$$\Delta\boldsymbol{s} = -\boldsymbol{s} + \sigma\mu\boldsymbol{X}^{-1}\boldsymbol{1} - \boldsymbol{X}^{-1}\boldsymbol{S}\Delta\boldsymbol{x} \tag{2.6b}$$

in which the first equation is called the *augmented system* and

$$\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\Delta\boldsymbol{y} = -\boldsymbol{r}_p + \boldsymbol{A}(-\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{x} - \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1}) =: \boldsymbol{p} \tag{2.7a}$$

$$\Delta\boldsymbol{s} = -\boldsymbol{r}_d - \boldsymbol{A}^T\Delta\boldsymbol{y} \tag{2.7b}$$

$$\Delta\boldsymbol{x} = -\boldsymbol{x} + \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{S}^{-1}\boldsymbol{X}\Delta\boldsymbol{s} \tag{2.7c}$$

in which the first equation is called the *normal equation*, see [Wri97, p. 16]. Here, $\boldsymbol{D}$ is defined as the diagonal matrix $\boldsymbol{S}^{-1/2}\boldsymbol{X}^{1/2}$ to simplify the notation. Equation (2.7a) is called the normal equation because it can be written as the normal equation of a least-squares system with coefficient matrix $\boldsymbol{D}\boldsymbol{A}^T$ if $\boldsymbol{r}_p = \boldsymbol{0}$.

2.3. Shifting Error Terms

Our goal is to solve the normal equation using a preconditioner and an iterative solver. To figure out how many iterations of this solver are needed to obtain a sufficiently accurate solution, assume that some error term $\tilde{\boldsymbol{f}}$ is introduced. This means $\Delta\boldsymbol{y}$ solves

$$\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\Delta\boldsymbol{y} = \boldsymbol{p} + \tilde{\boldsymbol{f}} \tag{2.8}$$

instead of Equation (2.7a) and $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{s}$ are obtained from Equations (2.7b) and (2.7c) as before. Because calculating the search direction in this way is equivalent to using Equations (2.7a) to (2.7c) with $\boldsymbol{b} + \tilde{\boldsymbol{f}}$ instead of $\boldsymbol{b}$, the error propagates from the normal equation to the primal feasibility component in Equation (2.5):

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{pmatrix} \begin{pmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{r}_p \\ -\boldsymbol{r}_d \\ -\boldsymbol{x} \circ \boldsymbol{s} + \sigma\mu\boldsymbol{1} \end{pmatrix} + \begin{pmatrix} \tilde{\boldsymbol{f}} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{pmatrix} \tag{2.9}$$

For the convergence proofs of Monteiro and O'Neal [MO03] to work we need that the error in calculating the Newton direction only affects the complementarity component of the Newton system. This can be achieved by constructing a perturbation vector $\boldsymbol{v} \in \mathbb{R}^n$

satisfying $\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} = \tilde{\boldsymbol{f}}$ and subtracting $\boldsymbol{S}^{-1}\boldsymbol{v}$ from $\Delta\boldsymbol{x}$. The perturbed Newton direction $(\hat{\Delta}\boldsymbol{x}, \hat{\Delta}\boldsymbol{y}, \hat{\Delta}\boldsymbol{s})$ is determined using

$$\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} = \boldsymbol{p} + \tilde{\boldsymbol{f}} \tag{2.10a}$$

$$\hat{\Delta}\boldsymbol{s} = -\boldsymbol{r}_d - \boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} \tag{2.10b}$$

$$\hat{\Delta}\boldsymbol{x} = -\boldsymbol{x} + \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{S}^{-1}\boldsymbol{X}\hat{\Delta}\boldsymbol{s} - \boldsymbol{S}^{-1}\boldsymbol{v} \tag{2.10c}$$

and solves

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{X} \end{pmatrix} \begin{pmatrix} \hat{\Delta}\boldsymbol{x} \\ \hat{\Delta}\boldsymbol{y} \\ \hat{\Delta}\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{r}_p \\ -\boldsymbol{r}_d \\ -\boldsymbol{x} \circ \boldsymbol{s} + \sigma\mu\boldsymbol{1} \end{pmatrix} - \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{v} \end{pmatrix} \tag{2.11}$$

as shown by Monteiro and O'Neal [MO03, p. 10]. Hence, we successfully shifted the error term in the Newton system. We will see that we can construct $\boldsymbol{v}$ from $\tilde{\boldsymbol{f}}$ in such a way that the size of the error $\|\boldsymbol{v}\|_2$ is bounded in terms of the error $\|\tilde{\boldsymbol{f}}\|_2$ in the next chapter. To see why this is important, consider the following IPM algorithm and the corresponding convergence result.

2.4. The Infeasible IPM Algorithm

Algorithm 1: Infeasible IPM

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, $\gamma \in (0,1)$, $\sigma \in (0,1)$, tolerance $\varepsilon_\mu \in (0,1)$ and an initial point $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0)$

Initialize $k = 0$ and $\mu^0 = (\boldsymbol{x}^0)^T\boldsymbol{s}^0/n$;

while $\mu^k = (\boldsymbol{x}^k)^T\boldsymbol{s}^k/n > \varepsilon_\mu\mu^0$ **do**

  Compute $\boldsymbol{r}_p^k = \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b}$, $\boldsymbol{r}_d^k = \boldsymbol{A}^T\boldsymbol{y}^k + \boldsymbol{s}^k - \boldsymbol{c}$, $\boldsymbol{X} = \operatorname{diag}(\boldsymbol{x}^k)$, $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{s}^k)$;

  Solve the following linear system such that $\|\boldsymbol{v}\|_2 \leq \gamma\sigma\mu^k/4$

$$\begin{pmatrix} \boldsymbol{A} & 0 & 0 \\ 0 & \boldsymbol{A}^T & \boldsymbol{I} \\ \boldsymbol{S} & 0 & \boldsymbol{X} \end{pmatrix} \begin{pmatrix} \hat{\Delta}\boldsymbol{x} \\ \hat{\Delta}\boldsymbol{y} \\ \hat{\Delta}\boldsymbol{s} \end{pmatrix} = \begin{pmatrix} -\boldsymbol{r}_p^k \\ -\boldsymbol{r}_d^k \\ -\boldsymbol{x}^k \circ \boldsymbol{s}^k + \sigma\mu^k\boldsymbol{1} \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ \boldsymbol{v} \end{pmatrix}; \tag{2.12}$$

  Determine the maximum $\tilde{\alpha} \in [0,1]$ such that for all $\alpha \in [0,\tilde{\alpha}]$ we have $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k) + \alpha(\hat{\Delta}\boldsymbol{x}, \hat{\Delta}\boldsymbol{y}, \hat{\Delta}\boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$;

  Compute $\bar{\alpha} = \arg\min\{\, (\boldsymbol{x}^k + \alpha\hat{\Delta}\boldsymbol{x})^T(\boldsymbol{s}^k + \alpha\hat{\Delta}\boldsymbol{s}) \mid \alpha \in [0,\tilde{\alpha}] \,\}$;

  Set $(\boldsymbol{x}^{k+1}, \boldsymbol{y}^{k+1}, \boldsymbol{s}^{k+1}) = (\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k) + \bar{\alpha}(\hat{\Delta}\boldsymbol{x}, \hat{\Delta}\boldsymbol{y}, \hat{\Delta}\boldsymbol{s})$;

  Increment $k \leftarrow k + 1$;

return $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$;

Theorem 2.1. *Assume that* $\gamma \in (0,1)$ *and* $\sigma \in (0, \frac{4}{5})$ *are constant and that the initial point* $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ *satisfies* $\boldsymbol{x}^0 \circ \boldsymbol{s}^0 \geq (1-\gamma)\mathbf{1}$ *and* $(\boldsymbol{x}^0, \boldsymbol{s}^0) \geq (\boldsymbol{x}^*, \boldsymbol{s}^*)$ *for some* $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$. *Then Algorithm 1 generates an iterate* $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$ *satisfying* $\mu^k \leq \varepsilon_\mu \mu^0$ *and* $\|\boldsymbol{r}^k\| \leq \varepsilon_\mu \|\boldsymbol{r}^0\|$ *within* $O(n^2 \log(1/\varepsilon_\mu))$ *iterations.*

Note that apart from conditions for the initial point this theorem only requires the search direction to satisfy Equation (2.11) and $\|\boldsymbol{v}\|_2$ to be sufficiently small. The proof for Theorem 2.1 will be given in Section 4.1.

3. Sketching — A Useful Dimensionality Reduction Technique

Based on [Cho+20] we will use sketching to determine a preconditioner for $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$. These results not only apply to this specific matrix but all symmetric positive definite matrices of the form $\boldsymbol{B}^T\boldsymbol{B}$ for some $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ with $m \ll n$. To emphasise this, the theorems will all involve a general matrix $\boldsymbol{B}$ and their consequences for $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T$ will be discussed afterwards.

3.1. Overview and a Sparse Sketching Result

The following introduction to sketching is based on [Woo14] and begins with the central concept of a subspace embedding:

Definition 3.1. Let $\varepsilon \in (0, 1)$. A $(1 \pm \varepsilon)$ ℓ_2*-subspace embedding* for the column space of a matrix $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ is a matrix $\boldsymbol{W} \in \mathbb{R}^{w\times n}$ such that

$$(1-\varepsilon)\|\boldsymbol{B}\boldsymbol{z}\|_2^2 \leq \|\boldsymbol{W}\boldsymbol{B}\boldsymbol{z}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{B}\boldsymbol{z}\|_2^2$$

for all $\boldsymbol{z} \in \mathbb{R}^m$.

Note that this definition does not depend on a particular basis for the representation of the column space of $\boldsymbol{B}$. Nonetheless, we will often refer to $\boldsymbol{W}$ as a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding of $\boldsymbol{B}$. Intuitively, a subspace embedding of some m-dimensional subspace of $\mathbb{R}^n$ is a linear map from the (large) space $\mathbb{R}^n$ to a (smaller) space $\mathbb{R}^w$ that approximately preserves all the distances of the subspace. There are multiple equivalent characterisations of subspace embeddings used in the literature.

Theorem 3.2. *Assume that $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ has full column rank and let*

$$\mathcal{S} = \{\, \boldsymbol{B}\boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{R}^m \,\} = \{\, \boldsymbol{U}\boldsymbol{z} \mid \boldsymbol{z} \in \mathbb{R}^m \,\} \subset \mathbb{R}^n$$

for some matrix $\boldsymbol{U} \in \mathbb{R}^{n\times m}$ with orthonormal columns. Then the following statements are equivalent

1. *$\boldsymbol{W}$ is a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding for the column space of $\boldsymbol{B}$*
2. *$(1-\varepsilon)\|\boldsymbol{B}\boldsymbol{z}\|_2^2 \leq \|\boldsymbol{W}\boldsymbol{B}\boldsymbol{z}\|_2^2 \leq (1+\varepsilon)\|\boldsymbol{B}\boldsymbol{z}\|_2^2$ for all $\boldsymbol{z} \in \mathbb{R}^d$*

3. $|\|\boldsymbol{W}\boldsymbol{s}\|_2^2 - 1| \leq \varepsilon$ *for all* $\boldsymbol{s} \in \mathcal{S}$ *with* $\|\boldsymbol{s}\|_2 = 1$

4. $\|\boldsymbol{U}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{U} - \boldsymbol{I}_d\|_2 \leq \varepsilon$

Proof. The first equivalence holds by definition. Statement 3 is equivalent to

$$1 - \varepsilon \leq \|\boldsymbol{W}\boldsymbol{s}\|_2^2 \leq 1 + \varepsilon \quad \forall \boldsymbol{s} \in \mathcal{S} \text{ with } \|\boldsymbol{s}\|_2 = 1.$$

In this form statement 2 immediately implies 3 and choosing $\boldsymbol{s} = \|\boldsymbol{B}\boldsymbol{z}\|_2^{-1}\boldsymbol{B}\boldsymbol{z}$ for $\boldsymbol{B}\boldsymbol{z} \neq \boldsymbol{0}$ and multiplying both sides by $\|\boldsymbol{B}\boldsymbol{z}\|_2^2$ we obtain that the two statements are indeed equivalent.

Note that the matrix $\boldsymbol{U}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{U} - \boldsymbol{I}_d$ is real and symmetric and therefore its 2-norm is given by the maximum absolute value of the Rayleigh quotient [Dem97, Equation (5.2)]. Using $\|\boldsymbol{U}\boldsymbol{e}\|_2 = \|\boldsymbol{e}\|_2$, this implies

$$\begin{aligned}
\|\boldsymbol{U}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{U} - \boldsymbol{I}_d\|_2 &= \max_{\boldsymbol{e}\in\mathbb{R}^d, \|\boldsymbol{e}\|_2=1} \left|\boldsymbol{e}^T\left(\boldsymbol{U}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{U} - \boldsymbol{I}_d\right)\boldsymbol{e}\right| \\
&= \max_{\boldsymbol{e}\in\mathbb{R}^d, \|\boldsymbol{e}\|_2=1} \left|(\boldsymbol{U}\boldsymbol{e})^T\boldsymbol{W}^T\boldsymbol{W}(\boldsymbol{U}\boldsymbol{e}) - \boldsymbol{e}^T\boldsymbol{e}\right| \\
&= \max_{\boldsymbol{s}\in\mathcal{S}, \|\boldsymbol{s}\|_2=1} \left|\boldsymbol{s}^T\boldsymbol{W}^T\boldsymbol{W}\boldsymbol{s} - 1\right| \\
&= \max_{\boldsymbol{s}\in\mathcal{S}, \|\boldsymbol{s}\|_2=1} \left|\|\boldsymbol{W}\boldsymbol{s}\|_2^2 - 1\right|.
\end{aligned}$$

and proves the equivalence of statements 3 and 4. □

Using the QR decomposition of $\boldsymbol{B}$ it is always possible to design a 'perfect' subspace embedding with $\varepsilon = 0$ and $w = m$ but computing this QR decomposition is expensive. Instead, we can use random matrices $\boldsymbol{W}$ which have a high chance of being a subspace embedding.

Definition 3.3. Let $m, n, w \in \mathbb{N}$ and $\delta, \varepsilon \in (0, 1)$ be given. A probability distribution Π over matrices $\boldsymbol{W} \in \mathbb{R}^{w\times n}$ is an *oblivious ℓ_2-subspace embedding* (OSE) if, for any matrix $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ a matrix $\boldsymbol{W}$ drawn from Π is a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding with probability at least $1 - \delta$.

Because we will later compute a decomposition of the sketched matrix $\boldsymbol{W}\boldsymbol{B}$, we want w to be as small as possible. A classic result in sketching is that there are OSEs with $w = O((m + \log(1/\delta))\varepsilon^{-2})$ without any dependence on n. Woodruff [Woo14, Theorem 6] shows how this follows from the Johnson-Lindenstrauss Lemma and that it can be achieved by choosing the matrix entries to be properly normalized independent Gaussian random variables. Because a matrix $\boldsymbol{W}$ with Gaussian entries is dense, computing the matrix product $\boldsymbol{W}\boldsymbol{B}$ takes $O(w \operatorname{nnz}(\boldsymbol{B}))$ operations, where $\operatorname{nnz}(\boldsymbol{B})$ denotes the number of nonzero entries of $\boldsymbol{B}$. A sparse OSE where all sampled matrices only have s nonzero entries per column can improve the cost of this product to $O(s \operatorname{nnz}(\boldsymbol{B}))$. The following sparse OSE result is taken from [Coh16] and will not be proven here. Chowdhury et al. [Cho+20] use the sketching result from [CNW16], but we replaced it by a simpler one giving the same complexity guarantees in the end.

Definition 3.4 ([CFS21])**.** An *s-hashing matrix* $\boldsymbol{W} \in \mathbb{R}^{w \times n}$ is constructed in the following way: For each column s entries are sampled randomly without replacement and for each such entry its value is randomly drawn from $\{-\frac{1}{\sqrt{s}}, \frac{1}{\sqrt{s}}\}$. All other entries of $\boldsymbol{W}$ are set to zero. This defines a probability distribution over $\mathbb{R}^{w \times n}$ where each sampled matrix has exactly $s \cdot n$ nonzero entries.

Theorem 3.5 (Theorem 4.2 in [Coh16])**.** *For any $\delta < 1/2$, $\varepsilon < 1/2$ and $m, n \in \mathbb{N}$ there exist $s, w \in \mathbb{N}$ with*

$$w = O\left(\frac{m \log(m/\delta)}{\varepsilon^2}\right) \quad \text{and} \quad s = O\left(\frac{\log(m/\delta)}{\varepsilon}\right)$$

such that the s-hashing matrices form an oblivious ℓ_2-subspace embedding.

3.2. Sketching-Based Preconditioning

To simplify the runtime bounds and the following discussion, we will from now on assume that $\varepsilon < 1/2$ is fixed and furthermore assume that $\boldsymbol{W} \in \mathbb{R}^{w \times n}$ is an s-hashing matrix and a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding of the column space of $\boldsymbol{B} \in \mathbb{R}^{n \times m}$. By Theorem 3.5 w and s can be chosen such that the latter assumption holds with probability at least $1 - \delta$ and

$$w = O(m \log(m/\delta)) \quad \text{and} \quad s = O(\log(m/\delta)). \tag{3.1}$$

How can these sketching results help with our goal of solving $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\Delta\boldsymbol{y} = \boldsymbol{p}$ with an iterative method? We will first show how a decomposition of the sketched matrix $\boldsymbol{W}\boldsymbol{B}$ can be used to determine a two-sided preconditioner for $\boldsymbol{B}^T\boldsymbol{B}$ which reduces the condition number down to a constant and then see how this can be applied to the equation above.

Chowdhury et al. [Cho+20] use an SVD of the sketched matrix $\boldsymbol{W}\boldsymbol{B}$ to get a two-sided preconditioner, but we decided to construct it using the computationally cheaper QR decomposition $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{W}\boldsymbol{B}$. Here $\boldsymbol{R} \in \mathbb{R}^{m \times m}$ is upper triangular and $\boldsymbol{Q} \in \mathbb{R}^{w \times m}$ has orthonormal columns. To ensure that the QR decomposition exists we need that $\boldsymbol{B}$ has full column rank. Then we directly get $\operatorname{rank}(\boldsymbol{W}\boldsymbol{B}) = m$, because $\varepsilon < 1$ implies none of the elements in the column space of $\boldsymbol{B}$ can be in the nullspace of $\boldsymbol{W}$. The proposed preconditioned matrix has the form

$$\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{R}^{-1}. \tag{3.2}$$

Note that applying the preconditioner from the left and from the right in this way ensures that the matrix stays symmetric positive definite, and we can apply the CG algorithm to any linear system involving this matrix. Moreover, because $\boldsymbol{R}$ is upper triangular multiplying $\boldsymbol{R}^{-1}$ or $\boldsymbol{R}^{-T}$ with a vector can be done efficiently in $O(m^2)$ using a triangular solve. The following theorem shows that this preconditioner leads to bounds on the condition number of the matrix which can be used to bound the number of CG iterations needed to reach a certain error estimate.

Lemma 3.6. *Suppose $\boldsymbol{W}$ is a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding for $\boldsymbol{B}$ and $\boldsymbol{QR}$ is the QR decomposition of $\boldsymbol{WB}$. The singular values of $\boldsymbol{R}^{-T}\boldsymbol{B}^T$ and $\boldsymbol{BR}^{-1}$ are bounded by*

$$\frac{1}{1+\varepsilon} \leq \sigma_i^2(\boldsymbol{R}^{-T}\boldsymbol{B}^T) = \sigma_i^2(\boldsymbol{BR}^{-1}) \leq \frac{1}{1-\varepsilon} \quad \text{for } i = 1, \ldots, m$$

which implies the following bound on the condition number of their product:

$$\kappa_2(\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{BR}^{-1}) \leq \frac{1+\varepsilon}{1-\varepsilon}$$

Proof. The matrices $\boldsymbol{R}^{-T}\boldsymbol{B}^T$ and $\boldsymbol{BR}^{-1}$ are transposes of each other and hence have the same singular values. By the Courant–Fisher min-max theorem (e.g. [HJ91, Theorem 3.1.2]) the minimum and maximum squared singular values of $\boldsymbol{BR}^{-1}$ are the same as the minimum and maximum values of $\|\boldsymbol{BR}^{-1}\tilde{\boldsymbol{z}}\|_2^2/\|\tilde{\boldsymbol{z}}\|_2^2$ over all $\tilde{\boldsymbol{z}} \in \mathbb{R}^n$. By Definition 3.1 applied to $\boldsymbol{z} = \boldsymbol{R}^{-1}\tilde{\boldsymbol{z}}$ we have

$$(1-\varepsilon)\|\boldsymbol{BR}^{-1}\tilde{\boldsymbol{z}}\|_2^2 \leq \|\boldsymbol{WBR}^{-1}\tilde{\boldsymbol{z}}\|_2^2 = \|\boldsymbol{QRR}^{-1}\tilde{\boldsymbol{z}}\|_2^2 = \|\boldsymbol{Q}\tilde{\boldsymbol{z}}\|_2^2 = \|\tilde{\boldsymbol{z}}\|_2^2$$

for all $\tilde{\boldsymbol{z}} \in \mathbb{R}^m$ and similarly $(1+\varepsilon)\|\boldsymbol{BR}^{-1}\tilde{\boldsymbol{z}}\|_2^2 \geq \|\tilde{\boldsymbol{z}}\|_2^2$. Therefore,

$$\frac{1}{1+\varepsilon} \leq \frac{\|\boldsymbol{BR}^{-1}\tilde{\boldsymbol{z}}\|_2^2}{\|\tilde{\boldsymbol{z}}\|_2^2} \leq \frac{1}{1-\varepsilon}$$

which proves the bounds on the singular values. The condition number of a matrix is defined as the largest singular value divided by the smallest singular value. Because $\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{BR}^{-1} = (\boldsymbol{BR}^{-1})^T(\boldsymbol{BR}^{-1})$ the singular values of this matrix are just the squares of the singular values of $\boldsymbol{BR}^{-1}$, so

$$\kappa_2(\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{BR}^{-1}) = \frac{\sigma_{\max}^2(\boldsymbol{BR}^{-1})}{\sigma_{\min}^2(\boldsymbol{BR}^{-1})} \leq \frac{1+\varepsilon}{1-\varepsilon}$$

and we are done. □

Remark. Note that the condition number bound has *no dependence* on the condition number of $\boldsymbol{B}$ but is bounded by a constant that depends only on ε.

Lemma 3.7. *Suppose $\boldsymbol{W}$ is a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding for $\boldsymbol{B}$ for $\varepsilon < 1/2$ and $\boldsymbol{QR}$ is the QR decomposition of $\boldsymbol{WB}$. Let $\boldsymbol{q}^j$ be the iterates when solving*

$$\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{BR}^{-1}\boldsymbol{q} = \boldsymbol{R}^{-T}\boldsymbol{p}$$

with the CG algorithm and let $\boldsymbol{f}^j$ be the residuals defined by $\boldsymbol{f}^j = \boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{BR}^{-1}\boldsymbol{q}^j - \boldsymbol{R}^{-T}\boldsymbol{p}$. Then the ℓ_2-norm of the residuals converges at least linearly to zero:

$$\|\boldsymbol{f}^{j+1}\|_2 \leq \frac{\varepsilon}{1-\varepsilon}\|\boldsymbol{f}^j\|_2 \text{ for all } j \in \mathbb{N}$$

Proof. Theorem 8 of [Bou+09] showed that

$$\|\boldsymbol{f}^{j+1}\|_2 \leq \frac{\kappa_2(\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{R}^{-1}) - 1}{2}\|\boldsymbol{f}^j\|_2.$$

Using the bound on the condition number from Lemma 3.6 and $\varepsilon < 1/2$ we obtain

$$\frac{\kappa_2(\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{R}^{-1}) - 1}{2} \leq \frac{\frac{1+\varepsilon}{1-\varepsilon} - 1}{2} = \frac{\varepsilon}{1-\varepsilon} < 1$$

which proves the claim. □

The previous two theorems show that computing the QR decomposition of a sketched matrix $\boldsymbol{W}\boldsymbol{B}$ gives an effective way to solve any linear system of the form $\boldsymbol{B}^T\boldsymbol{B}\tilde{\boldsymbol{q}} = \boldsymbol{p}$ by replacing it with

$$\boldsymbol{R}^{-T}\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{q} = \boldsymbol{R}^{-T}\boldsymbol{p} \quad \text{and} \quad \tilde{\boldsymbol{q}} = \boldsymbol{R}^{-1}\boldsymbol{q} \tag{3.3}$$

and solving the first part using the CG method. Applying this technique to Equation (2.7a) we get

$$\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}\boldsymbol{q} = \boldsymbol{R}^{-T}\boldsymbol{p} \quad \text{and} \quad \Delta\boldsymbol{y} = \boldsymbol{R}^{-1}\boldsymbol{q} \tag{3.4}$$

by setting $\boldsymbol{B} = \boldsymbol{D}\boldsymbol{A}^T$.

3.3. Choosing the Perturbation Vector

Now, as mentioned in Section 2.3 we need to define a perturbation vector $\boldsymbol{v} \in \mathbb{R}^n$ to shift the error term in the Newton system. Because the CG method is applied to Equation (3.4) an error term $\boldsymbol{f} = \boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}\boldsymbol{q} - \boldsymbol{R}^{-T}\boldsymbol{p}$ propagates so that

$$\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\hat{\Delta\boldsymbol{y}} = \boldsymbol{p} + \boldsymbol{R}^T\boldsymbol{f} \tag{3.5}$$

and the perturbation vector has to satisfy $\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} = \boldsymbol{R}^T\boldsymbol{f}$ for a given $\boldsymbol{f}$. Similar to the method of Chowdhury et al. [Cho+20], we choose this vector by utilising the previously computed decomposition of the sketched matrix $\boldsymbol{W}\boldsymbol{D}\boldsymbol{A}^T$. This also allows us to bound the size of the perturbation vector by the size of $\boldsymbol{f}$.

Lemma 3.8. *Suppose $\boldsymbol{W}$ is an s-hashing matrix and $\boldsymbol{Q}\boldsymbol{R}$ is the QR decomposition of $\boldsymbol{W}\boldsymbol{D}\boldsymbol{A}^T$. Given a vector $\boldsymbol{f} \in \mathbb{R}^m$ the vector*

$$\boldsymbol{v} = (\boldsymbol{S}\boldsymbol{X})^{1/2}\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{f}$$

satisfies $\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} = \boldsymbol{R}^T\boldsymbol{f}$ and $\|\boldsymbol{v}\|_2 \leq n\sqrt{\mu}\|\boldsymbol{f}\|_2$.

Proof. Using the definition of $\boldsymbol{Q}$ and $\boldsymbol{R}$ we obtain

$$\begin{aligned}\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} &= \boldsymbol{A}\boldsymbol{S}^{-1}(\boldsymbol{S}\boldsymbol{X})^{1/2}\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{f} = \boldsymbol{A}\boldsymbol{D}\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{f} \\ &= (\boldsymbol{Q}\boldsymbol{R})^T\boldsymbol{Q}\boldsymbol{f} = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{f} = \boldsymbol{R}^T\boldsymbol{f}\end{aligned}$$

which proves the first part. To bound the norm consider

$$\|\boldsymbol{v}\|_2 = \|(\boldsymbol{S}\boldsymbol{X})^{1/2}\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{f}\|_2 \leq \|(\boldsymbol{S}\boldsymbol{X})^{1/2}\|_2\|\boldsymbol{W}^T\|_2\|\boldsymbol{Q}\boldsymbol{f}\|_2.$$

Now, $\|(\boldsymbol{S}\boldsymbol{X})^{1/2}\|_2 \leq \|(\boldsymbol{S}\boldsymbol{X})^{1/2}\|_F = \sqrt{n\mu}$ by definition of μ and $\|\boldsymbol{Q}\boldsymbol{f}\|_2 = \|\boldsymbol{f}\|_2$ because $\boldsymbol{Q}$ has orthonormal columns. Lastly, we need to bound $\|\boldsymbol{W}^T\|_2$. Remember that $\boldsymbol{W}$ is constructed by choosing exactly s nonzero entries per column and each of these entries is drawn from $\{-\frac{1}{\sqrt{s}}, \frac{1}{\sqrt{s}}\}$. Let $\mathcal{J}_i$ denote the set of indices of the nonzero entries of column i in $\boldsymbol{W}$ and let $\boldsymbol{z}_{\mathcal{J}_i} \in \mathbb{R}^s$ denote the restriction of $\boldsymbol{z} \in \mathbb{R}^w$ to these indices. Using standard norm inequalities

$$\begin{aligned}\|\boldsymbol{W}^T\boldsymbol{z}\|_2^2 &= \sum_{i=1}^{n}\left(\sum_{j=1}^{w}\boldsymbol{W}_{ji}\boldsymbol{z}_j\right)^2 = \sum_{i=1}^{n}\left(\sum_{j\in\mathcal{J}_i}\boldsymbol{W}_{ji}\boldsymbol{z}_j\right)^2 \\ &\leq \sum_{i=1}^{n}\left(\sum_{j\in\mathcal{J}_i}|\boldsymbol{W}_{ji}||\boldsymbol{z}_j|\right)^2 = \sum_{i=1}^{n}\left(\frac{1}{\sqrt{s}}\|\boldsymbol{z}_{\mathcal{J}_i}\|_1\right)^2 \\ &\leq \sum_{i=1}^{n}\left(\frac{1}{\sqrt{s}}\sqrt{s}\|\boldsymbol{z}_{\mathcal{J}_i}\|_2\right)^2 = \sum_{i=1}^{n}\|\boldsymbol{z}_{\mathcal{J}_i}\|_2^2 \\ &\leq n\|\boldsymbol{z}\|_2^2\end{aligned}$$

which implies $\|\boldsymbol{W}^T\|_2 \leq \sqrt{n}$. Combining these results,

$$\|\boldsymbol{v}\|_2 \leq \|(\boldsymbol{S}\boldsymbol{X})^{1/2}\|_2\|\boldsymbol{W}^T\|_2\|\boldsymbol{Q}\boldsymbol{f}\|_2 \leq \sqrt{n\mu}\sqrt{n}\|\boldsymbol{f}\|_2 = n\sqrt{\mu}\|\boldsymbol{f}\|_2$$

as claimed. □

Note that compared to the corresponding result in Lemma 4 of [Cho+20] the bound $\|\boldsymbol{v}\|_2 \leq n\sqrt{\mu}\|\boldsymbol{f}\|_2$ is worse by a factor of $\sqrt{n}$, but the proof does not introduce any additional randomness and this change does not affect any asymptotic runtime bounds.

3.4. Computing an Approximate Newton Direction

The previous considerations give rise to Algorithm 2 that tries to determine an approximate Newton direction as in Equation (2.11) with $\|\boldsymbol{v}\|_2 \leq \varepsilon_{\boldsymbol{v}}$. The behaviour of this algorithm in the case that $\boldsymbol{W}$ is not a subspace embedding for $\boldsymbol{D}\boldsymbol{A}^T$ will not be analysed, may not be well-defined and will be referred to as a *failure case*. It is unclear whether there is an efficient way to recognize such a sketching failure but in any case choosing δ appropriately later ensures that this will be an exceptionally rare occurrence.

Algorithm 2: Approximate Newton direction

Input: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with rank m, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{x}, \boldsymbol{s} \in \mathbb{R}^n_{>0}$, $\boldsymbol{y} \in \mathbb{R}^m$, $\sigma \in (0,1)$, sketching tolerance $\varepsilon \in (0, \frac{1}{2})$, failure probability $\delta \in (0, \frac{1}{2})$, error tolerance $\varepsilon_{\boldsymbol{v}} > 0$

1 Compute $\boldsymbol{r}_p = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, $\boldsymbol{r}_d = \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c}$, $\boldsymbol{X} = \operatorname{diag}(\boldsymbol{x})$, $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{s})$;

2 Compute $\boldsymbol{p} = -\boldsymbol{r}_p + \boldsymbol{A}(-\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{x} - \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1})$;

3 Construct a random s-hashing matrix $\boldsymbol{W} \in \mathbb{R}^{w\times m}$ as in Theorem 3.5 for the given δ and ε;

4 Compute the QR decomposition $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{W}\boldsymbol{D}\boldsymbol{A}^T$;

5 Run the CG method with $\boldsymbol{q}^0 = \boldsymbol{0}$ until $\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}\boldsymbol{q} = \boldsymbol{R}^{-T}\boldsymbol{p} + \boldsymbol{f}$ for some error $\boldsymbol{f}$ satisfying $\|\boldsymbol{f}\|_2 \le \varepsilon_{\boldsymbol{v}}/(n\sqrt{\mu})$;

6 Compute $\boldsymbol{v} = (\boldsymbol{S}\boldsymbol{X})^{1/2}\boldsymbol{W}^T\boldsymbol{Q}\boldsymbol{f}$;

7 Compute $\hat{\Delta\boldsymbol{y}} = \boldsymbol{R}^{-1}\boldsymbol{q}$;

8 Compute $\hat{\Delta\boldsymbol{s}} = -\boldsymbol{r}_d - \boldsymbol{A}^T\hat{\Delta\boldsymbol{y}}$;

9 Compute $\hat{\Delta\boldsymbol{x}} = -\boldsymbol{x} + \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{S}^{-1}\boldsymbol{X}\hat{\Delta\boldsymbol{s}} - \boldsymbol{S}^{-1}\boldsymbol{v}$;

10 **return** $(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}})$;

Theorem 3.9. *Assume that $\gamma \in (0,1)$, $\sigma \in (0,1)$ and $\varepsilon \in (0, \frac{1}{2})$ are constant, that the initial point $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ satisfies $\boldsymbol{x}^0 \ge \boldsymbol{x}^*$ and $\boldsymbol{s}^0 \ge \max\{\boldsymbol{s}^*, |\boldsymbol{A}^T\boldsymbol{y}^0 - \boldsymbol{c}|\}$ for some $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$ and that $\boldsymbol{A}$ has full row rank. Let $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$ be such that $\boldsymbol{r} = \eta\boldsymbol{r}^0$ for some $\eta \in [0,1]$. Algorithm 2 computes an approximation of the Newton direction that satisfies Equation (2.11) with $\|\boldsymbol{v}\|_2 \le \varepsilon_{\boldsymbol{v}}$ with probability at least $1-\delta$. Moreover, if the algorithm does not fail, Line 5 only needs $O(\log(n^2\mu/\varepsilon_{\boldsymbol{v}}))$ CG iterations and the whole algorithm needs $O(\log(m/\delta)(\operatorname{nnz}(\boldsymbol{A}) + m^3) + \log(n^2\mu/\varepsilon_{\boldsymbol{v}})(\operatorname{nnz}(\boldsymbol{A}) + m^2))$ operations assuming $n \le \operatorname{nnz}(\boldsymbol{A})$.*

This theorem will be proven along with the convergence of Algorithm 1 in the next chapter.

4. Convergence Proofs

After the previous two chapters introduced the important ideas and algorithms, we need to establish their convergence properties. Note that Algorithm 1 is the same infeasible inexact long-step IPM as in [MO03] but with a different preconditioner. Thus, the following proofs are similar to those in their work adapted to the new preconditioner using the ideas of Chowdhury et al. [Cho+20]. The proofs in [Cho+20] were often schematic, especially the runtime considerations. The following section therefore tries to clearly lay out how the different lemmas imply Theorems 2.1 and 3.9 and how the runtimes for Algorithms 1 and 2 can be derived. The essential assumptions in the following proofs are that the algorithms work with exact arithmetic, that solutions of (P) and (D) exist and that $\boldsymbol{A}$ has full row rank.

4.1. Convergence Proof for the Infeasible IPM Algorithm

Let $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$ denote a possible iterate during the course of Algorithm 1, $(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}})$ the Newton direction determined in Line 4 and

$$(\boldsymbol{x}(\alpha), \boldsymbol{y}(\alpha), \boldsymbol{s}(\alpha)) \coloneqq (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) + \alpha(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}}) \tag{4.1}$$

$$\mu(\alpha) \coloneqq \boldsymbol{x}(\alpha)^T \boldsymbol{s}(\alpha) \tag{4.2}$$

$$\boldsymbol{r}(\alpha) \coloneqq (\boldsymbol{A}\boldsymbol{x}(\alpha) - \boldsymbol{b}, \boldsymbol{A}^T\boldsymbol{y}(\alpha) + \boldsymbol{s}(\alpha) - \boldsymbol{c}). \tag{4.3}$$

Using this notation Monteiro and O'Neal [MO03] showed that the step size $\bar{\alpha}$ determined in Algorithm 1 is bounded from below.

Lemma 4.1 (Lemma 3.6 in [MO03])**.** *Assume that*

- $\gamma \in (0, 1)$, $\sigma \in (0, \frac{4}{5})$,
- $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$ *and*
- $(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}})$ *satisfies Equation (2.11) such that* $\|\boldsymbol{v}\|_\infty \leq \gamma\sigma\mu/4$.

Then the step size $\bar{\alpha}$ *determined in Lines 5 and 6 satisfies*

$$\bar{\alpha} \geq \min\left\{ 1, \frac{\min\left\{ \gamma\sigma, 1 - \frac{5}{4}\sigma \right\}\mu}{4\|\hat{\Delta\boldsymbol{x}} \circ \hat{\Delta\boldsymbol{s}}\|_\infty} \right\}$$

and

$$\mu(\bar{\alpha}) \leq \left(1 - \left(1 - \frac{5}{4}\sigma\right)\frac{\bar{\alpha}}{2}\right)\mu.$$

Note that the assumptions in this lemma are not stated in this form in the original lemma but are instead inferred from the definition of their algorithm. In the proof they show that $\|\boldsymbol{v}\|_\infty \leq \gamma\sigma\mu/4$ and then deduce the results from it. It suffices now to upper-bound $\|\hat{\Delta}\boldsymbol{x} \circ \hat{\Delta}\boldsymbol{s}\|_\infty$ to show that μ decreases enough in each iteration. To this end, we need a key observation which motivated us to shift the error term in Section 2.3 using the perturbation vector $\boldsymbol{v}$: The form of the error term in Equation (2.12) ensures that at each iteration the residuals $\boldsymbol{r} = (\boldsymbol{r}_p, \boldsymbol{r}_d)$ lie on the line segment between $\boldsymbol{r}^0$ and $\mathbf{0}$ because $\boldsymbol{r}(\alpha) = (1-\alpha)\boldsymbol{r}(0)$. In other words, $\boldsymbol{r} = \eta\boldsymbol{r}^0$ with $\eta \in [0,1]$ for every iterate in Algorithm 1.

Lemma 4.2 (Lemma 16 in [Cho+20])**.** *Assume that*

- $\gamma \in (0,1)$, $\sigma \in (0,1)$,
- $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ *satisfies* $(\boldsymbol{x}^0, \boldsymbol{s}^0) \geq (\boldsymbol{x}^*, \boldsymbol{s}^*)$ *for some* $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$,
- $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$ *with* $\boldsymbol{r} = \eta\boldsymbol{r}^0$ *for some* $\eta \in [0,1]$ *and*
- $(\hat{\Delta}\boldsymbol{x}, \hat{\Delta}\boldsymbol{y}, \hat{\Delta}\boldsymbol{s})$ *satisfies Equation (2.11) such that* $\|\boldsymbol{v}\|_2 \leq \gamma\sigma\mu/4$.

Then

$$\|\boldsymbol{D}^{-1}\hat{\Delta}\boldsymbol{x}\|_2, \|\boldsymbol{D}\hat{\Delta}\boldsymbol{s}\|_2 \leq \left(1 + \frac{\sigma^2}{1-\gamma} - 2\sigma\right)^{1/2} \sqrt{n\mu} + \frac{6}{\sqrt{1-\gamma}} n\sqrt{\mu} + \frac{\gamma\sigma}{4\sqrt{1-\gamma}}\sqrt{\mu}$$

which implies that $\|\boldsymbol{D}^{-1}\hat{\Delta}\boldsymbol{x}\|_2$ *and* $\|\boldsymbol{D}\hat{\Delta}\boldsymbol{s}\|_2$ *are bounded by* $Cn\sqrt{\mu}$ *for some constant* $C > 0$ *depending on* γ *and* σ.

The proofs for these two lemmas can be found in the stated references and their combination is enough to show the convergence of Algorithm 1.

Proof of Theorem 2.1. By the choice of the initial point $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{N}_{-\infty}(\gamma)$ and $\bar{\alpha}$ in Lines 5 and 6, every iterate $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$ is inside the specified neighbourhood $\mathcal{N}_{-\infty}(\gamma)$. Therefore, $\mu^k \leq \varepsilon_\mu \mu^0$ implies $\|\boldsymbol{r}^k\| \leq \varepsilon_\mu \|\boldsymbol{r}^0\|$, and it suffices to show the former for some $k = O(n^2 \log(1/\varepsilon_\mu))$.

By Lemma 4.2 and the fact that $\boldsymbol{D}$ and $\boldsymbol{D}^{-1}$ are diagonal matrices the search direction determined in Line 4 satisfies

$$\|\hat{\Delta}\boldsymbol{x} \circ \hat{\Delta}\boldsymbol{s}\|_\infty \leq \|\hat{\Delta}\boldsymbol{x} \circ \hat{\Delta}\boldsymbol{s}\|_2 \leq \|\boldsymbol{D}^{-1}\hat{\Delta}\boldsymbol{x}\|_2 \|\boldsymbol{D}\hat{\Delta}\boldsymbol{s}\|_2 \leq C_1 n^2 \mu^k$$

in every iteration, such that by Lemma 4.1 ($\|\boldsymbol{v}\|_\infty \leq \|\boldsymbol{v}\|_2 \leq \gamma\sigma\mu^k/4$)

$$\bar{\alpha} \geq \min\left\{1, \frac{\min\left\{\gamma\sigma, 1 - \frac{5}{4}\sigma\right\}\mu^k}{4C_1 n^2 \mu^k}\right\} \geq \frac{C_2}{n^2}$$

where $C_1, C_2 > 0$ are constants depending only on γ and σ. Plugging this inequality in the second bound in Lemma 4.1 we get

$$\mu^{k+1} = \mu(\bar{\alpha}) \leq \left(1 - \left(1 - \frac{5}{4}\sigma\right)\frac{\bar{\alpha}}{2}\right)\mu^k \leq \left(1 - \left(1 - \frac{5}{4}\sigma\right)\frac{C_2}{2n^2}\right)\mu^k \leq \left(1 - \frac{C_3}{n^2}\right)\mu^k$$

for some constant $C_3 > 0$. By induction this means that $\mu^k \leq (1 - \frac{C_3}{n^2})^k \mu^0$ for all $k \geq 0$. Now, let $k \geq n^2 \log(1/\varepsilon_\mu)/C_3$ which implies

$$k \log\left(1 - \frac{C_3}{n^2}\right) \leq k\left(-\frac{C_3}{n^2}\right) \leq -\log(1/\varepsilon_\mu) = \log(\varepsilon_\mu)$$

using $\log(\beta) \leq \beta - 1$ for all $\beta > 0$. Applying the exponential function on both sides gives $(1 - \frac{C_3}{n^2})^k \leq \varepsilon_\mu$ which shows that $O(n^2 \log(1/\varepsilon_\mu))$ iterations suffice to get $\mu^k \leq \varepsilon_\mu \mu^0$. □

4.2. Convergence Proof for the Approximate Newton Direction Algorithm

Next, we turn to the correctness of Algorithm 2 which is concerned with calculating a sufficiently good approximation of the Newton direction. As linear convergence of the CG method was already established in Lemma 3.7, our main goal is to bound the norm of the initial residual $\boldsymbol{R}^{-T}\boldsymbol{p}$. To this end, we introduce Lemma 4.3 whose proof can already be found in [MO03].

Lemma 4.3 (Lemma 3.2 in [MO03], Lemma 11 in [Cho+20])**.** *Assume that*

- $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ *satisfies* $(\boldsymbol{x}^0, \boldsymbol{s}^0) \geq (\boldsymbol{x}^*, \boldsymbol{s}^*)$ *for some* $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$ *and*
- $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{G}$ *satisfies* $\boldsymbol{r} = \eta \boldsymbol{r}^0$ *for some* $\eta \in [0, 1]$.

Then $\eta \leq \boldsymbol{x}^T \boldsymbol{s} / \boldsymbol{x}^{0T} \boldsymbol{s}^0$ *implies* $\eta(\boldsymbol{x}^{0T} \boldsymbol{s} + \boldsymbol{x}^T \boldsymbol{s}^0) \leq 3n\mu$.

Again, this makes use of the observation that the residual $\boldsymbol{r}$ of any iterate lies on the line segment between $\boldsymbol{r}^0$ and $\boldsymbol{0}$. Note that additionally, by the definition of the neighbourhood $\mathcal{N}_{-\infty}(\gamma)$, any point in it with $\boldsymbol{r} = \eta \boldsymbol{r}^0$ also satisfies $\eta \leq \mu^k / \mu^0 = \boldsymbol{x}^{kT} \boldsymbol{s}^k / \boldsymbol{x}^{0T} \boldsymbol{s}^0$.

Lemma 4.4 (Lemma 12 in [Cho+20] (simplified))**.** *Assume that*

- $\gamma \in (0, 1)$, $\sigma \in (0, 1)$, $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$,
- $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ *satisfies* $\boldsymbol{x}^0 \geq \boldsymbol{x}^*$ *and* $\boldsymbol{s}^0 \geq \max\{\, \boldsymbol{s}^*, |\boldsymbol{A}^T \boldsymbol{y}^0 - \boldsymbol{c}| \,\}$
- $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}) \in \mathcal{N}_{-\infty}(\gamma)$ *satisfies* $\boldsymbol{r} = \eta \boldsymbol{r}^0$ *for some* $\eta \in [0, 1]$ *and*
- $\boldsymbol{R}^{-T} \in \mathbb{R}^{m \times m}$ *satisfies* $\|\boldsymbol{R}^{-T} \boldsymbol{A} \boldsymbol{D}\|_2 \leq \sqrt{2}$.

Then

$$\|\boldsymbol{R}^{-T} \boldsymbol{p}\|_2 \leq \sqrt{\frac{2\mu}{1 - \gamma}}\left(7n + \sigma\sqrt{n}\right) \leq 8\sqrt{\frac{2}{1 - \gamma}} n \sqrt{\mu}$$

where $\boldsymbol{p} = -\boldsymbol{r}_p + \boldsymbol{A}(-\boldsymbol{S}^{-1} \boldsymbol{X} \boldsymbol{r}_d + \boldsymbol{x} - \sigma\mu \boldsymbol{S}^{-1} \boldsymbol{1})$ *as in Equation (2.7a).*

Proof. First, we will deduce bounds for $\boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*)$ and $\boldsymbol{X}\boldsymbol{r}_d^0$ that we need later. Note that because $\boldsymbol{0} \leq \boldsymbol{x}^* \leq \boldsymbol{x}^0$ and $\boldsymbol{s} \geq \boldsymbol{0}$ we have $\boldsymbol{0} \leq \boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*) \leq \boldsymbol{S}\boldsymbol{x}^0$, so

$$\|\boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*)\|_2 \leq \|\boldsymbol{S}\boldsymbol{x}^0\|_2 \leq \|\boldsymbol{S}\boldsymbol{x}^0\|_1 = \boldsymbol{s}^T\boldsymbol{x}^0.$$

Using a similar argument as above, $\boldsymbol{0} \leq |\boldsymbol{A}^T\boldsymbol{y}^0 - \boldsymbol{c}| \leq \boldsymbol{s}^0$ implies $\|\boldsymbol{X}|\boldsymbol{A}^T\boldsymbol{y}^0 - \boldsymbol{c}|\|_2 \leq \|\boldsymbol{X}\boldsymbol{s}^0\|_2$ and therefore

$$\begin{aligned}\|\boldsymbol{X}\boldsymbol{r}_d^0\|_2 = \|\boldsymbol{X}(\boldsymbol{A}^T\boldsymbol{y}^0 + \boldsymbol{s}^0 - \boldsymbol{c})\|_2 &\leq \|\boldsymbol{X}\boldsymbol{s}^0\|_2 + \|\boldsymbol{X}|\boldsymbol{A}^T\boldsymbol{y}^0 - \boldsymbol{c}|\|_2 \\ &\leq 2\|\boldsymbol{X}\boldsymbol{s}^0\|_2 \leq 2\|\boldsymbol{X}\boldsymbol{s}^0\|_1 = 2\boldsymbol{x}^T\boldsymbol{s}^0.\end{aligned}$$

Now, consider the expression in the claim. By the definition of p in Equation (2.7a) and the assumption $\boldsymbol{r} = \eta\boldsymbol{r}^0$, we get

$$\begin{aligned}\boldsymbol{R}^{-T}\boldsymbol{p} &= \boldsymbol{R}^{-T}\Big(-\boldsymbol{r}_p + \boldsymbol{A}(-\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{x} - \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1})\Big) \\ &= \boldsymbol{R}^{-T}\Big(-\boldsymbol{r}_p - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{A}\boldsymbol{x} - \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1}\Big) \\ &= \boldsymbol{R}^{-T}\Big(-\eta\boldsymbol{r}_p^0 - \eta\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d^0 + \boldsymbol{A}\boldsymbol{x} - \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1}\Big) \\ &= \boldsymbol{R}^{-T}\Big(-\eta\boldsymbol{A}(\boldsymbol{x}^0 - \boldsymbol{x}^*) - \eta\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d^0 + \boldsymbol{A}\boldsymbol{x} - \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1}\Big) \\ &= \boldsymbol{R}^{-T}\boldsymbol{A}\Big(-\eta(\boldsymbol{x}^0 - \boldsymbol{x}^*) - \eta\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d^0 + \boldsymbol{x} - \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1}\Big) \\ &= \boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{S}^{-1}\Big(-\eta\boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*) - \eta\boldsymbol{X}\boldsymbol{r}_d^0 + \boldsymbol{S}\boldsymbol{x} - \sigma\mu\boldsymbol{1}\Big) \\ &= \boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}(\boldsymbol{S}\boldsymbol{X})^{-1/2}\Big(-\eta\boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*) - \eta\boldsymbol{X}\boldsymbol{r}_d^0 + \boldsymbol{S}\boldsymbol{x} - \sigma\mu\boldsymbol{1}\Big).\end{aligned}$$

Because of that, it suffices to bound each part of the last expression: $\|\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}\|_2 \leq \sqrt{2}$ by assumption,

$$\|(\boldsymbol{S}\boldsymbol{X})^{-1/2}\|_2 = \max_{1\leq i\leq n}(x_i s_i)^{-1/2} \leq \sqrt{\frac{1}{(1-\gamma)\mu}}$$

by the definition of the neighbourhood and $\|\boldsymbol{S}\boldsymbol{x}\|_2 \leq \|\boldsymbol{S}\boldsymbol{x}\|_1 = n\mu$ using the definition of μ. This implies

$$\begin{aligned}\|\boldsymbol{R}^{-T}\boldsymbol{p}\|_2 &\leq \|\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}\|_2\|(\boldsymbol{S}\boldsymbol{X})^{-1/2}\|_2\Big(\eta\|\boldsymbol{S}(\boldsymbol{x}^0 - \boldsymbol{x}^*)\| + \eta\|\boldsymbol{X}\boldsymbol{r}_d^0\| + \|\boldsymbol{S}\boldsymbol{x}\|_2 + \sigma\mu\|\boldsymbol{1}\|_2\Big) \\ &\leq \sqrt{2}\sqrt{\frac{1}{(1-\gamma)\mu}}\Big(\eta\boldsymbol{s}^T\boldsymbol{x}^0 + 2\eta\boldsymbol{x}^T\boldsymbol{s}^0 + n\mu + \sigma\mu\sqrt{n}\Big) \\ &\leq \sqrt{2}\sqrt{\frac{1}{(1-\gamma)\mu}}\Big(6n\mu + n\mu + \sigma\mu\sqrt{n}\Big) = \sqrt{\frac{2\mu}{1-\gamma}}\Big(7n + \sigma\sqrt{n}\Big)\end{aligned}$$

with the help of Lemma 4.3. □

Equipped with this result, we can prove the main theorem regarding Algorithm 2.

Proof of Theorem 3.9. By Theorem 3.5 the matrix $\boldsymbol{W}$ selected in Line 3 of Algorithm 2 is a subspace embedding for $\boldsymbol{D}\boldsymbol{A}^T$ with probability at least $1-\delta$, so it suffices to consider

this case exclusively and from now on assume that $\boldsymbol{W}$ is indeed a $(1 \pm \varepsilon)$ ℓ_2-subspace embedding for $\boldsymbol{D}\boldsymbol{A}^T$. In this case the QR decomposition $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{W}\boldsymbol{D}\boldsymbol{A}^T$ exists and by Lemma 3.6 we have

$$\|\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}\|_2 \leq \sqrt{\frac{1}{1-\varepsilon}} \leq \sqrt{2}$$

using $\varepsilon < \frac{1}{2}$. Let $\boldsymbol{q}^j$ be the iterates of the CG algorithm in Line 5 and let $\boldsymbol{f}^j$ be the residuals defined by $\boldsymbol{f}^j = \boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}\boldsymbol{R}^{-1}\boldsymbol{q}^j - \boldsymbol{R}^{-T}\boldsymbol{p}$. Using Lemma 4.4 and $\boldsymbol{q}^0 = \boldsymbol{0}$ we have $\|\boldsymbol{f}^0\|_2 = \|\boldsymbol{R}^{-T}\boldsymbol{p}\|_2 \leq Cn\sqrt{\mu}$ for some constant $C > 0$ depending on γ. Lemma 3.7 immediately implies that $\|\boldsymbol{f}^j\|_2$ converges to zero and so Line 5 will terminate. Moreover, it tells us that for $j \geq \log_{\frac{1-\varepsilon}{1+\varepsilon}}(\varepsilon_{\boldsymbol{v}}/Cn^2\mu) = \log_{\frac{1+\varepsilon}{1-\varepsilon}}(Cn^2\mu/\varepsilon_{\boldsymbol{v}})$ we have

$$\|\boldsymbol{f}^j\|_2 \leq \left(\frac{1-\varepsilon}{1+\varepsilon}\right)^j \|\boldsymbol{f}^0\|_2 \leq \frac{\varepsilon_{\boldsymbol{v}}}{Cn^2\mu} Cn\sqrt{\mu} \leq \frac{\varepsilon_{\boldsymbol{v}}}{n\sqrt{\mu}}$$

so Line 5 will terminate after $O(\log(n^2\mu/\varepsilon_{\boldsymbol{v}}))$ CG iterations. By Lemma 3.8 and the definition of $\boldsymbol{v}$ in Line 6 $\|\boldsymbol{f}^j\|_2 \leq \varepsilon_{\boldsymbol{v}}/n\sqrt{\mu}$ implies $\|\boldsymbol{v}\|_2 \leq \varepsilon_{\boldsymbol{v}}$. Lines 7 to 9 compute $(\hat{\Delta}\boldsymbol{x}, \hat{\Delta}\boldsymbol{y}, \hat{\Delta}\boldsymbol{s})$ which satisfy Equation (2.10) for $\tilde{\boldsymbol{f}} = \boldsymbol{R}^T\boldsymbol{f}$. Using Lemma 3.8 again we can see that

$$\begin{aligned}
\boldsymbol{A}\hat{\Delta}\boldsymbol{x} &= \boldsymbol{A}(-\boldsymbol{x} + \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{S}^{-1}\boldsymbol{X}\hat{\Delta}\boldsymbol{s} - \boldsymbol{S}^{-1}\boldsymbol{v}) \\
&= -\boldsymbol{A}\boldsymbol{x} + \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\hat{\Delta}\boldsymbol{s} - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} \\
&= -\boldsymbol{A}\boldsymbol{x} + \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1} + \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} \\
&= -\boldsymbol{A}\boldsymbol{x} + \sigma\mu\boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{1} + \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{X}\boldsymbol{r}_d + \boldsymbol{p} + \boldsymbol{R}^T\boldsymbol{f} - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} \\
&= -\boldsymbol{r}_p + \boldsymbol{R}^T\boldsymbol{f} - \boldsymbol{A}\boldsymbol{S}^{-1}\boldsymbol{v} \\
&= -\boldsymbol{r}_p \\
\boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} + \hat{\Delta}\boldsymbol{s} &= \boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} - \boldsymbol{r}_d - \boldsymbol{A}^T\hat{\Delta}\boldsymbol{y} \\
&= -\boldsymbol{r}_d \\
\boldsymbol{S}\hat{\Delta}\boldsymbol{x} + \boldsymbol{X}\hat{\Delta}\boldsymbol{s} &= \boldsymbol{S}(-\boldsymbol{x} + \sigma\mu\boldsymbol{S}^{-1}\boldsymbol{1} - \boldsymbol{S}^{-1}\boldsymbol{X}\hat{\Delta}\boldsymbol{s} - \boldsymbol{S}^{-1}\boldsymbol{v}) + \boldsymbol{X}\hat{\Delta}\boldsymbol{s} \\
&= -\boldsymbol{S}\boldsymbol{x} + \sigma\mu\boldsymbol{1} - \boldsymbol{X}\hat{\Delta}\boldsymbol{s} - \boldsymbol{v} + \boldsymbol{X}\hat{\Delta}\boldsymbol{s} \\
&= -\boldsymbol{x} \circ \boldsymbol{s} + \sigma\mu\boldsymbol{1} - \boldsymbol{v}
\end{aligned}$$

which shows that the determined search direction indeed satisfies Equation (2.11) with $\|\boldsymbol{v}\|_2 \leq \varepsilon_{\boldsymbol{v}}$.

After showing that the algorithm works correctly, we need to bound the running time. Multiplication with $\boldsymbol{X}, \boldsymbol{X}^{1/2}, \boldsymbol{S}, \boldsymbol{S}^{1/2}$ and their inverses can be done in $O(n)$, multiplication with $\boldsymbol{A}$ and $\boldsymbol{A}^T$ in $O(\operatorname{nnz}(\boldsymbol{A}))$ and multiplication with $\boldsymbol{R}^{-1}$ and $\boldsymbol{R}^{-T}$ in $O(m^2)$ using triangular solves. Because $m \leq n \leq \operatorname{nnz}(\boldsymbol{A})$, we can thus conclude that Lines 1, 2 and 7 to 9 run in $O(\operatorname{nnz}(\boldsymbol{A}) + m^2)$ operations. Constructing the random s-hashing matrix in Line 3 requires $O(ns)$ time as it is proportional to the number of nonzero entries and computing $\boldsymbol{W}\boldsymbol{D}\boldsymbol{A}^T$ in Line 4 can be done in $O(s \operatorname{nnz}(\boldsymbol{A}))$. Computing the

QR decomposition of a $w \times m$ matrix takes $O(wm^2)$ operations and so the preconditioner $\boldsymbol{R}$ is determined in $O(ns + wm^2)$. The number of CG iterations in Line 5 were already bounded by $O(\log(n^2\mu/\varepsilon_{\boldsymbol{v}}))$ and the runtime of each iteration is dominated by a matrix-vector product involving $\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}$ which takes $O(\operatorname{nnz}(\boldsymbol{A}) + m^2)$ flops. Lastly, the perturbation vector $\boldsymbol{v}$ is calculated in Line 6 in $O(ns + m^2)$ time. Combining these upper bounds gives

$$O\Big(\operatorname{nnz}(\boldsymbol{A}) + m^2 + ns + wm^2 + \log\Big(n^2\mu/\varepsilon_{\boldsymbol{v}}\Big)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^2\Big)\Big)$$

operations which simplifies to

$$O\Big(\log(m/\delta)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^3\Big) + \log\Big(n^2\mu/\varepsilon_{\boldsymbol{v}}\Big)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^2\Big)\Big)$$

operations because w and s were chosen according to Theorem 3.5. □

4.3. Runtime Bounds for the Combined Algorithm

Theorems 2.1 and 3.9 were the stepping stones towards the final result of this chapter. The combination of Algorithms 1 and 2 is an efficient algorithm for solving linear programs where $\boldsymbol{A}$ is sparse and $m \ll n$:

Theorem 4.5. *Assume that $\gamma \in (0,1)$, $\sigma \in (0, \frac{4}{5})$, $\varepsilon \in (0, \frac{1}{2})$ and $\varepsilon_\mu \in (0,1)$ are constant, that the initial point $(\boldsymbol{x}^0, \boldsymbol{y}^0, \boldsymbol{s}^0) \in \mathcal{G}$ satisfies $\boldsymbol{x}^0 \circ \boldsymbol{s}^0 \geq (1-\gamma)\mathbf{1}$, $\boldsymbol{x}^0 \geq \boldsymbol{x}^*$ and $\boldsymbol{s}^0 \geq \max\{\boldsymbol{s}^*, |\boldsymbol{A}^T\boldsymbol{y}^0 - \boldsymbol{c}|\}$ for some $(\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{s}^*) \in \mathcal{F}^*$ and that $\boldsymbol{A}$ has full row rank. Furthermore, assume that Algorithm 1 uses Algorithm 2 with $\varepsilon_{\boldsymbol{v}} = \gamma\sigma\mu^k/4$ to determine the approximate Newton direction in Lines 3 and 4. Then it is possible to choose δ in such a way that the algorithm generates an iterate $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$ satisfying $\mu^k \leq \varepsilon_\mu\mu^0$ and $\|\boldsymbol{r}^k\| \leq \varepsilon_\mu\|\boldsymbol{r}^0\|$ using $O(n^2\log(n)(\operatorname{nnz}(\boldsymbol{A})+m^3))$ operations with probability at least 99%.*

Proof. The assumptions of this theorem and the invariants $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k) \in \mathcal{N}_{-\infty}(\gamma)$ and $\boldsymbol{r}^k = \eta\boldsymbol{r}^0$ for some $\eta \in [0,1]$ provided by Algorithm 1 provide the assumptions of Theorem 2.1 and Theorem 3.9, so the combined algorithm will generate an iterate $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$ satisfying $\mu^k \leq \varepsilon_\mu\mu^0$ and $\|\boldsymbol{r}^k\| \leq \varepsilon_\mu\|\boldsymbol{r}^0\|$ after $O(n^2\log(1/e_\mu))$ IPM iterations as long as Algorithm 2 does not fail in any iteration. To simplify the runtime bounds we assume that ε_μ is constant, so $O(n^2\log(1/\varepsilon_\mu)) = O(n^2)$.

The probability of zero sketching failures over k IPM iterations is at least $(1-\delta)^k$ because the random sketching matrices are independently drawn for each iteration. Let $\delta_{\text{IPM}} = 1\%$ be a bound for the overall failure probability, let the number of IPM iterations be bounded by Cn^2 for some constant $C > 0$ and define

$$\delta = -\frac{\log(1 - \delta_{\text{IPM}}/2)}{Cn^2} = O(n^{-2}).$$

Because

$$\lim_{n\to\infty}(1-\delta)^{Cn^2} = \lim_{n\to\infty}\left(1 + \frac{\log(1-\delta_{\text{IPM}}/2)}{Cn^2}\right)^{Cn^2} = 1 - \delta_{\text{IPM}}/2 > 1 - \delta_{\text{IPM}}$$

we can conclude that $(1-\delta)^{Cn^2} \geq 1 - \delta_{\text{IPM}}$ is satisfied for n large enough. In other words, for large enough instances an arbitrarily low fixed failure probability $\delta_{\text{IPM}} > 0$ can be guaranteed for some $\delta = O(n^{-2})$.

With this choice of δ and $\varepsilon_{\boldsymbol{v}} = \gamma\sigma\mu/4$ Lines 3 and 4 of Algorithm 1 (which use Algorithm 2) need

$$\begin{aligned}
&O\Big(\log(m/\delta)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^3\Big) + \log\Big(n^2\mu/\varepsilon_{\boldsymbol{v}}\Big)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^2\Big)\Big)\\
&= O\Big(\log\Big(mn^2\Big)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^3\Big) + \log\Big(n^2\Big)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^2\Big)\Big)\\
&= O\Big(\log(n)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^3\Big)\Big)
\end{aligned}$$

operations per iteration by Theorem 3.9.

To show that the remaining operations in Lines 5 to 8 do not change the asymptotic runtime bound for each IPM iteration, first consider

$$\begin{aligned}
(\boldsymbol{x}^k(\alpha), \boldsymbol{y}^k(\alpha), \boldsymbol{s}^k(\alpha)) &\coloneqq (\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k) + \alpha(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}})\\
\mu^k(\alpha) &\coloneqq \boldsymbol{x}^k(\alpha)^T \boldsymbol{s}^k(\alpha)\\
\boldsymbol{r}^k(\alpha) &\coloneqq (\boldsymbol{A}\boldsymbol{x}^k(\alpha) - \boldsymbol{b}, \boldsymbol{A}^T\boldsymbol{y}^k(\alpha) + \boldsymbol{s}^k(\alpha) - \boldsymbol{c}).
\end{aligned}$$

and their dependence on α. As $\boldsymbol{x}^k(\alpha)$, $\boldsymbol{y}^k(\alpha)$ and $\boldsymbol{s}^k(\alpha)$ are linear in α, $\mu^k(\alpha)$ and $\boldsymbol{x}^k(\alpha) \circ \boldsymbol{s}^k(\alpha)$ are quadratic functions in alpha in each component. Because the search direction satisfies Equation (2.11), $\boldsymbol{r}^k(\alpha) = (1-\alpha)\boldsymbol{r}^k(0)$ and so $\|\boldsymbol{r}^k(\alpha)\|_2/\|\boldsymbol{r}^0\|_2$ is linear in α. Therefore, all terms of interest are either linear or quadratic in α, and we will assume that all polynomial coefficients for the quadratic functions are determined at the start which can be done in $O(n)$.

We begin in Line 5 in which the maximum $\tilde{\alpha} \in [0,1]$ is determined such that for all $\alpha \in [0, \tilde{\alpha}]$ we have $(\boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k) + \alpha(\hat{\Delta\boldsymbol{x}}, \hat{\Delta\boldsymbol{y}}, \hat{\Delta\boldsymbol{s}}) \in \mathcal{N}_{-\infty}(\gamma)$. Note that $\tilde{\alpha}$ is the minimum $\alpha \in [0,1]$ such that

$$x_i^k(\alpha)s_i^k(\alpha) - (1-\gamma)\mu^k(\alpha) = 0 \text{ for some } 1 \leq i \leq n \quad \text{or} \quad \frac{\|\boldsymbol{r}^k(\alpha)\|_2}{\|\boldsymbol{r}^0\|_2} - \frac{\mu^k(\alpha)}{\mu^0} = 0$$

because $(\boldsymbol{x}^k(0), \boldsymbol{y}^k(0), \boldsymbol{s}^k(0)) \in \mathcal{N}_{-\infty}(\gamma)$. Using this characterisation $\tilde{\alpha}$ can be determined in $O(n)$ by solving $n+1$ quadratic equations.

Next, computing $\bar{\alpha}$ in Line 6 is equivalent to minimizing the quadratic function $\mu(\alpha)$ in $[0, \tilde{\alpha}]$. Computing the stationary point of the quadratic function $\mu(\alpha)$ takes $O(1)$ and determining whether the minimum is attained at the boundary of $[0, \tilde{\alpha}]$ or at the stationary point can also be done in $O(1)$.

Lastly, Line 7 takes $O(n)$ and Line 8 $O(1)$ operations. We can conclude that all computations outside Lines 3 and 4 can be completed in $O(n)$ and therefore the worst-case complexity per iteration is still bounded by $O(\log(n)(\operatorname{nnz}(\boldsymbol{A}) + m^3))$. Because $O(n^2)$ iterations need to be performed the total complexity is

$$O\Big(n^2\log(n)\Big(\operatorname{nnz}(\boldsymbol{A}) + m^3\Big)\Big)$$

as claimed. □

5. Experiments

In this chapter we evaluate the preconditioning technique introduced in Section 3.2 inside the implementation of the IPM algorithm of Andersen and Andersen [AA00] provided by the open source Python-based scientific computing library `scipy` [Vir+20].

5.1. The Homogeneous Algorithm

The interior-point method implemented in `scipy` is based on the MOSEK interior-point optimizer as published by Andersen and Andersen [AA00]. The most important difference to the general infeasible IPM described in Algorithm 1 is its use of the homogeneous model presented in [XHY96]. While in Chapter 2 the original LP (P) was solved by considering the primal and dual LP at the same time and adding a complementary condition, the homogeneous algorithm replaces this primal-dual formulation by the self-dual LP

$$\begin{aligned}
\boldsymbol{r}_p &\coloneqq \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\tau = \boldsymbol{0} && \text{(5.1a)}\\
\boldsymbol{r}_d &\coloneqq \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s} - \boldsymbol{c}\tau = \boldsymbol{0} && \text{(5.1b)}\\
\boldsymbol{r}_g &\coloneqq -\boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{y} - \kappa = 0 && \text{(5.1c)}\\
\boldsymbol{x}, \boldsymbol{s} &\geq \boldsymbol{0} && \text{(5.1d)}\\
\tau, \kappa &\geq 0 && \text{(5.1e)}
\end{aligned}$$

where τ and κ are additional variables. Observe that setting $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{s}$, τ and κ to zero gives a solution, so the LP above is always feasible. The crucial advantage of using this homogeneous model is that there is always a solution where $(\boldsymbol{x}, \tau)$ and $(\boldsymbol{s}, \kappa)$ are strictly complementary and every such solution can be transformed into either a solution of the original LP (if $\tau > 0$) or a certificate that the original LP is infeasible or unbounded (if $\kappa > 0$). In other words, by adding two additional variables the interior-point method becomes able to handle infeasible and unbounded LPs.

The Newton system that would correspond to Equation (2.5) is now given by

$$\begin{pmatrix}
\boldsymbol{A} & -\boldsymbol{b} & & & \\
 & -\boldsymbol{c} & \boldsymbol{A}^T & \boldsymbol{I} & \\
-\boldsymbol{c}^T & & \boldsymbol{b}^T & & -1 \\
\boldsymbol{S} & & & \boldsymbol{X} & \\
 & \kappa & & & \tau
\end{pmatrix}
\begin{pmatrix}
\Delta\boldsymbol{x} \\ \Delta\tau \\ \Delta\boldsymbol{y} \\ \Delta\boldsymbol{s} \\ \Delta\kappa
\end{pmatrix}
=
\begin{pmatrix}
-\boldsymbol{r}_p \\ -\boldsymbol{r}_d \\ -\boldsymbol{r}_g \\ -\boldsymbol{x} \circ \boldsymbol{s} + \sigma\mu\boldsymbol{1} \\ -\tau\kappa + \sigma\mu
\end{pmatrix} \tag{5.2}$$

where $\mu = (\boldsymbol{x}^T\boldsymbol{s} + \tau\kappa)/(n+1)$. In the implementation the right-hand side is modified to balance out the ambitiousness of reducing the infeasibility and the complementarity gap

using heuristics. Optionally, one can also enable an adaptation of Mehrotra's predictor-corrector approach first proposed in [Meh92] which first solves Equation (5.2) for $\sigma = 0$ and then solves the same system with a modified right-hand side to determine the search direction. To solve Equation (5.2) for arbitrary right-hand sides it is transformed into two linear systems $\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{u} = \boldsymbol{p}$ for different vectors $\boldsymbol{p}$ as detailed in section 5 of [AA00]. This allows us to use the sketching-based preconditioning explained in Section 3.2 inside this IPM implementation. Moreover, because the two linear systems involve the same matrix the preconditioner only needs to be determined once for both systems.

Some more differences between Algorithm 1 and this implementation need to be addressed. First, the neighbourhood used in this algorithm drops both the one-sided ∞-norm bound and the bound on the residual norms:

$$\mathcal{N} = \{\, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}, \tau, \kappa \mid \boldsymbol{x}, \boldsymbol{s} \geq \boldsymbol{0},\ \tau, \kappa \geq 0 \,\} \tag{5.3}$$

Second, the step size α is essentially chosen to be the largest possible step inside the neighbourhood without any line search minimizing μ. Third, the stopping criterion used is based on

$$\rho_p \coloneqq \frac{\|\boldsymbol{r}_p\|_2}{\max(1, \|\boldsymbol{r}_p^0\|_2)},\ \rho_d \coloneqq \frac{\|\boldsymbol{r}_d\|_2}{\max(1, \|\boldsymbol{r}_d^0\|_2)},\ \rho_A \coloneqq \frac{|\boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{y}|}{\tau + |\boldsymbol{b}^T\boldsymbol{y}|} \tag{5.4}$$

measuring the size of the primal and dual residuals and of the duality gap. Let $\rho_{\text{tol}} = \max(\rho_p, \rho_d, \rho_A)$ then the algorithm stops once ρ_{tol} drops below a user-specified tolerance or when infeasibility or unboundedness is detected. Lastly, the starting point of the algorithm is chosen to be $(\boldsymbol{x}, \tau, \boldsymbol{y}, \boldsymbol{s}, \kappa) = (\boldsymbol{1}, 1, \boldsymbol{0}, \boldsymbol{1}, 1)$ as Andersen and Andersen [AA00] report that it works well for most problems.

5.2. Problem Instances

To evaluate the performance of the IPM, random LP instances

$$\min \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0} \tag{P}$$

with $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$ were generated that fit all the desired characteristics: $m \ll n$, $\boldsymbol{A}$ is sparse, $\operatorname{rank}(\boldsymbol{A}) = m$ and the LP is feasible and bounded. To that end, we chose $m = 1\,000$, $n = 100\,000$ and constructed $\boldsymbol{A}$ by randomly choosing two nonzero entries per column which were filled with values drawn from a standard normal distribution and also adding normally distributed values on the diagonal. Next, a random feasible point $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ was generated where the entries of $\boldsymbol{x}$ and $\boldsymbol{s}$ where drawn from a uniform distribution on $[0, 1]$ and the entries of $\boldsymbol{y}$ from a standard normal distribution. Setting $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{c} = \boldsymbol{A}^T\boldsymbol{y} + \boldsymbol{s}$ ensures feasibility and boundedness of (P).

5.3. Experiment Configuration

The experiments were all run on an Intel® Core™ i7-1165G7 using `scipy` version 1.6.2 and `numpy` version 1.20.2 with OpenBLAS version 0.3.8. The code for the homogen-

eous IPM algorithm described above was taken from the `scipy.optimize` package[1] and adapted to incorporate iterative solvers and sketching-based preconditioning. To avoid inconsistent results the routines for eliminating unnecessary constraints and automatically rescaling the instances were disabled as were the use of the predictor-corrector method and the initial point heuristic. Each parameter combination in the following experiments was run twice on the same four randomly generated instances. To visualize the range of recorded values the following graphs all include error bars. The height of the line or bar indicates the median value while the error bars indicate the range from the 5th percentile to the 95th percentile.

5.4. Impact of the Sketching Parameters

The first question we want to tackle is how good the preconditioners are, how fast they can be computed and how the answers to these questions are influenced by the sketching parameters w and s as defined in Definition 3.4. Remember that w controls the number of columns of the sparse sketching matrix, while s determines the number of nonzero entries per column. By Theorem 3.5 we expect larger values of w and s to give better preconditioners while also increasing the runtime of computing $\boldsymbol{WDA}^T$ and its QR decomposition.

Figure 5.1 shows how the condition number of the condition numbers of the unpreconditioned matrix $\boldsymbol{AD}^2\boldsymbol{A}^T$ and the preconditioned matrix $\boldsymbol{R}^{-T}\boldsymbol{AD}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}$ evolves during a run of the IPM algorithm. Here, the sketching matrix is constructed using $w = 2m$ and $s \in \{2, 3, 4, 5\}$. While the condition number of the unpreconditioned system steadily rises up to about 10^{12}, the values for the preconditioned systems look much better. For all cases except $s = 2$ the condition numbers stay below 100. Figure 5.2 shows those condition numbers for all combinations of $s \in \{3, 4, 5\}$ and $w \in \{1.5m, 2m, 2.5m, 3m\}$ and includes values from all iterations. As predicted by the theory the condition numbers are effectively bounded by a constant which decreases as w and s increase. Interestingly, the lower end of the error bars line up for every fixed value of w, suggesting that the value of w is the decisive factor for how low the condition number can become while s mainly influences the range of the distribution.

Next, Figure 5.3 shows the sparsity of $\boldsymbol{WDA}^T$ by plotting the number of nonzero entries of the matrix. While we will see that in this work the impact of the sparsity on the running time is negligible, for other applications the sparsity preservation that is achieved by sketching with a sparse matrix may be of great importance. The red line indicates the number of nonzero entries of $\boldsymbol{A}$ itself while all other colours bear the same meaning as in Figure 5.2. A simple upper bound to the number of nonzero entries is given by $\operatorname{nnz}(\boldsymbol{WDA}^T) \leq s \operatorname{nnz}(\boldsymbol{A})$. Even though in reality the number of nonzero entries is lower, the graph reflects that s is the main influence on the number of nonzero entries while the value of w only plays a minor role. Interestingly, for example for $w = 2m$ about 25% to 40% of the entries of the sketched matrix are nonzero while the same value for $\boldsymbol{A}$ is at around 0.2%. This means that the absolute number of nonzero

[1] https://github.com/scipy/scipy/blob/v1.6.2/scipy/optimize/_linprog_ip.py

Figure 5.1.: Condition numbers as the IPM algorithm progresses

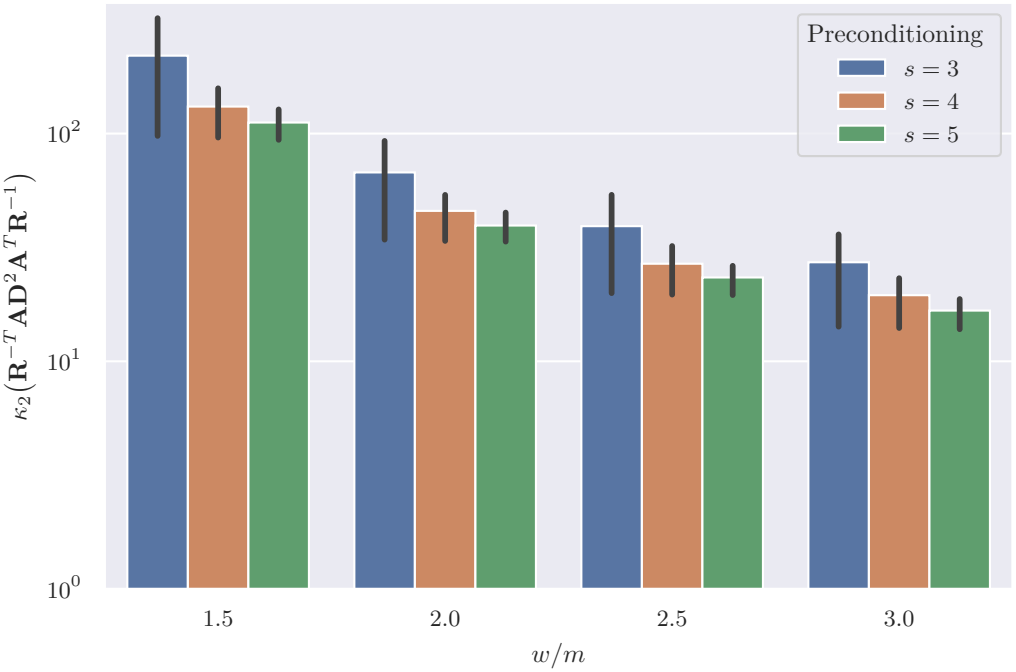

Figure 5.2.: Condition numbers depending on w and s

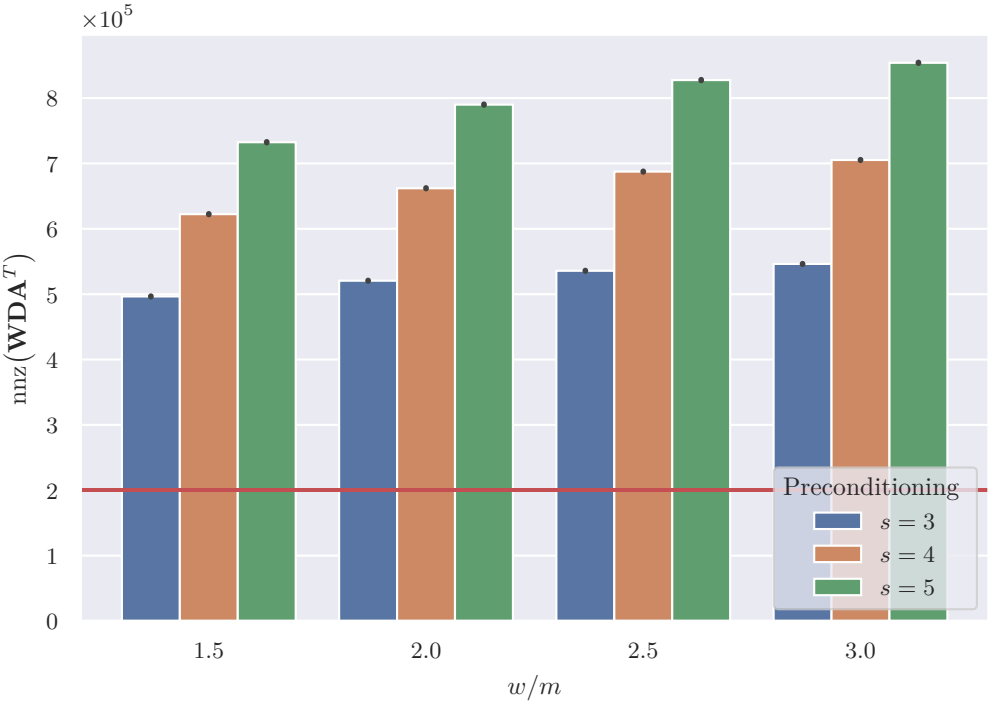

Figure 5.3.: Sparsity of $\boldsymbol{WDA}^T$ depending on w and s

entries is only increased by a small factor, yet the relative density is increased to the point where the matrix is not very sparse anymore.

Figures 5.4 to 5.6 visualize the time spent in the different steps of sketching. First, consider generating the sketching matrix $\boldsymbol{W}$. It turns out, that randomly sampling the row indices from $\{1, \dots, m\}$ for all columns without replacement is much faster than sampling them with replacement for each column separately, so it was beneficial to generate slightly more row indices than needed and then discard those instance where collisions occurred. This deletion process might explain the low dependence on w in Figure 5.4 but in any case the time is negligibly small for our purposes. The time it takes to multiply the sketching matrix with $\boldsymbol{DA}^T$ is similarly small as shown by Figure 5.5 and mainly depends on the value of s as predicted by the theory. The majority of the sketching process is actually spent determining a QR decomposition of $\boldsymbol{WDA}^T$, see Figure 5.6. Because the available sparse QR implementation did not bring any performance gains, a QR decomposition algorithm for dense matrices was used whose runtime, unsurprisingly, mainly depends on the matrix dimensions, i.e. on w. As such the runtime of determining a preconditioner depends mainly on w, so we chose a moderate value of $w = 2m$ and a large value of $s = 5$ for the following experiments.

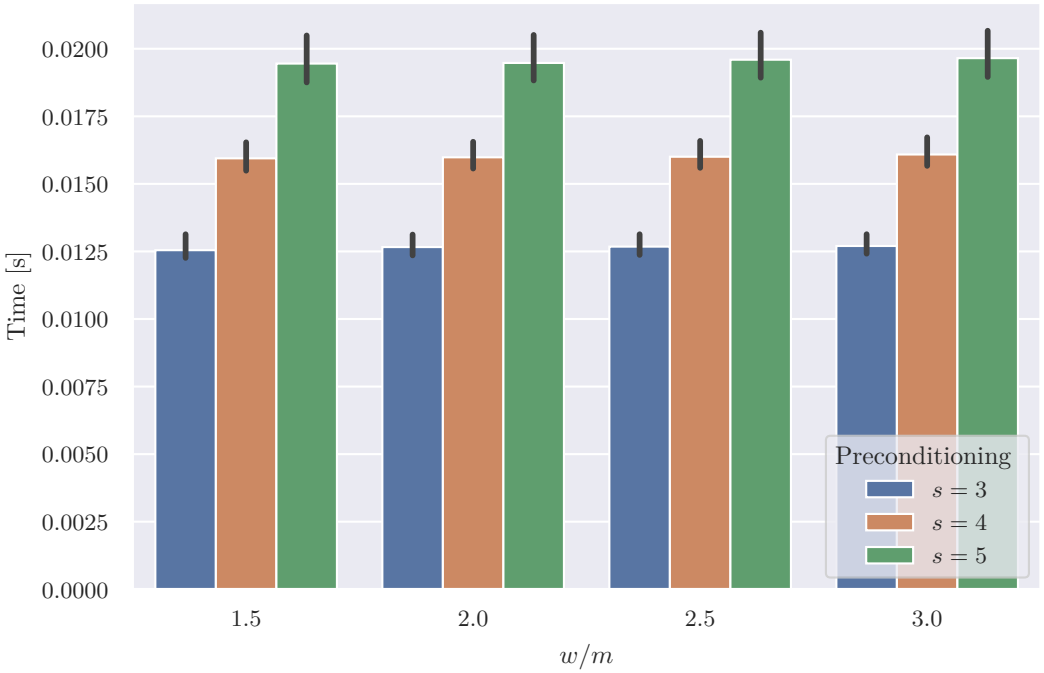

Figure 5.4.: Time spent generating the s-hashing matrix $\boldsymbol{W}$

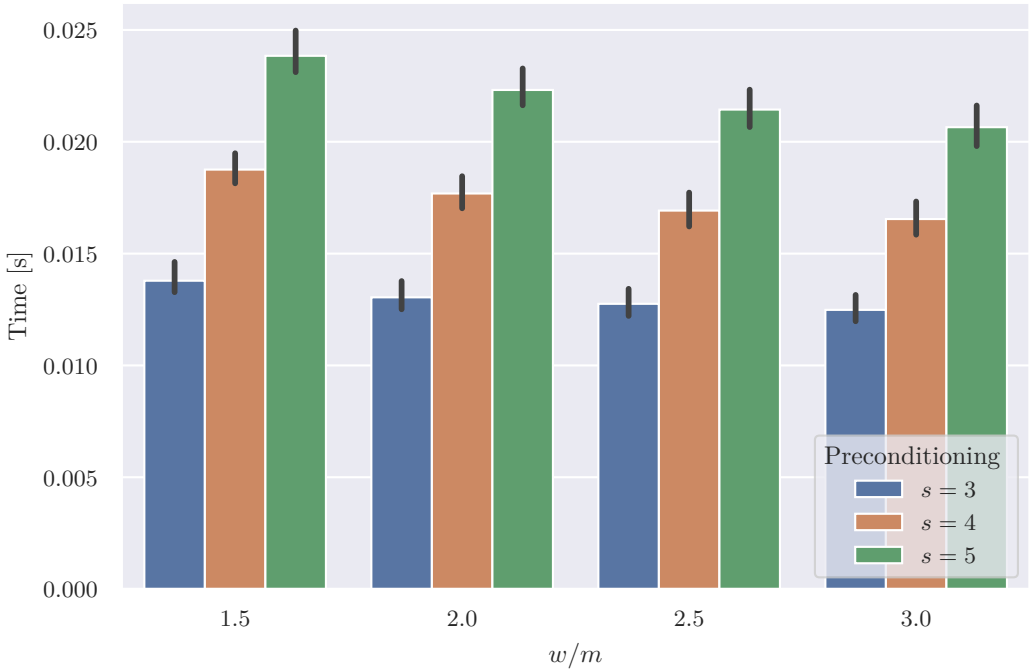

Figure 5.5.: Time spent computing the product $\boldsymbol{WDA}^T$

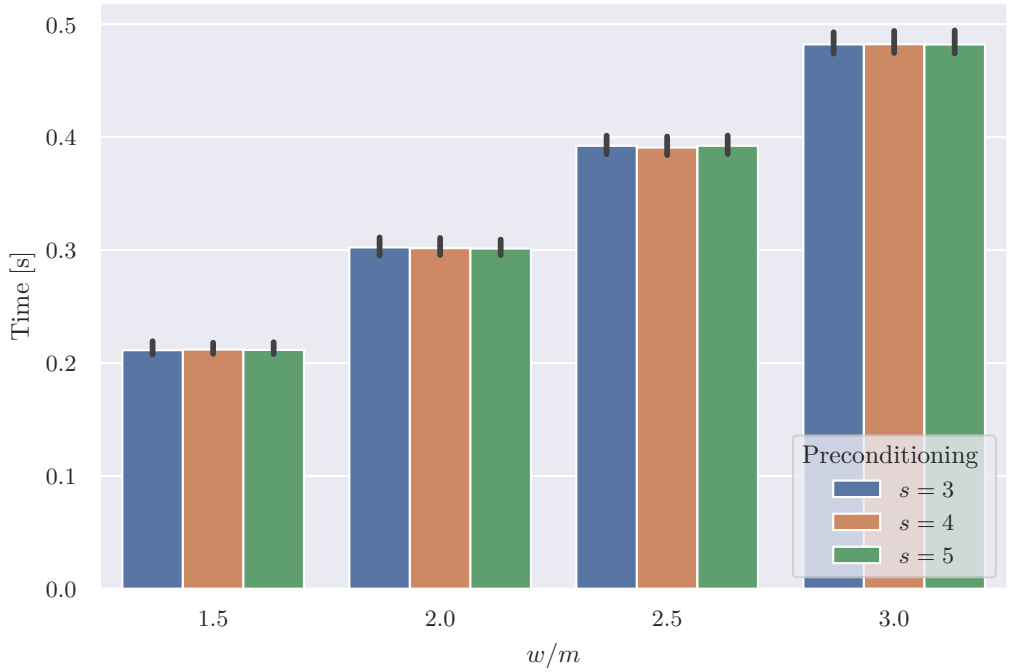

Figure 5.6.: Time spent computing the QR decomposition of $\boldsymbol{WDA}^T$

5.5. Impact of the CG Tolerance

Any direct solver for a linear system takes a fixed amount of time to achieve a solution of very high accuracy. Iterative solvers come with the benefit that the solutions are refined in every iteration so that a trade-off between running time and accuracy is possible. Due to numerical difficulties iterative solvers cannot achieve arbitrarily low errors, so we decided to fix the number of CG iterations at different values to see how they influence the solver accuracy and subsequently the course of the IPM.

Figure 5.7 shows the impact of varying the number of CG iterations on the relative errors for the preconditioned system and for the normal equation using data from all iterations. The figure also includes the relative error achieved by using a Cholesky decomposition of the normal equation. Evidently, the CG method applied to the preconditioned system fails to improve the residuals any further once the number of iterations reaches roughly 100 and the relative errors remain worse compared to the Cholesky decomposition.

The effect on the accuracy of the IPM is exemplified in Figure 5.8. Here, the solution accuracy as measured by ρ_{tol} is tracked over the final iterations of the IPM for one of the randomly generated LP instances. It can be clearly seen that at some point during the optimisation process a breakdown occurs that leads to a sudden increase of ρ_{tol}. In our experiments this was always due to an increase in ρ_p which measures the primal feasibility. Even though the minimum ρ_{tol} value is achieved at different IPM iterations

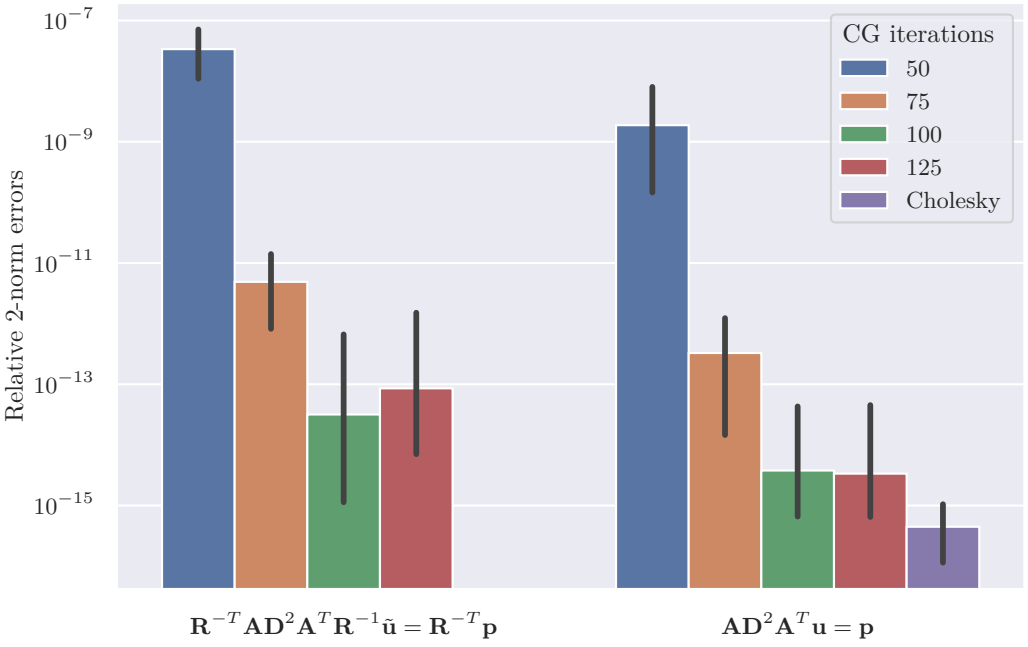

Figure 5.7.: Relative errors achieved by CG and Cholesky

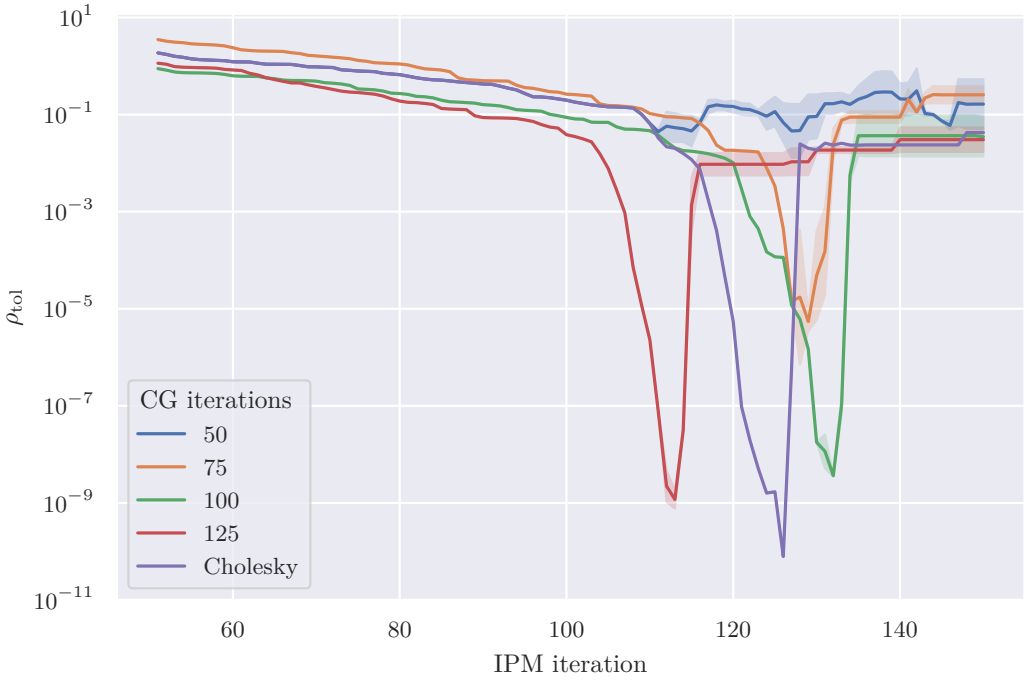

Figure 5.8.: Solution accuracy during the latter IPM iterations

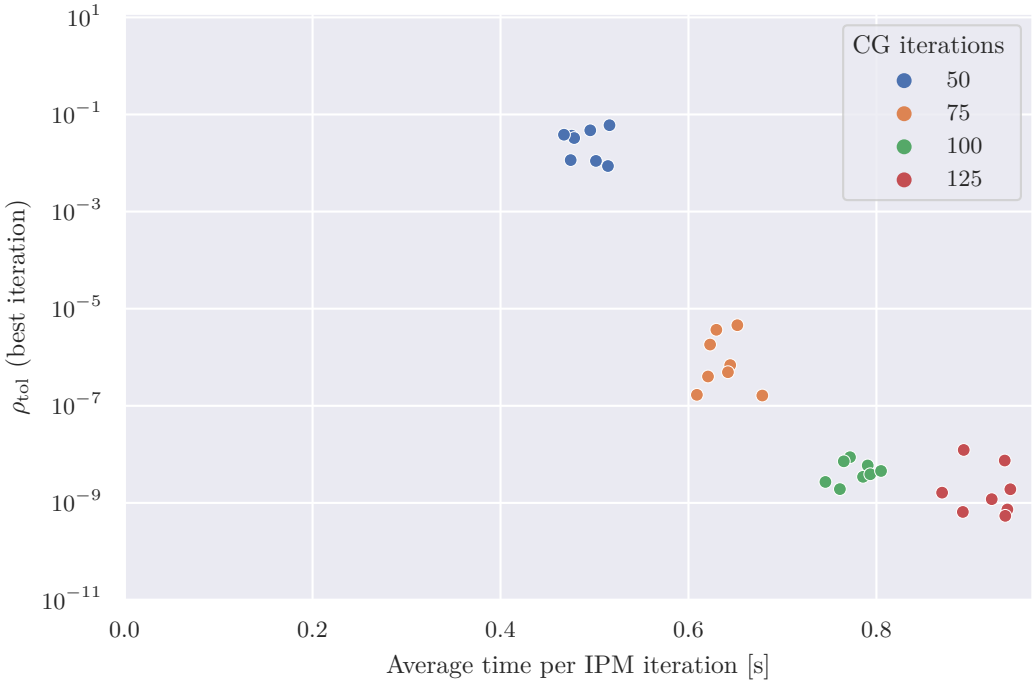

Figure 5.9.: Solution accuracy vs. time per IPM iteration

for the different parameter settings the average number of iterations it took for ρ_{tol} to reach its minimum across multiple LP instances was around 125 independently of the number of CG iterations. As expected, the minimum value of ρ_{tol} is lower the smaller the error in the solution of the normal equation. The trade-off between solution accuracy and time spent per IPM iteration can be more closely examined in Figure 5.9. Here, the best value of ρ_{tol} that was achieved during each run of the algorithm is plotted against the average time each iteration took.

This time per iteration includes both the time to compute the preconditioner $\boldsymbol{R}$ and the time it takes the CG solver to perform a fixed amount of iterations. The latter is very much influenced by the way the matrix-vector products required for the CG algorithm are implemented. For these experiments all products were evaluated lazily in the sense that the matrix $\boldsymbol{R}^{-T}\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^T\boldsymbol{R}^{-1}$ was not formed explicitly but every time a vector was to be multiplied with it, three matrix-vector products and two triangular solves were performed. Preliminary experiments indicate that due to very efficient routines for matrix inversion and matrix-matrix multiplication explicitly forming the product is similarly fast for 100 CG iterations and outperforms our implementation if more iterations are performed.

5.6. Comparison with Direct Solvers

To finish this chapter, a direct comparison between our iterative approach and direct solvers is due. The two direct methods we will compare it with are a Cholesky decomposition of $\boldsymbol{AD}^2\boldsymbol{A}^T$ and a QR decomposition of the unsketched matrix $\boldsymbol{DA}^T$. The parameters for preconditioning were chosen to be $w = 2m$ and $s = 5$ and 100 CG iterations were performed. Table 5.1 shows that, while computing the QR decomposition of the sketched matrix $\boldsymbol{WDA}^T$ is much faster than computing the QR decomposition of the full matrix $\boldsymbol{DA}^T$, the preconditioned CG method cannot compete with the speed of the Cholesky decomposition. Additionally, Table 5.2 demonstrates that the direct solvers generate more accurate solutions as the iterative approach can.

Method	Computing $\boldsymbol{WDA}^T$	QR/Cholesky decomposition	Solving $\boldsymbol{AD}^2\boldsymbol{A}^T\boldsymbol{u} = \boldsymbol{p}$	Total
Preconditioned CG	0.052	0.382	0.687	1.121
Cholesky decomposition	—	0.043	0.004	0.047
QR decomposition	—	24.074	0.005	24.079

Table 5.1.: Runtime comparison: Average time per IPM iteration in seconds

Method	Best ρ_{tol}		
	5th percentile	Median	95th percentile
Preconditioned CG	5.818×10^{-10}	1.169×10^{-8}	6.937×10^{-8}
Cholesky decomposition	3.278×10^{-11}	9.673×10^{-11}	1.299×10^{-10}
QR decomposition	8.473×10^{-11}	9.310×10^{-11}	1.015×10^{-10}

Table 5.2.: Accuracy comparison: Best ρ_{tol} over 150 IPM iterations

5.7. Summary

The experiments in this chapter showed that indeed the QR decomposition of $\boldsymbol{WDA}^T$ produces good preconditioners that are able to reduce the condition number of $\boldsymbol{AD}^2\boldsymbol{A}^T$ by multiple orders of magnitude. The quality of the linear solver for the normal equation directly affects the solution accuracy of the IPM algorithm as a whole so that the time-accuracy trade-off of the iterative solver can be transferred to a time-accuracy trade-off for the IPM algorithm. Unfortunately, even if only very few CG iterations are performed solving the normal equation using the Cholesky decomposition is much faster because computing the QR decomposition of $\boldsymbol{WDA}^T$ already takes about ten times as long as computing the Cholesky decomposition of $\boldsymbol{AD}^2\boldsymbol{A}^T$. Whether there is a parameter regime regarding matrix dimensions and the sparsity of $\boldsymbol{A}$, where the preconditioned CG method can outperform the Cholesky decomposition, is still an open question.

Bibliography

[AA00] Erling D. Andersen and Knud D. Andersen. 'The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm'. In: *High Performance Optimization*. Ed. by Hans Frenk, Kees Roos, Tamás Terlaky and Shuzhong Zhang. Springer US, 2000, pp. 197–232. ISBN: 978-1-4757-3216-0. DOI: 10.1007/978-1-4757-3216-0_8. URL: https://doi.org/10.1007/978-1-4757-3216-0_8.

[AC06] Nir Ailon and Bernard Chazelle. 'Approximate Nearest Neighbors and the Fast Johnson-Lindenstrauss Transform'. In: *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 2006, pp. 557–563. ISBN: 1595931341. DOI: 10.1145/1132516.1132597.

[Ach03] Dimitris Achlioptas. 'Database-friendly random projections: Johnson-Lindenstrauss with binary coins'. In: *Journal of Computer and System Sciences* 66.4 (2003). Special Issue on PODS 2001, pp. 671–687. ISSN: 0022-0000. DOI: 10.1016/S0022-0000(03)00025-4. URL: https://www.sciencedirect.com/science/article/pii/S0022000003000254.

[AMT10] Haim Avron, Petar Maymounkov and Sivan Toledo. 'Blendenpik: Supercharging LAPACK's Least-Squares Solver'. In: *SIAM Journal on Scientific Computing* 32.3 (2010), pp. 1217–1236. DOI: 10.1137/090767911.

[Bou+09] R. Bouyouli, G. Meurant, L. Smoch and H. Sadok. 'New results on the convergence of the conjugate gradient method'. In: *Numerical Linear Algebra with Applications* 16.3 (2009), pp. 223–236. DOI: https://doi.org/10.1002/nla.618. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.618.

[Bra+20] Jan van den Brand, Yin Tat Lee, Aaron Sidford and Zhao Song. 'Solving Tall Dense Linear Programs in Nearly Linear Time'. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. Association for Computing Machinery, 2020, pp. 775–788. ISBN: 9781450369794. DOI: 10.1145/3357713.3384309. URL: https://doi.org/10.1145/3357713.3384309.

[BV08] Ulrich Brenner and Jens Vygen. 'Analytical methods in VLSI placement'. In: *Handbook of Algorithms for VLSI Physical Design Automation* (2008), pp. 327–346.

[CFS21] Coralia Cartis, Jan Fiala and Zhen Shao. 'Efficient sparse sketching for large-scale linear least squares'. Unpublished paper. 2021.

[Cho+20] Agniva Chowdhury, Palma London, Haim Avron and Petros Drineas. 'Faster Randomized Infeasible Interior Point Methods for Tall/Wide Linear Programs'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 8704–8715. URL: https://proceedings.neurips.cc/paper/2020/file/630eff1b380505a67570dff952ce4ad7-Paper.pdf.

[CNW16] Michael B. Cohen, Jelani Nelson and David P. Woodruff. 'Optimal Approximate Matrix Product in Terms of Stable Rank'. In: *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Ed. by Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani and Davide Sangiorgi. Vol. 55. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 11:1–11:14. ISBN: 978-3-95977-013-2. DOI: 10.4230/LIPIcs.ICALP.2016.11. URL: http://drops.dagstuhl.de/opus/volltexte/2016/6278.

[Coh16] Michael B. Cohen. 'Nearly Tight Oblivious Subspace Embeddings by Trace Inequalities'. In: *Proceedings of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2016, pp. 278–287. ISBN: 9781611974331. DOI: 10.1137/1.9781611974331.ch21. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611974331.ch21.

[Dan65] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1965.

[Dem97] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Jan. 1997. ISBN: 978-0-89871-389-3. DOI: 10.1137/1.9781611971446. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971446.

[HJ91] Roger A. Horn and Charles R. Johnson. 'Singular value inequalities'. In: *Topics in Matrix Analysis*. Cambridge University Press, 1991, pp. 134–238. DOI: 10.1017/CBO9780511840371.004.

[Kar84] N. Karmarkar. 'A new polynomial-time algorithm for linear programming'. In: *Combinatorica* 4.4 (Oct. 1984), pp. 373–395. ISSN: 1439-6912. DOI: 10.1007/BF02579150. URL: https://doi.org/10.1007/BF02579150.

[Kha79] L. G. Khachiyan. 'A polynomial algorithm in linear programming'. Russian. In: *Doklady Akademii Nauk SSSR* 244 (1979), pp. 1093–1096. ISSN: 0002-3264.

[Kha80] L. G. Khachiyan. 'Polynomial algorithms in linear programming'. Russian. In: *Zhurnal Vychislitel'noĭ Matematiki i Matematicheskoĭ Fiziki* 20 (1980), pp. 51–68. ISSN: 0044-4669.

[Lon+18] Palma London, Shai Vardi, Adam Wierman and Hanling Yi. 'A Parallelizable Acceleration Framework for Packing Linear Programs'. In: *AAAI Conference on Artificial Intelligence* (2018). URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17118.

[Meh92] Sanjay Mehrotra. 'On the Implementation of a Primal-Dual Interior Point Method'. In: *SIAM Journal on Optimization* 2.4 (1992), pp. 575–601. DOI: 10.1137/0802028. URL: https://doi.org/10.1137/0802028.

[MG11] Ofer Meshi and Amir Globerson. 'An Alternating Direction Method for Dual MAP LP Relaxation'. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba and Michalis Vazirgiannis. Springer Berlin Heidelberg, 2011, pp. 470–483. ISBN: 978-3-642-23783-6.

[MO03] Renato D.C. Monteiro and Jerome O'Neal. 'Convergence Analysis of a Long-Step Primal-Dual Infeasible Interior-Point LP Algorithm Based on Iterative Linear Solvers'. In: *Manuscript, School of Industrial and Systems Engineering, Georgia Institute of Technology* (Oct. 2003).

[MSM14] Xiangrui Meng, Michael A. Saunders and Michael W. Mahoney. 'LSRN: A Parallel Iterative Solver for Strongly Over- or Underdetermined Systems'. In: *SIAM Journal on Scientific Computing* 36.2 (2014), pp. C95–C118. DOI: 10.1137/120866580. URL: https://doi.org/10.1137/120866580.

[NP93] Esmond G. Ng and Barry W. Peyton. 'Block Sparse Cholesky Algorithms on Advanced Uniprocessor Computers'. In: *SIAM Journal on Scientific Computing* 14.5 (1993), pp. 1034–1056. DOI: 10.1137/0914063.

[Vir+20] Pauli Virtanen et al. 'SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python'. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[Woo14] David P. Woodruff. 'Sketching as a Tool for Numerical Linear Algebra'. In: *Foundations and Trends in Theoretical Computer Science* 10.1–2 (2014), pp. 1–157. ISSN: 1551-305X. DOI: 10.1561/0400000060.

[Wri97] Stephen J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997. ISBN: 0-89871-382-X/pbk. DOI: 10.1137/1.9781611971453. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971453.

[XHY96] Xiaojie Xu, Pi-Fang Hung and Yinyu Ye. 'A simplified homogeneous and self-dual linear programming algorithm and its implementation'. In: *Annals of Operations Research* 62.1 (Dec. 1996), pp. 151–171. ISSN: 1572-9338. DOI: 10.1007/BF02206815. URL: https://doi.org/10.1007/BF02206815.

[YZ11] Junfeng Yang and Yin Zhang. 'Alternating Direction Algorithms for ℓ_1-Problems in Compressive Sensing'. In: *SIAM Journal on Scientific Computing* 33.1 (2011), pp. 250–278. DOI: 10.1137/090777761.

[Zhu+04] Ji Zhu, Saharon Rosset, Robert Tibshirani and Trevor Hastie. '1-norm Support Vector Machines'. In: *Advances in Neural Information Processing Systems*. Ed. by S. Thrun, L. Saul and B. Schölkopf. Vol. 16. MIT Press, 2004. URL: https://proceedings.neurips.cc/paper/2003/file/49d4b2faeb4b7b9e745775793141e2b2-Paper.pdf.

A. Source Code Excerpt

This appendix contains the two key elements of the implementation: The function `_construct_sketching_matrix` is concerned with generating the sketching matrix. It is capable of generating sparse sketching matrices and Gaussian sketches but only sparse sketches were used in our experiments. The function `_get_solver` determines how the normal equation is solved and is capable of handling the three methods discussed in Section 5.6.

```
import numpy as np
import scipy

...

default_rng = np.random.default_rng()

def _construct_sketching_matrix(
    w, n, s=3, sparse=True, rng=default_rng
):
    """
    Randomly construct a sketching matrix of size (w, n) where w is
    assumed to be smaller than n. If sparse is True, each column
    contains s nonzero entries randomly drawn from +-1/sqrt(s).
    Otherwise all matrix entries are drawn from a normal distribution.

    Parameters
    ----------
    w : int
        Number of rows of the sketching matrix
    n : int
        Number columns of the sketching matrix
    s : int
        Number of nonzero entries in each row if ``sparse = True``
    sparse : bool
        True if the sketching matrix should be sparse.
    rng : np.random.Generator (default = np.random.default_rng())
        A random number generator used to construct the random matrix

    Returns
```

```
    -------
    A random sketching matrix

    """
    if sparse:
        # The matrix is constructed in the COO format:
        # data[i] is the entry at (row_indices[i], column_indices[i])
        # for each index i

        # 1. Generate the entries: randomly chosen from +- 1/sqrt(s)
        data = rng.choice(
            [-1 / np.sqrt(s), 1 / np.sqrt(s)], size=s * n
        )

        # 2. Generate the row indices: Draw s elements from
        # {1, ..., w} without replacement 1.1*n times. Delete all
        # groups of s elements where the same element was chosen twice
        # and take the first n of the remaining groups to be the
        # row_indices.
        row_indices = rng.choice(w, size=(int(n * 1.1), s))
        row_indices.sort()
        row_indices = row_indices[
            (row_indices[..., 1:] != row_indices[..., :-1]).all(
                axis=-1
            )
        ]
        row_indices = row_indices[:n]
        row_indices.shape = (n * s,)

        # 3. Generate the column indices by repeating each i from
        # {1, ..., n} s times: [1, 1, 1, 2, 2, 2, ..., n, n, n] for
        # s = 3 for example
        column_indices = np.repeat(np.arange(n), s)

        # 4. Construct the matrix as described and convert it to the
        # CSR sparse matrix format before returning to allow fast
        # matrix multiplication
        mat = scipy.sparse.coo_matrix(
            (data, (row_indices, column_indices)), shape=(w, n)
        )
        return mat.tocsr()
    else:
        # Return a Gaussian sketch
        return rng.normal(size=(w, n)) / np.sqrt(w)
```

```
# This function is called once for every IPM iteration to get a method
# for solving the normal equation for arbitrary right-hand sides. Note
# that the notation for the matrices in the paper by Andersen and
# Andersen is different to the one used in this work. The matrix
# "Dinv" thus corresponds to D^2, "Dinv_half" corresponds to D and "M"
# corresponds to A D^2 A^T. The diagonal matrix "Dinv" is passed to
# this function as a 1-D vector of its diagonal elements.
def _get_solver(A, Dinv, options):
    """
    Given ``A`` and `` Dinv`` return a handle to a linear system
    solver for the matrix ``M = A @ Dinv @ A.T``, i.e. a function that
    takes in an arbitrary right-hand side ``r`` and returns
    ``M^{-1} r``

    Parameters
    ----------
    A : 2-D array
    Dinv : 1-D array
        As defined in [4] Equation 8.31
    options : AllOptions
        Provides the options to choose the linear solver.

    Returns
    -------
    solve : function
        Handle to the appropriate solver function

    References
    ----------
    .. [4] Andersen, Erling D., and Knud D. Andersen. "The MOSEK
           interior point optimizer for linear programming: an
           implementation of the homogeneous algorithm." High
           performance optimization. Springer US, 2000. 197-232.
    """
    method = options.linear_solver.solver

    if method is Solver.CHOLESKY:
        # The function scipy.sparse.diags turns a 1-D vector into a
        # sparse diagonal matrix where the vector entries are on the
        # diagonal, "@" is they Python operator for matrix
        # multiplication
        M = A @ scipy.sparse.diags(Dinv, 0, format="csc") @ A.T
```

```
        # The matrix L is the Cholesky factor of M: LL^T = M
        # The toarray() method converts the sparse matrix to a dense
        # one, so that a dense Cholesky solver can be used.
        L = scipy.linalg.cho_factor(M.toarray())

        def solve(r):
            return scipy.linalg.cho_solve(L, r)

        return solve

    elif method is Solver.QR:
        Dinv_half = np.sqrt(Dinv)
        M_half = scipy.sparse.diags(Dinv_half, format="csc") @ A.T

        # Find the QR decomposition of DA^T.
        R = np.linalg.qr(M_half.toarray(), mode="r")

        def solve(r):
            r2 = scipy.linalg.solve_triangular(
                R, r, trans="T"
            )  # r2 = R^{-T} r
            x = scipy.linalg.solve_triangular(
                R, r2, trans="N"
            )  # x = R^{-1} r2
            return x

        return solve

    elif method is Solver.PRECONDITIONED_CG:
        Dinv_half = np.sqrt(Dinv)

        # Construct the sketching matrix according to the chosen
        # options:
        # - sketching_factor determines the w/m factor
        # - sketching_sparsity determines the value of s
        sketching_matrix = _construct_sketching_matrix(
            w=int(
                options.preconditioning.sketching_factor * A.shape[0]
            ),
            n=A.shape[1],
            s=options.preconditioning.sketching_sparsity,
            sparse=True,
        )
```

```
# Explicitly construct the sketched matrix WDA^T ...
sketched_matrix = (
    sketching_matrix
    @ scipy.sparse.diags(Dinv_half, format="csc")
    @ A.T
)

# ... and find its QR decomposition
R = np.linalg.qr(sketched_matrix.toarray(), mode="r")

# scipy.sparse.linalg.aslinearoperator turns matrices into
# linear operators. They can be chained together using matrix
# multiplication syntax without forming the product
# explicitly. Taking the product with a vector afterwards
# applies all of the matrices in the right order.
A_op = scipy.sparse.linalg.aslinearoperator(A)
Dinv_op = scipy.sparse.linalg.aslinearoperator(
    scipy.sparse.diags(Dinv, 0, format="csc")
)
# The linear operator of R^{-1} is constructed by defining
# that multiplication with R^{-1} and R^{-T} can be performed
# using triangular solves
Rinv_op = scipy.sparse.linalg.LinearOperator(
    R.shape,  # R is square
    matvec=lambda x: scipy.linalg.solve_triangular(
        R, x, trans="N"
    ),
    rmatvec=lambda x: scipy.linalg.solve_triangular(
        R, x, trans="T"
    ),
)
# The preconditioned matrix is now given by the correct chain
# of matrix-vector multiplications
preconditioned_matrix = (
    Rinv_op.T @ A_op @ Dinv_op @ A_op.T @ Rinv_op
)

def solve(r):
    # Solve the preconditioned linear system with the CG
    # method:
    # - maxiter bounds the number of CG iterations
    # - tol is the required relative tolerance (2-norm)
    # - atol is the required absolute tolerance (2-norm)
```

```
        # We used tol = atol = 0; maxiter = 50, 75, 100, 125
        x, info = scipy.sparse.linalg.cg(
            A=preconditioned_matrix,
            b=Rinv_op.T @ r,
            maxiter=options.linear_solver.solver_maxiter,
            tol=options.linear_solver.solver_rtol,
            atol=options.linear_solver.solver_atol,
        )
        # info < 0 indicates a numerical failure
        if info < 0:
            raise scipy.linalg.LinAlgError(f"CG failed ({info})!")
        return Rinv_op @ x

    return solve
```