

IMMUNITY TO INCREASING CONDITION NUMBERS OF LINEAR SUPERIORIZATION VERSUS LINEAR PROGRAMMING

JAN SCHRÖDER¹, YAIR CENSOR², PHILIPP SÜSS^{1,*}, KARL-HEINZ KÜFER¹

¹*Optimization Department, Fraunhofer - ITWM, Kaiserslautern 67663, Germany*

²*Department of Mathematics, University of Haifa, Mt. Carmel, Haifa 3498838, Israel*

Abstract. Given a family of linear constraints and a linear objective function one can consider whether to apply a Linear Programming (LP) algorithm or use a Linear Superiorization (LinSup) algorithm on this data. In the LP methodology one aims at finding an optimal point, i.e., a point that fulfills the constraints and has the minimal value of the objective function over these constraints. The Linear Superiorization approach considers the same data as in linear programming problems but instead of attempting to solve with linear programming methods it employs perturbation resilient feasibility-seeking algorithms that steer the iterations toward a feasible point with reduced (not necessarily minimal) objective function value. This aim of the superiorization method (SM) is less demanding than aiming to reach full-fledged constrained optimality and it places more importance on reaching feasibility than on reaching optimality. Previous studies compared LP and LinSup in terms of their respective outputs and the resources they use. Here, we investigate classical LP approaches and LinSup in terms of their sensitivity to condition numbers of the system of linear constraints. Condition numbers are a measure for the impact of deviations in the input data on the output of a problem and, in particular, they describe the factor of error propagation when given wrong or erroneous data. Therefore, the ability of LP and LinSup to cope with increased condition numbers, thus with ill-posed problems, is an important matter to consider which was not studied until now. We investigate experimentally the advantages and disadvantages of both LP and LinSup on exemplary sets of data of problems of linear programming with multiple condition numbers and different problem dimensions.

Keywords. Bounded perturbations; Condition number; Feasibility-seeking; Ill-posed problem; Linear superiorization.

2020 Mathematics Subject Classification. 90C29, 90C90, 92C50.

1. INTRODUCTION

In this paper we investigate experimentally and discuss the behavior of the linear superiorization (LinSup) methodology for various different condition numbers and compare it to the behavior of several linear programming (LP) solvers. The paper is in the spirit of [1] and contains a follow-up analysis regarding the impact of condition numbers on the runtime of the respective algorithms.

*Corresponding author.

E-mail address: philipp.suess@itwm.fraunhofer.de (P. Süß).

Received 6 March 2026; Accepted 3 July 2026; Published online 1 August 2026.

©2026 Journal of Applied and Numerical Optimization

We set up linear programming problems with different condition numbers of the linear constraints in a manner that keeps the problems structurally similar in order to enable the comparative experimental study. Then we apply to these problems the LinSup superiorization method and some established implementations of the simplex algorithm and an interior point method.

Imposing stopping rules that are fair toward all algorithms, we compare the outputs on problems with increasing condition numbers and on problems with increasing dimensions.

1.1. The experimental methodology. LinSup is not an LP solver. As an offspring of the superiorization methodology (SM), LinSup perturbs linear feasibility-seeking algorithms to reduce objective function values while retaining the feasibility-seeking nature of the algorithm and without paying a high computational price.

The declared aim of LinSup is to find a feasible point whose objective function value is “superior” (meaning smaller or equal) to that of a feasible point that can be reached by the corresponding unperturbed feasibility-seeking algorithm, i.e., the exactly same feasibility-seeking algorithm that the LinSup employs. This aim is less demanding than full-fledged linear programming but more demanding than plain feasibility-seeking.¹

In view of this difference between LinSup and an LP solver, but recognizing that both are initialized at a starting point which is not known to be a feasible point, we maintain the following experimental methodology. For a given problem we run LinSup until it reaches some predetermined fixed small threshold level of “infeasibility” (i.e., “constraints violation”). When LinSup stopped according to this stopping rule we record the objective function value at this point and the runtime elapsed.

Then we run on the same problem, starting from the same initialization point, the LP solver that we wish to include in our comparative study, and stop it when it reaches the same predetermined fixed level of “infeasibility” that was used for stopping the LinSup run. When stopped we record the objective function value at this point and the runtime elapsed for this LP solver as well.

We also record the infeasibility and the objective function values along the whole runtime and use these in order to demonstrate the behavior of each algorithm by plotting its objective function values and its infeasibility values versus time, respectively. More details are supplied later in the paper.

1.2. Context and previous work. The superiorization methodology (SM) is a relatively new method that aims to improve the performance of an existing iterative algorithm by interlacing into it perturbation steps [2]. In the context of optimization problems, this concept can be applied by interlacing into a feasibility-seeking algorithm (for example a projection method) perturbations of negative gradient steps that reduce the objective function values.

Since its development, the SM has successfully been applied in various practical applications such as intensity-modulated radio therapy [3], image reconstruction [4] and telecommunication networks [5]. Recent works include the use of machine learning for the determination of the

¹Although we came across it after the SM was developed, one can borrow support for the reasoning of the SM from the American scientist and Nobel-laureate Herbert Simon who was in favor of “satisficing” rather than “maximizing”. Satisficing is a decision-making strategy that aims for a satisfactory or adequate result, rather than the optimal solution. This is because aiming for the optimal solution may necessitate needless expenditure of time, energy and resources. The term “satisfice” was coined by Herbert Simon in 1956 ([6]). See also: <https://en.wikipedia.org/wiki/Satisficing>.

perturbation steps [7] and systematic biasing of steepest descent methods in order to obtain solutions that are less dependent on noise in the data [8]. Both have been applied to imaging.

In the particular context of linear programming problems (LPs), there is a vast literature for both feasibility-seeking problems (for example the Agmon-Motzkin-Schönberg (AMS) algorithm [9], [10], [11, Algorithm 5.4.1]) and for the LP problem (e.g., simplex algorithm, ellipsoid method [12]).

Furthermore, for the LinSup case, the “guarantee question of the SM” has been answered positively. This is the question whether superiorization can actually converge to a feasible point with a smaller or equal objective function value than that of the point reached by the unsuperiorized version of the same feasibility-seeking algorithm. In [13], the authors employed the principle of concentration of measure to show that this result holds with high probability. A similar conclusion for the nonlinear case is yet to be found.

In this paper we compare experimentally the SM with optimization algorithms for LPs in an organized, reproducible and fair manner for problems of varying difficulty, indicated by the problems’ dimensions and their condition numbers.

Accordingly, our work is an extension of the results in [1], where other features of LinSup were compared to the simplex method. We augment these results by varying the condition numbers across multiple problem instances. The condition number of the linear system has been known to have a significant impact on the performance of certain methods, often leading to the failure of an algorithm ([14]). Since many real-world problems have an inherently high condition number, immunity against ill-conditioning of problems is a desirable property for any algorithm. Often it is enough to have a rough understanding of the order of magnitude of the condition number and there are efficient algorithms for its estimation ([15, 16]). For literature on preconditioning of matrix problems, we refer to, e.g., [17].

1.3. Our results. In this project we used LP data aiming at the task of finding a feasible point with reduced (not necessarily minimal) objective function value. In our numerical work the superiorization method LinSup performed better than LP solvers for this task. This better performance, for this task, was investigated for problems with increasing condition numbers of the linear systems and with increasing problem sizes. We found that in each of these cases LinSup delivered better results than some established LP solvers.

1.4. Outline. We give a brief overview of the superiorization method and of condition numbers in Sections 2.1 and 2.2, respectively. We set up multiple linear programming problems with varying condition numbers in Section 3.1 and run the LinSup method and several LP algorithms on them.

For details of our implementation of the LinSup method see Section 3.2. We give a brief description of the optimization suites that we use for our comparisons in Section 3.3 and of the experimental setup in Section 3.4. In Section 4 we present the numerical results, followed by a discussion of remaining challenges and future work in Section 5.

2. PRELIMINARY BACKGROUND

2.1. The superiorization methodology. The Superiorization Methodology (SM) evolved from the investigation of feasibility-seeking models of some important real-world problems such as image reconstruction from projections and radiation therapy treatment planning.

Feasibility-seeking algorithms, mainly projection methods, generate iterative sequences that (under reasonable conditions) converge to a point in the feasible set. Their main advantage is that they perform projections onto the individual sets whose intersection is the feasible set and not directly onto the feasible set and the underlying premise is that such projections onto the individual sets are more manageable.

When one wishes to find feasible points with a reduced, not necessarily minimized, value of an imposed objective function then the SM comes into play. The principle of the SM is to interlace into the iterates of a feasibility-seeking iterative process perturbations that steer the iterates toward superior (meaning smaller or equal) objective function values without losing the overall convergence of the sequence of perturbed iterates to a feasible point. To this end “bounded perturbations” are used.

How all this is done is rigorously described in earlier papers on the SM, consult, e.g., [2] for a recent review, read also [18]. A key feature of the SM is that it does not aim for a constrained optimal function value, but is content with settling for a feasible point with reduced objective function value – reduced in comparison to the objective function value of a feasible point that would be reached by the same feasibility-seeking algorithm without perturbations.

For many applications this is sufficient, in particular, whenever the introduction of an objective function is only a secondary goal. Fulfillment of the constraints, in this context, is considered by the modeler of the real-world problem to be much more important, see, e.g., [2, 4, 3].

Many papers on the SM are cited in [19] which is a Webpage dedicated to superiorization and perturbation resilience of algorithms that contains a continuously updated bibliography on the subject. This Webpage (which includes 208 entries as of June 6, 2026) is a source for the wealth of work done in this field to date, including two special issues of journals [20] and [21] dedicated to research of the SM. Recent work includes [22, 23, 24, 25, 26, 27].

We find especially interesting the recent work of Ma et al. [28] who proposed a novel decomposition framework for derivative-free optimization (DFO) algorithms which significantly extended the scope of current DFO solvers to larger-scale problems. They proved that their proposed framework closely relates to the superiorization methodology.

Information about feasibility-seeking algorithms and the convex feasibility problem, that lie at the foundation of the SM approach, can be found, e.g., in Bauschke and Borwein’s SIAM review paper [29] and in [30].

2.2. Condition numbers. The relative condition number is a measure of the impact of deviations in the input data on the output data of a problem. In particular, it describes the factor of error propagation when given wrong or erroneous data.

Let the function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ represent some mathematical problem and let $x \in \mathbb{R}^n$ be its input, where $\mathbb{R}^n$ stands for the Euclidean n -dimensional space. Denote with $\tilde{x} \in \mathbb{R}^n$ the disturbed input data. Then the relative condition number of the problem at the point x is defined as (see

[31, equation (12.4)])

$$\kappa_{\text{rel}} := \limsup_{\tilde{x} \rightarrow x} \frac{\|f(\tilde{x}) - f(x)\|}{\|\tilde{x} - x\|} \frac{\|x\|}{\|f(x)\|}, \quad (2.1)$$

as long as $f(x) \neq 0$. Otherwise, it is $\kappa_{\text{rel}} = \infty$.

In particular, the condition number is independent of a chosen numerical algorithm for solving the problem f , but the algorithms convergence speed may depend on the magnitude of the condition number (see [14, Section 4]).

In the following, we are interested in the condition number of matrices. When $f(x) = Ax$ describes the problem of matrix multiplication, where $A \in \mathbb{R}^{m \times n}$ and with $\|\cdot\| = \|\cdot\|_2$, the above formula becomes

$$\kappa_{\text{rel}} = \limsup_{\tilde{x} \rightarrow x} \frac{\|A(\tilde{x} - x)\|}{\|\tilde{x} - x\|} \frac{\|x\|}{\|Ax\|}. \quad (2.2)$$

Since f is differentiable and by writing $\tilde{x} - x = hv$, for some unit vector v and $h = \|\tilde{x} - x\|$ we get

$$\kappa_{\text{rel}} = \lim_{h \rightarrow 0} \frac{\|A(x + hv - x)\|}{h} \frac{\|x\|}{\|Ax\|} = \|A\| \frac{\|x\|}{\|Ax\|} \leq \|A\| \|A^{-1}\|. \quad (2.3)$$

The term on the right is called the “condition number of the matrix A ” (see [31, Equation (12.15)]), where A^{-1} denotes the inverse or, if A is non-square, the pseudo-inverse of A ,

$$\kappa(A) := \|A\| \|A^{-1}\| = \frac{\sigma_{\max}}{\sigma_{\min}}, \quad (2.4)$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are the maximal and minimal nonzero singular values of A , respectively.

The condition number plays a significant role in the analysis of numerical problems and is subject to extensive studies in the literature [14, 15, 31, 32]. Several methods exist to improve high condition numbers of ill-conditioned problems (these are the, so-called, pre-conditioning methods, see e.g., [17]) in order to increase the accuracy of calculated solutions. This is often necessary because many real-world applications give rise to condition numbers of significant magnitude. This is the key motivation for the investigation in this paper.

3. PROBLEM SETUP AND IMPLEMENTATION DETAILS

3.1. The problem formulation. We consider LP problems of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \langle c, x \rangle \\ \text{s.t.} \quad & Ax \leq b \\ & \ell \leq x \leq u, \end{aligned} \quad (3.1)$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c, \ell, u \in \mathbb{R}^n$. Write $A = U\Sigma V$ via the singular value decomposition (cf. [33]) with semi-orthogonal matrices U and V and diagonal matrix $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_q)$ of singular values. Without loss of generality, let $\sigma_1 \geq \dots \geq \sigma_q > 0$. The condition number κ of A is (see [32])

$$\kappa(A) := \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{\sigma_1}{\sigma_q}. \quad (3.2)$$

We want to construct a sequence of matrices A_κ of a specified condition number κ in such a way that they remain structurally similar to each other. To this end we reverse the singular value

decomposition, that is, we create exactly one pair of U and V which contains the structure of the problem and construct, for different values of κ , diagonal matrices Σ_κ , which impose the condition number of the problem. Then, we calculate $A_\kappa := U\Sigma_\kappa V$.

As any matrix has a singular value decomposition, this makes it possible to define any matrix via this approach too. In our construction we focus, without loss of generality, on matrices of full rank, because otherwise one can remove rows or columns until full rank is achieved. For any chosen value of κ we construct the diagonal matrix Σ_κ , by setting $q := \min(m, n) = \text{rank}(A)$ and defining singular values as

$$\sigma_i := \frac{t}{z_i} + \frac{1-t}{s}, \quad (3.3)$$

where $t := \frac{\kappa-1}{q-1}$, $s = 10$ and $z_i = \frac{s \cdot i}{q}$. We chose this setup of the singular values σ_i , because in real-world applications the singular values often behave approximately proportional to $\frac{1}{i}$ (instead of linear which seems like an obvious first choice for our problem).

The parameter s is used to control the magnitude of the singular values since in this model we always have $\sigma_q = \frac{1}{s}$. Due to the choice of t it is easy to see that

$$\frac{\sigma_1}{\sigma_q} = \frac{\sigma_{\max}}{\sigma_{\min}} = \kappa \quad (3.4)$$

as desired. For U and V we generate random semi-orthogonal matrices and set

$$A = U \cdot \text{diag}(\sigma_1, \dots, \sigma_q) \cdot V. \quad (3.5)$$

Furthermore, in order to guarantee the non-emptiness of the feasible set in (3.1) we set $b := A\mathbb{1} + \mathbb{1}$, $u = -\ell = 100 \cdot \mathbb{1}$ and randomly choose c . This choice of parameters implies the feasibility of $x = \mathbb{1}$.

We use the above set up to generate LP problems for multiple dimensions, with matrix sizes ranging from 80×100 to 4000×5000 (cf. [1]).

3.2. The superiorization algorithm. We aim to apply separately LP solvers and the LinSup method for the data $A = (a^i)_{i=1}^m, b, c, \ell, u$ that appears in (3.1). In LinSup, presented here in Algorithm 1, we chose for the feasibility-seeking algorithm the projection method of Agmon Motzkin and Schönberg (AMS) as the “basic algorithm”.

For some (usually small value) $r \geq 0$, this algorithm sequentially cyclically projects onto the individual half-spaces $\langle a^i, x \rangle \leq b_i$ using (see, e.g., [1, p. 6]) for each individual projection the closed-form formula

$$P_i(x) := \begin{cases} x - \lambda \frac{\langle a^i, x \rangle - b_i + r \|a^i\|}{\|a^i\|^2} a^i, & \text{if } \langle a^i, x \rangle > b_i, \\ x, & \text{otherwise,} \end{cases} \quad (3.6)$$

and a full sweep through all half-spaces is done by the algorithmic operator $\mathcal{A}$, which is a composition of the individual projections

$$\mathcal{A} := P_m \circ \dots \circ P_1. \quad (3.7)$$

The parameter λ describes a relative overshoot or undershoot of the projection. For our purposes we set $\lambda = 1$ so that (if $r = 0$) the projection P_i maps precisely onto the i -th individual half-space $\langle a^i, x \rangle \leq b_i$.

For the direction vectors in the perturbations used in LinSup of Algorithm 1 we chose the normalized negative gradient of the objective function in (3.1), which is constant throughout and equals $-\frac{c}{\|c\|}$.

For the step-sizes η_k we take an exponentially decreasing null sequence $(10 \cdot 0.99^n)_{n \in \mathbb{N}}$ with restarts every $\tau_{\text{restart}} = 20$ iterations as described in [34, p. 6]. The starting step-size $\eta_0 = 10$ is decreased by the kernel $\alpha = 0.99$ in each iteration, unless there is a restart. In that case, we set $\eta_k = \eta_0 \alpha^\rho$, where ρ is the number of previous restarts during this run.

In other words, the k -th iteration consists of a gradient step $-\eta_k \frac{c}{\|c\|}$ followed by a cyclic sweep of projections onto the half-spaces via $\mathcal{A}$ as given in (3.6) and (3.7).

This process was repeated, until the iterate x^k became feasible up to the infeasibility tolerance of $\varepsilon = 10^{-8}$ and the relative change $\frac{\|x^k - x^{k-1}\|}{\|x^{k-1}\|}$ from the previous iterate was negligible, i.e., became smaller than ε . Algorithm 1 is the pseudo-code for this LinSup process. All runs were initialized at the all-zeros vector $x^0 = 0$. The parameters ε , α , η_0 , λ , τ_{restart} can be adjusted for individual preferences or a particular problem.

Algorithm 1: The LinSup superiorization algorithm

Data: A, b, c, ℓ, u, x^0

Output: Superiorized vector y

Parameters: $\varepsilon = 10^{-8}$, $\alpha = 0.99$, $\eta_0 = 10$, $\lambda = 1$, $\tau_{\text{restart}} = 20$

```

1  $k \leftarrow 0$ ;
2  $\tau \leftarrow 0$ ;
3  $\rho \leftarrow 0$ ;
4  $x^{-1} \leftarrow x^0 + \mathbb{1}$ ;
5 while  $\max_i (\langle a^i, x^k \rangle - b_i) \geq \varepsilon$  or  $\frac{\|x^k - x^{k-1}\|}{\|x^{k-1}\|} \geq \varepsilon$  do
6    $x^{k+1} \leftarrow x^k - \eta_k \frac{c}{\|c\|}$ ;
7    $x^{k+1} \leftarrow \mathcal{A}(x^{k+1})$ ;
8    $x^{k+1} \leftarrow \max(x^{k+1}, \ell)$ ;
9    $x^{k+1} \leftarrow \min(x^{k+1}, u)$ ;
10   $\tau \leftarrow \tau + 1$ ;
11  if  $\tau < \tau_{\text{restart}}$  then
12     $\eta_{k+1} \leftarrow \eta_k \alpha$ ;
13  else
14     $\tau \leftarrow 0$ ;
15     $\rho \leftarrow \rho + 1$ ;
16     $\eta_{k+1} \leftarrow \eta_0 \alpha^\rho$ ;
17  end
18   $k \leftarrow k + 1$ ;
19 end
20  $y \leftarrow x^k$ ;
21 return  $y$ 

```

3.3. The LP solvers. We compared the actions of LinSup with those of the following LP solvers:

- (1) `scipy.simplex`
- (2) `scipy.revised simplex`
- (3) `scipy.interior-point`
- (4) `gurobi.primal simplex`.

Scipy is a library for scientific computing in the programming language Python. It is freely available at www.scipy.org. Its optimization suite `scipy.optimize` contains multiple implementations of common optimization algorithms like the SQP method, the dogleg method or the conjugate gradient method. Since we are working with the data of an LP, we employed specialized LP methods only, in particular the simplex and revised simplex method as well as the interior-point method of scipy's `linprog` function.

Gurobi is a commercial state-of-the-art solver for linear and nonlinear, continuous and (mixed) integer problems. It is available at www.gurobi.com and offers a wide range of customization, including multiple algorithms, globalization strategies, preconditioning, etc. For our experiments we turned off Gurobi's automated choice of algorithm and instead forced it to use its implementation of the primal simplex to ensure consistency of the output data.

3.4. Setting up the experiments. To compare the numerical performance of the different algorithms with each other in a fair manner we performed the following steps: First we ran the LinSup algorithm with an infeasibility threshold of $\varepsilon = 10^{-8}$ as a stopping rule. We recorded the runtimes, objective function values and the maximal constraint violations at each iteration k as well as the total runtime T_{sup} , i.e. the time that the superiorization algorithm required until its usual termination criterion. Note that this value will be different for each individual experiment.

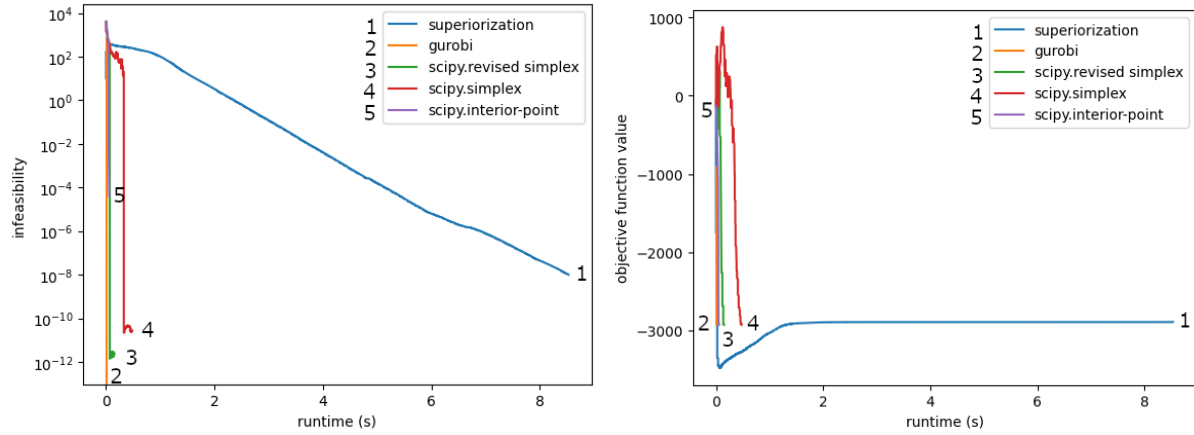
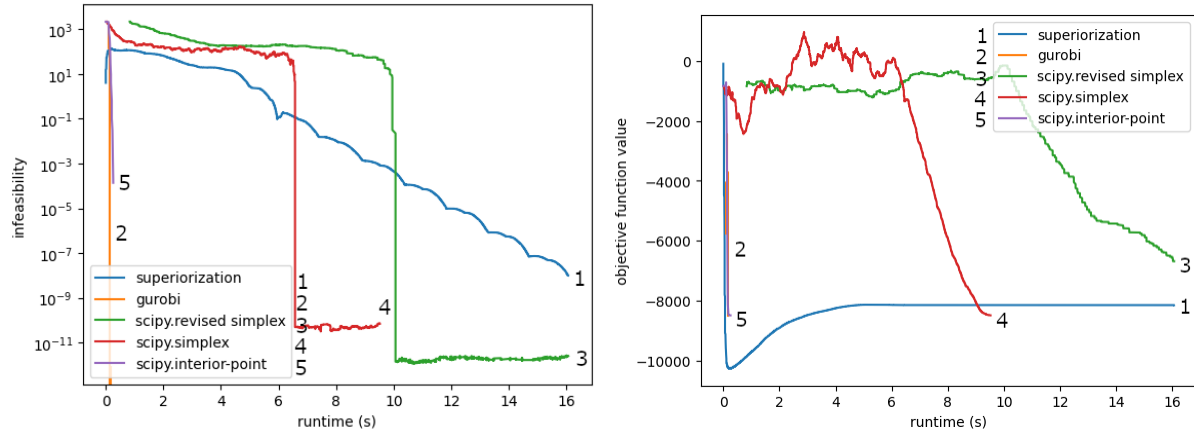
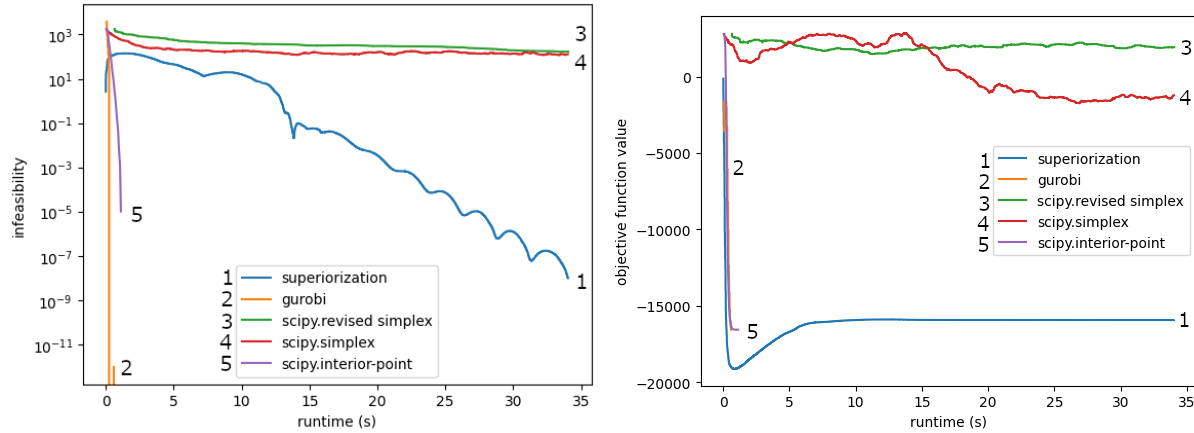
Then we turned toward the LP solvers: We imposed on each solver the same infeasibility threshold of $\varepsilon = 10^{-8}$ as a stopping rule. Additionally we required the solver to stop, if it did not stop earlier, after the time T_{sup} was exceeded. In this way, if the solver would not terminate according to the infeasibility threshold we stopped it after the time T_{sup} was reached and then extracted the current iterate.

We then compared objective function values and maximal constraint violation of the LP solver with the corresponding values from the LinSup algorithm. Similarly, we also documented the runtimes, objective function values and maximal constraint violations for each iteration of the LP solvers and plotted the runtimes against objective function values and maximal constraint violations.

This experimental set up does not aim at reaching LP optimality because LinSup is not an LP solver. The aim is to reach a certain level of infeasibility (i.e., getting close to feasibility as we wish) and, at such a level, compare LinSup with the LP solvers with respect to the runtimes needed to reach this level of infeasibility and by how they performed in reducing (not necessarily reaching optimality) the objective function value.

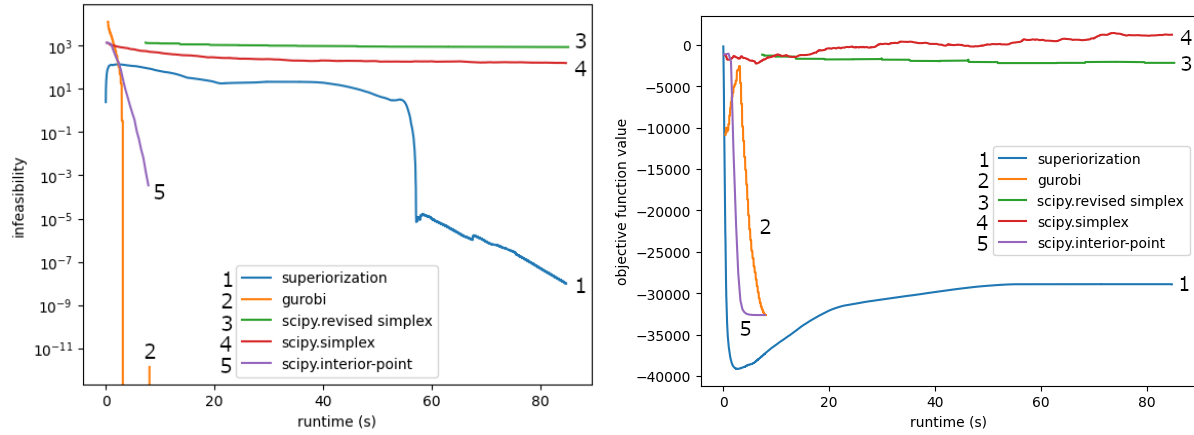
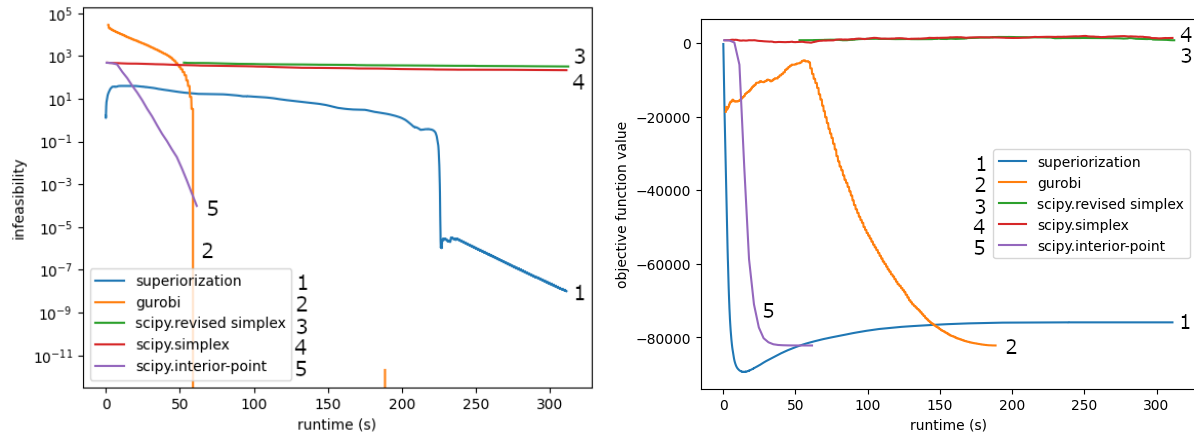
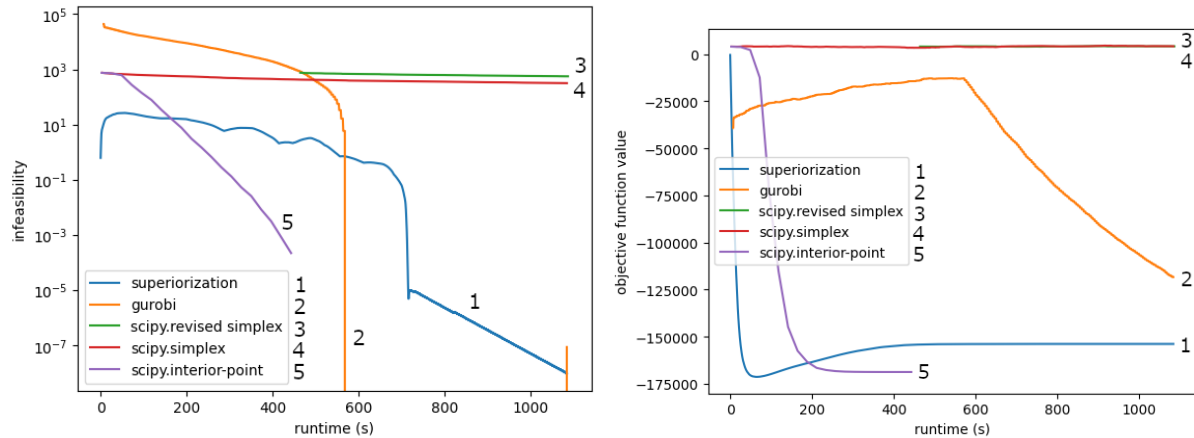
4. EXPERIMENTAL RESULTS OF THE COMPARATIVE INVESTIGATION

We split the presentation of our results into two parts, first considering the effects of increasing problem sizes and second considering the effects of increasing condition numbers for fixed sizes.

(A) 80×100 (B) 200×250 

(C) 400×500 . In this dimension the superiorization algorithm starts terminating before scipy's simplex algorithms. We can see that at the time of termination, they have not even attained feasibility.

FIGURE 1. Dimensions $80 \times 100 - 400 \times 500$ at fixed $\kappa = 1000$.

(A) 800×1000 (B) 2000×2500 

(C) 4000×5000 . In this dimension we see that the superiorization algorithm terminates even before Gurobi's simplex does.

FIGURE 2. Dimensions $800 \times 1000 - 4000 \times 5000$ at fixed $\kappa = 1000$.

4.1. Sensitivity to increasing sizes when the condition number is fixed. Figures 1a–2c show the behavior of the different algorithms on problems with increasing dimension for fixed condition number $\kappa = 1000$. On the left we see the maximum violation of the constraints $\max_i(\langle a^i, x \rangle - b_i)$ plotted against the runtime. On the right we have the corresponding objective function values $\langle c, x \rangle$ plotted against the runtime.

The trend is clear: While in problems with smaller dimensions the simplex algorithms are considerably faster, with increasing problem dimensions the simplex implementations take much longer to terminate, up to the point where the superiorization method terminates quicker. We also see a clear difference in the algorithms styles: Simplex aims to become feasible first and then starts to improve the objective function. Superiorization, on the other hand, reduces objective function values in its initial iterations because then the step-sizes of the perturbations are still large and only as iterations proceed the effect of feasibility-seeking becomes stronger. Further, Gurobi outperformed both of the other simplex implementations, which is to be expected by a commercial product.

Nevertheless, we can see that Gurobi, too, needed more and more time until termination with increasing dimension and eventually the superiorization algorithm terminated quicker. Summarizing, our first finding is, that superiorization reaches its termination criterion considerably faster than the simplex implementations in higher dimensions. This shows that superiorization can produce better results than the simplex implementations, when both are terminated at a certain time, that is before the usual termination criterion is met. Furthermore, based on our computational experiments, we see that the superiorization methodology is quite robust with respect to increasing dimensions.

In particular, the runtime remains relatively small when compared to the scipy implementations of the simplex algorithm.

4.2. Immunity to increased condition numbers. We move on to the evaluation of the experiments with increasing condition numbers. Figures 3a – 4c demonstrate well the robustness with respect to increasing condition numbers. On the left we see the averaged runtimes of 5 problem instances for each algorithm, plotted against the condition numbers, on the right we see the averaged objective function values at termination, plotted against the condition numbers.

Notice the severely sub-optimal objective function value of scipy’s simplex for high condition numbers. This explains the low runtime as the algorithm realizes that these problems are hard and quickly “surrenders”.

The trend continues in Figures 4a – 4c and the revised simplex too starts to reach its limits. Superiorization on the other hand proves to be quite stable in terms of its runtime with regards to increasing condition numbers, at times surpassing the Gurobi implementation in both runtime and objective function value. While the total runtime *does* increase significantly with the condition number, it remains within reasonable time frames, considering the increased difficulty of the problems that accompanies higher condition numbers.

Next, we take a look at the infeasibility at termination (Figures 5a–5f). Notice how with increasing condition number the general trend is towards higher infeasibility, except for superiorization, which remains constant throughout. Like before, scipy.simplex and scipy.revised simplex turn out to be unreliable in moderate dimension already. Additionally, we see that the interior point method has trouble attaining the desired level of feasibility.

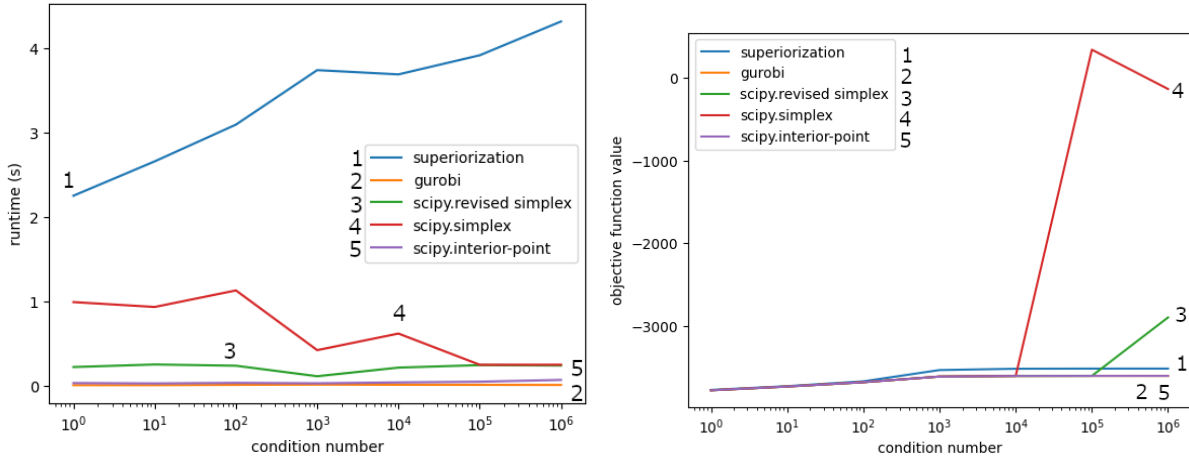
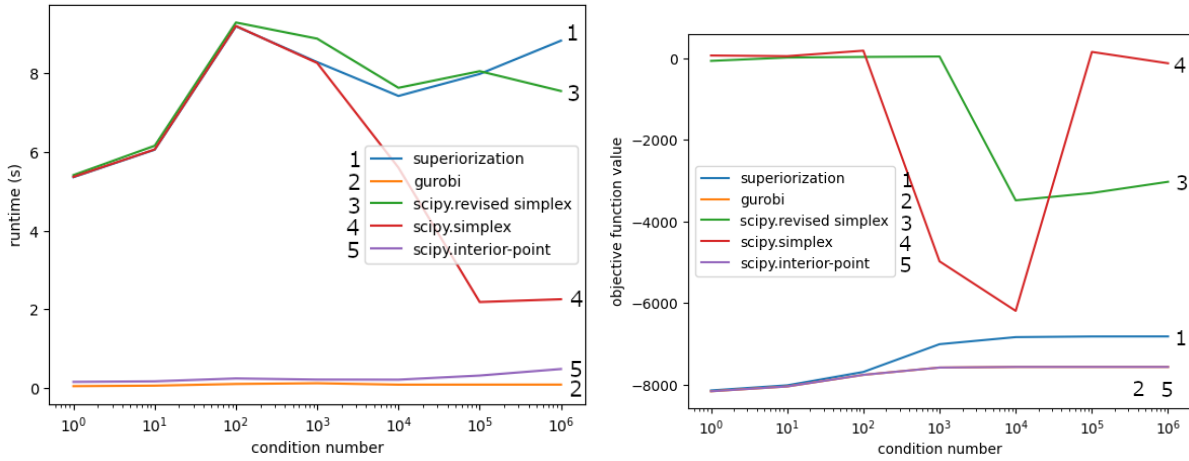
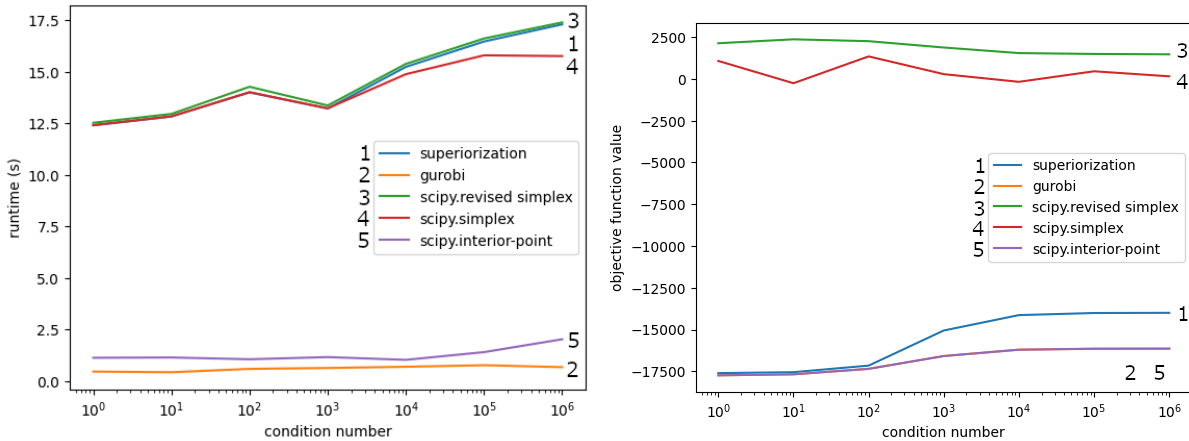
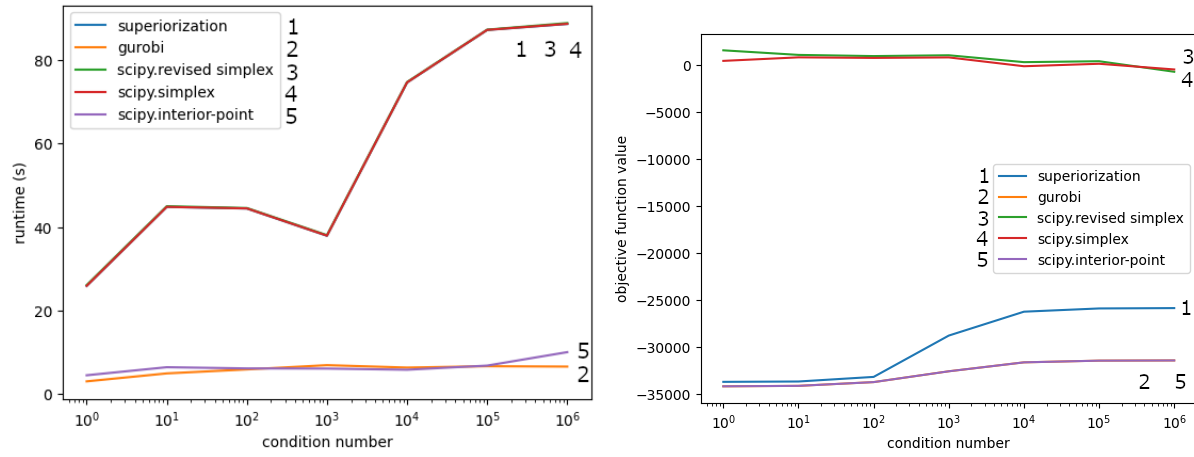
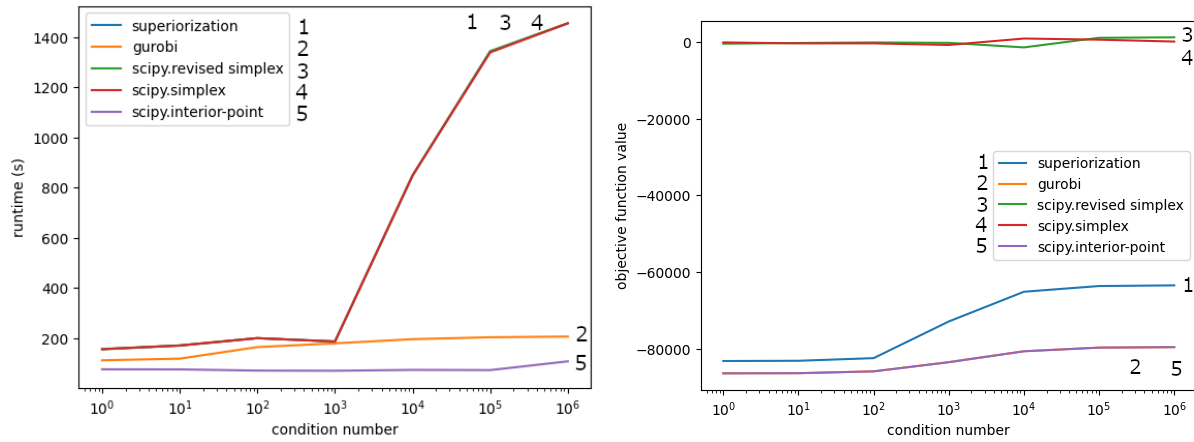
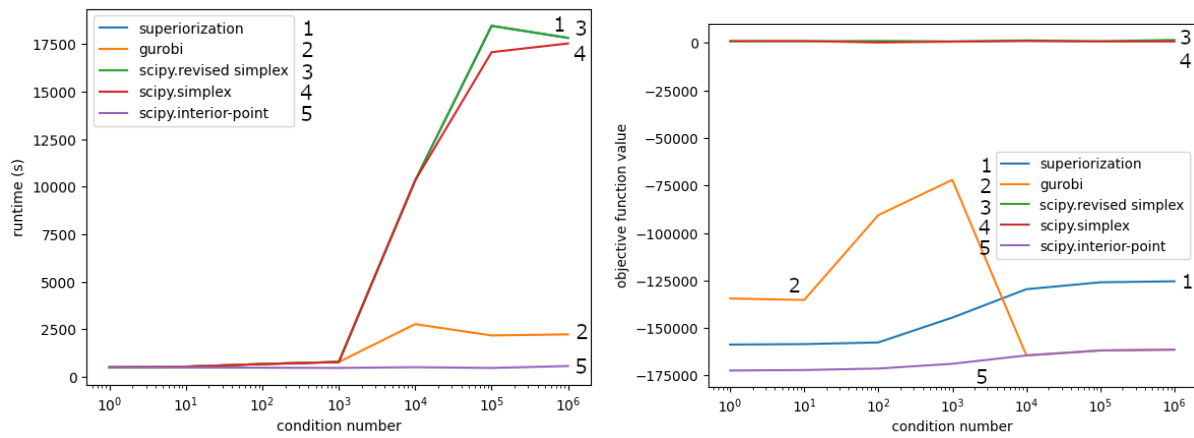
(A) 80×100 (B) 200×250 (C) 400×500

FIGURE 3. Behavior of the algorithms at fixed problem sizes $80 \times 100 - 400 \times 500$ and increasing condition numbers.

(A) 800×1000 (B) 2000×2500 (C) 4000×5000 FIGURE 4. Behavior of the algorithms at fixed problem sizes 800×1000 – 4000×5000 and increasing condition numbers.

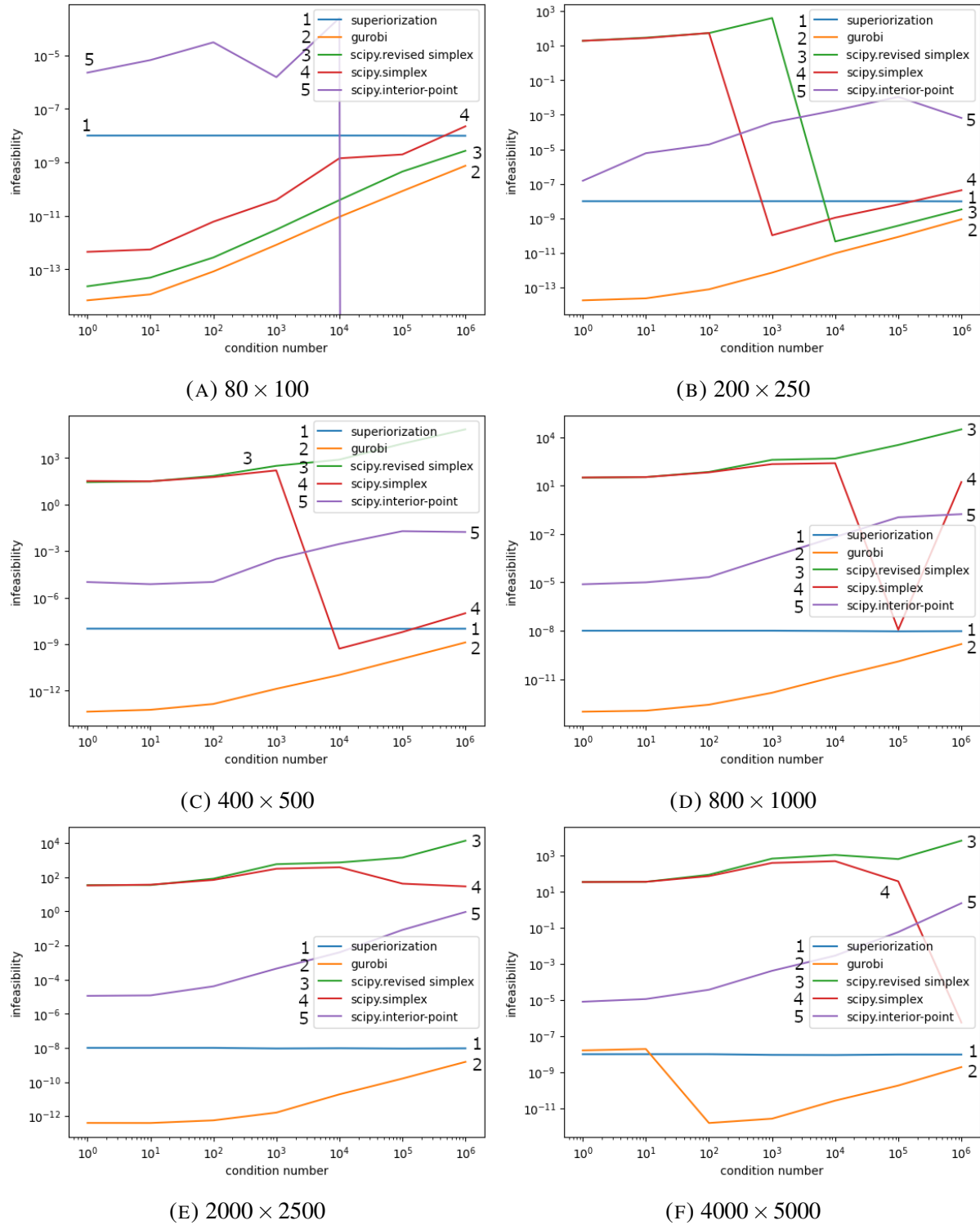


FIGURE 5. Average infeasibilities at termination of the problem instances.

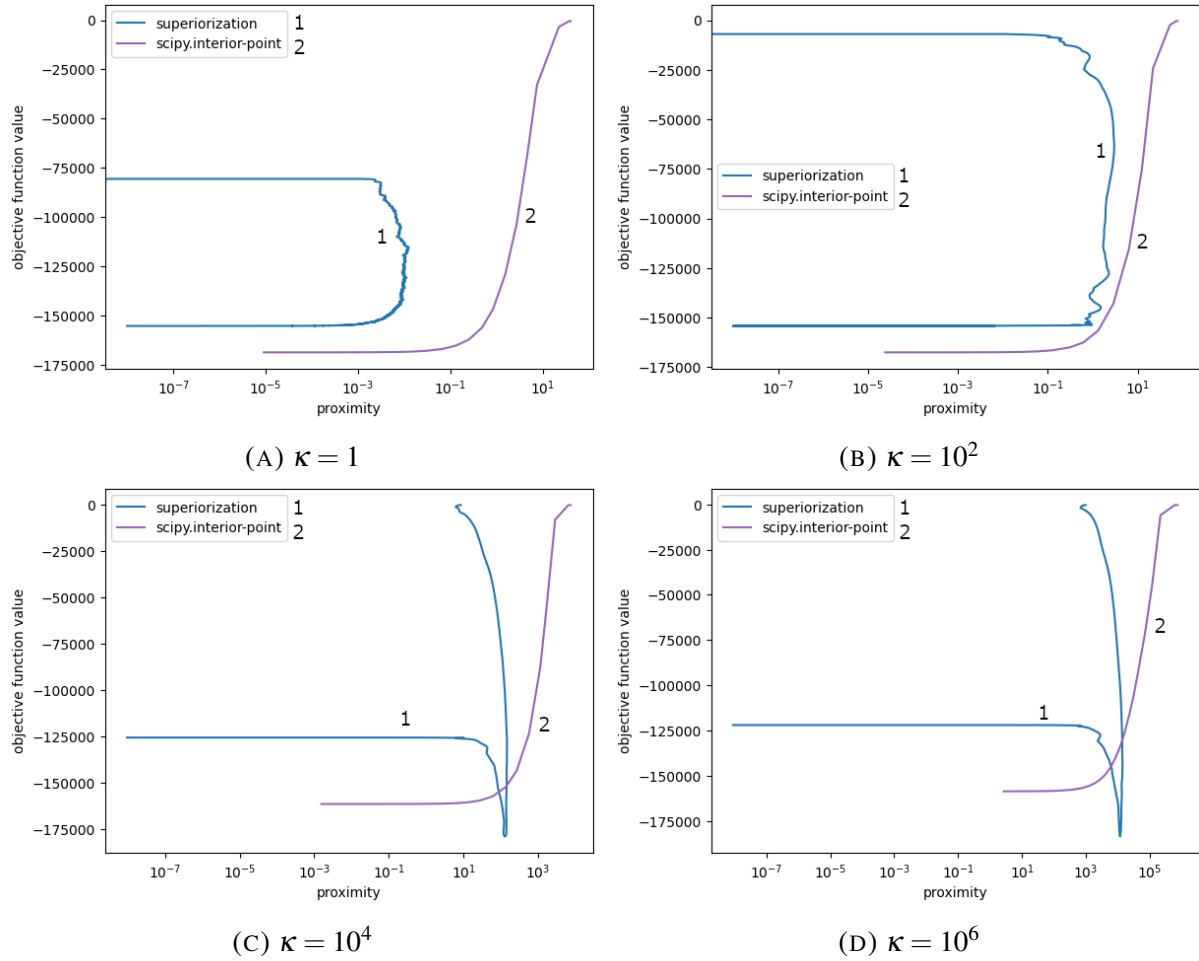


FIGURE 6. Proximity-Target-Curves of superiorization and of the interior-point method for multiple condition numbers. Notice once again the severe infeasibility (i.e., proximity) of some of the final iterates of the interior-point method.

We can see values of infeasibility of magnitudes up to $10^0 - 10^1$, significantly increasing with the condition number. In other words, while the interior point method seems to be able to quickly find the correct objective function value, it has issues doing so in a feasible manner, making it unreliable for practical applications with high condition numbers.

On the other hand, superiorization consistently reaches precisely the required level of feasibility, regardless of the condition number. This becomes particularly evident when looking at the proximity-target-curves, i.e., the plot of the level of infeasibility against the objective function value for all iterates of the respective algorithms. For further information on proximity-target curves for evaluating performance of iterative algorithms see [35, Section 4]. Figures 6a-6d showcase the behavior of the interior point method as well as of superiorization.

Hence, our second finding is, that the superiorization methodology is able to easily handle large condition numbers, especially when compared to the scipy implementations. We conjecture that this is due to the fact, that our superiorization implementation never considers the full problem at once, but performs individual projections onto the half-spaces instead. This comes at the cost of sacrificing feasibility during early iterations.

Furthermore, we conjecture, that the bounded perturbation resilience of the basic algorithm may play a role in absorbing errors that occur during computation, which would normally be amplified by the condition number.

This would mean that superiorization, in general, may be less affected by high condition numbers, than other current algorithms.

We observe that in our experiments, in terms of runtime and objective function value, at first glance the interior point method seemed to outperformed the other algorithms. This can be explained in the following way:

The condition number of a matrix can be interpreted as a measure of how linearly dependent its rows or columns are. A well-conditioned matrix (i.e., $\kappa = 1$) only has a single singular value and will be semi-orthogonal, whereas an ill-conditioned matrix (κ “large”) will have “almost” linearly dependent entries. Consequently, the half-spaces $\{x \in \mathbb{R}^n : \langle a^i, x \rangle \leq b_i\}$ will be almost parallel and the resulting polyhedron will consist of many facets and vertices.

A basic simplex implementation, which moves from vertex to vertex, will consequently face a long runtime. The interior point method, on the other hand, is not dependent on the vertices. It will take its path through the interior of the polyhedron regardless of its boundary.

However, as stated above, scipy’s interior point method seems to have trouble finding a feasible point at all for larger condition numbers, showcasing the strength of superiorization.

Combining all of the above observations, superiorization clearly offers a valuable contribution to large-scale optimization problems of high condition numbers. For large-enough problem instances it is faster than scipy’s simplex methods and more reliable than scipy’s interior point method in terms of feasibility.

5. CONCLUSIONS

In this paper, we evaluated experimentally the superiorization method (SM) and constrained optimization algorithms on a set of exemplary linear problems with varying condition numbers and varying problem sizes with the aim of investigating and comparing their immunity to increasing condition numbers. These results expand on the results in [1] and further show the strengths and weaknesses of linear superiorization and classical LP optimization algorithms. Our experimental results are promising for the observed problem sizes, and we are confident that the trend that we observed will continue for larger problem instances.

The superiorization method and constrained optimization algorithms use the same input data which consists of a family of constraints obtained from the modeling process along with a user-chosen objective function. But the two approaches aim at different end-points of their iterative processes. The easy accessibility of the superiorization methodology allows for quick implementations with the advance knowledge that the aim is not to reach a constrained optimum. At the same time the SM may compute its solutions at a lower runtime (in the case of simplex) and give better feasibility than the interior-point method.

As superiorization is a relatively new concept, we expect that, with further tuning of its parameters it will continue to find a place as a computational model and tool in situations in which users do not wish to invest efforts in seeking a constrained optimal point but rather wish to find a feasible point which is “superior” in the sense of having a smaller or equal objective function value than that of a feasible point reached by the same feasibility-seeking algorithm.

Another interesting point is that, as is well-known, interior point methods reach their performance limits for ill-conditioned nonlinear problems (e. g., in intensity-modulated radiotherapy). It would be interesting to compare the superiorization methodology in a nonlinear setting, with a different basic feasibility-seeking algorithm, to the interior point method to see if superiorization can contribute to solving these problems faster.

Acknowledgements

The authors thank the anonymous referee for valuable comments and suggestions, which helped improve the presentation and clarity of the paper. The work of the second author was supported by U.S. National Institutes of Health Grant Number R01CA266467 and by the Cooperation Program in Cancer Research of the German Cancer Research Center (DKFZ) and Israel's Ministry of Innovation, Science and Technology (MOST).

REFERENCES

- [1] Y. Censor, Can linear superiorization be useful for linear optimization problems? *Inverse Problems*, 33 (2017), 044006.
- [2] Y. Censor, Superiorization: The asymmetric roles of feasibility-seeking and objective function reduction, *Applid Set-Valued Analysis and Optimization*, 5 (2023), 325–346.
- [3] F. Barkmann, Y. Censor, and N. Wahl, Superiorization as a novel strategy for linearly constrained inverse radiotherapy treatment planning, *Frontiers in Oncology*, 13 (2023), 1–15.
- [4] Y. Censor, R. Davidi, G. T. Herman, R. W. Schulte, and L. Tetrushvili, Projected subgradient minimization versus superiorization, *Journal of Optimization Theory and Applications*, 160 (2014), 730–747.
- [5] J. Fink, Fixed Point Algorithms and Superiorization in Communication Systems, PhD thesis, Fakultät IV-Elektrotechnik und Informatik, Technische Universität Berlin, Germany, 2022.
- [6] H. A. Simon, Rational choice and the structure of the environment, *Psychological Review*, 63 (1956), 129–138.
- [7] Y. Jia, N. McMichael, P. Mokarzel, B. Thompson, D. Si, and T. Humphries, Superiorization-inspired unrolled sart algorithm with u-net generated perturbations for sparse-view and limited-angle ct reconstruction, *Physics in Medicine & Biology*, 67 (2022), 245004.
- [8] T. Nikazad and M. Abbasi, Biased superiorization of steepest descent: Redefining the reconstruction target in noisy inverse problems, *Electronic Transactions on Numerical Analysis*, 65 (2026), 93–109.
- [9] S. Agmon, The relaxation method for linear inequalities, *Canadian Journal of Mathematics*, 6 (1954), 382–392.
- [10] T. S. Motzkin and I. J. Schönberg, The relaxation method for linear inequalities, *Canadian Journal of Mathematics*, 6 (1954), 393–404.
- [11] Y. Censor and S. A. Zenios, *Parallel Optimization: Theory, Algorithms, and Applications*, Oxford University Press, New York, 1997.
- [12] K. G. Murty, *Linear Programming*, Wiley & Sons Ltd., Hoboken, 1991.
- [13] Y. Censor and E. Levy, An analysis of the superiorization method via the principle of concentration of measure, *Applied Mathematics & Optimization*, 83 (2021), 2273–2301.
- [14] A. Pyzara, B. Bylina, and J. Bylina, The influence of a matrix condition number on iterative methods' convergence. 2011 Federated Conference on Computer Science and Information Systems, pp. 459–464, 2011.
- [15] A. K. Cline, C. B. Moler, G. W. Stewart, and J. H. Wilkinson, An estimate for the condition number of a matrix, *SIAM Journal on Numerical Analysis*, 16 (1979), 368–375.
- [16] N. J. Higham, A survey of condition number estimation for triangular matrices, *SIAM Review*, 29 (1987), 575–596.
- [17] M. Benzi, Preconditioning techniques for large linear systems: A survey, *Journal of Computational Physics*, 182 (2022), 418–477.

- [18] G. T. Herman, Problem structures in the theory and practice of superiorization, *Journal of Applied and Numerical Optimization*, 2 (2020), 71–76.
- [19] Y. Censor, Superiorization and perturbation resilience of algorithms: A bibliography compiled and continuously updated, Available at: <http://math.haifa.ac.il/yair/bib-superiorization-censor.html>.
- [20] Y. Censor, G. T. Herman, and M. Jiang, Superiorization: Theory and applications, *Inverse Problems*, 33 (2017), 040301.
- [21] A. Gibali, G. T. Herman, and C. Schnörr, Superiorization versus constrained optimization: Analysis and applications, *Journal of Applied and Numerical Optimization*, 2 (2020), 1–2.
- [22] E. Bonacker, A. Gibali, K.-H. Küfer, and P. Süss, Speedup of lexicographic optimization by superiorization and its applications to cancer radiotherapy treatment, *Inverse Problems*, 33 (2017), 044012.
- [23] M. Abbasi and T. Nikazad, Bounded perturbations resilient iterative methods for linear systems and least squares problems: operator-based approaches, analysis, and performance evaluation, *Bit-Numerical Mathematics*, 64 (2024), 15.
- [24] M. Guenter, S. Collins, A. Ogilvy, W. Hare, and A. Jirasek, Superiorization versus regularization: A comparison of algorithms for solving image reconstruction problems with applications in computed tomography, *Medical Physics*, 49 (2022), 1065–1082.
- [25] G. T. Herman and R. Davidi, Image reconstruction from a small number of projections, *Inverse Problems*, 24 (2008), 045011.
- [26] V. Martín-Márquez, S. Reich, and S. Sabach, Iterative methods for approximating fixed points of Bregman nonexpansive operators, *Discrete and Continuous Dynamical Systems - Series S*, 6 (2012), 1043–1063.
- [27] M. J. Schrapp and G. T. Herman, Data fusion in x-ray computed tomography using a superiorization approach, *Review of Scientific Instruments*, 85 (2014), 053701.
- [28] K. Ma, N. Sahinidis, S. Rajagopalan, S. Amaran, and S. J. Bury, Decomposition in derivative-free optimization, *Journal of Global Optimization*, 81 (2021), 269–292.
- [29] H. H. Bauschke and J. M. Borwein, On projection algorithms for solving convex feasibility problems, *SIAM Review*, 38 (1996), 367–426.
- [30] H. Bauschke and V. Koch, Projection methods: Swiss army knives for solving feasibility and best approximation problems with halfspaces, *Contemp. Math.* 636 (2015), 1–40.
- [31] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, Twenty-fifth Anniversary Edition, SIAM, Philadelphia, 2022.
- [32] P. Deufllhard and A. Hohmann, *Numerische Mathematik 1*, De Gruyter, Berlin, 2008.
- [33] J. Bisgard, *Analysis and Linear Algebra: The Singular Value Decomposition and Applications*, The American Mathematical Society, 2021.
- [34] F. J. Aragón-Artacho, Y. Censor, A. Gibali, and D. Torregrosa-Belén, The superiorization method with restarted perturbations for split minimization problems with an application to radiotherapy treatment planning, *Applied Mathematics and Computation*, 440 (2023), 127627.
- [35] Y. Censor, E. Garduño, E. Helou, and G. Herman, Derivative-free superiorization: principle and algorithm, *Numerical Algorithms*, 88 (2021), 227–248.