


Article

On Implementing a Two-Step Interior Point Method for Solving Linear Programs

Sajad Fathi Hafshejani , Daya Gaur  and Robert Benkoczi

Department of Math and Computer Science, University of Lethbridge, Lethbridge, AB T1K 3M4, Canada; robert.benkoczi@uleth.ca

* Correspondence: sajad.fathihafshejan@uleth.ca (S.F.H.); daya.gaur@uleth.ca (D.G.)

Abstract: A new two-step interior point method for solving linear programs is presented. The technique uses a convex combination of the auxiliary and central points to compute the search direction. To update the central point, we find the best value for step size such that the feasibility condition is held. Since we use the information from the previous iteration to find the search direction, the inverse of the system is evaluated only once every iteration. A detailed empirical evaluation is performed on NETLIB instances, which compares two variants of the approach to the primal-dual log barrier interior point method. Results show that the proposed method is faster. The method reduces the number of iterations and CPU time(s) by 27% and 18%, respectively, on NETLIB instances tested compared to the classical interior point algorithm.

Keywords: linear programming; interior point method; Newton method

1. Introduction

1.1. Background

The following linear optimization (LO) problem is considered:

$$(P) \quad \min\{c^T x : Ax = b, x \geq 0\},$$

and the corresponding dual

$$(D) \quad \max\{b^T y : A^T y + s = c, s \geq 0\},$$

where $A \in \mathbb{R}^{m \times n}$, $x, c, s \in \mathbb{R}^n$ and $y, b \in \mathbb{R}^m$.

LO is an essential tool used widely in theoretical and practical science and engineering domains. Its applications extend to various fields such as operations research, engineering, economics and combinatorics. LO finds numerous applications across various fields. For instance, it is used in \mathcal{L}_1 -regularized support vector machines (SVMs) [1], basis pursuit (BP) problems [2], sparse inverse covariance matrix estimation (SICE) [3], non-negative matrix factorization (NMF) [4] and maximum a posteriori (MAP) inference [5].

The Interior-Point Method (IPM) is a powerful optimization technique that has been widely used in the field of LO. It offers an alternative approach to solving LO problems compared to traditional simplex methods. Karmarkar [6] described the first efficient algorithm for the IPM. Nesterov and Nemirovskii [7] introduced a class of self-concordant barrier functions and extended the algorithm to a more general type of optimization problems, including convex programming, non-linear complementarity problem (NCP), semidefinite optimization (SDO) and second-order cone optimization (SOCO) problems.

1.2. Literature Review

IPMs are generally divided into two categories, that is, small-update methods and large-update methods [8]. In practice, large-update methods are significantly more efficient.



Citation: Fathi Hafshejani, S.; Gaur, D.; Benkoczi, R. On Implementing a Two-Step Interior Point Method for Solving Linear Programs. *Algorithms* **2024**, *17*, 303. <https://doi.org/10.3390/a17070303>

Academic Editors: Shuai Li and Dunhui Xiao

Received: 3 June 2024

Revised: 2 July 2024

Accepted: 5 July 2024

Published: 8 July 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

However, in theory, the best results are associated with small-update methods. From another perspective, IPMs are divided into feasible and infeasible methods [9]. Feasible methods start from a point that satisfies the constraints and seek to find the optimal solution within the feasible region. In contrast, infeasible methods begin with a point that does not meet the problem's constraints (infeasible). The algorithm first seeks to find a feasible point, and once a feasible point is found, it moves towards the optimal solution [9]. The IPM based on the kernel function is a popular method. The kernel function is used not only to find the search direction for the algorithm but also to estimate a lower bound for the step size [10]. In this type of IPM, the complexity bounds of the interior point algorithm are computed based on the kernel function. Terlaky et al. [10] proposed a new variant of IPMs for solving linear optimization problems in which the self-regular barrier function replaces the logarithmic barrier function and they showed that their method has the best-known complexity bounds. Following this, several different kernel functions with the best-known complexity bounds have been introduced, including the exponential kernel functions [11], trigonometric kernel functions [12,13] and hyperbolic kernel functions [14,15].

Predictor-corrector methods for interior point algorithms are an effective approach for solving both linear and nonlinear optimization problems [16,17]. These methods incorporate two main steps, i.e., the predictor step and the corrector step. In the predictor step, an approximate direction is determined by solving a linearized system of the Karush–Kuhn–Tucker (KKT) conditions [17]. In the subsequent corrector step, this direction is refined to ensure that the iterates remain within the feasible region and maintain the balance between the primal and dual variables. This refinement typically involves solving another system of linear equations that corrects any infeasibilities introduced in the predictor step, thereby enhancing the convergence properties of the algorithm. Mehrotra's algorithm [17] is a popular predictor-corrector algorithm used for solving optimization problems. In recent years, several new predictor-corrector algorithms have been introduced, further enhancing the effectiveness and efficiency of IPMs for various optimization problems [18,19].

The Newton method is a popular optimization technique that plays a vital role in interior-point methods. In the context of IPMs, the Newton method is used to solve the system of equations that arise from the Karush–Kuhn–Tucker (KKT) optimality conditions, which include both the primal and dual feasibility conditions, as well as the complementary slackness conditions. This approach is known for its fast convergence rate, often exhibiting quadratic convergence. A modification of Newton's method was proposed by McDougall and Wotherspoon [20] that can significantly reduce the number of iterations when used to the root of functions. Their approach has a convergence of the order of $1 + \sqrt{2}$. To compute the search direction, their algorithm uses an auxiliary point and computes the gradient of the objective function only once in each iteration.

McDougall and Wotherspoon [20] applied this method to solve the power flow problem, which is a multivariate problem. They used the improved method to find the roots of a system of multivariable equations and demonstrated that it can reduce the number of iterations compared to Newton's method. Additionally, Argyros et al. [21] present some theoretical results related to the convergence of the method. In Example 4 of [21], it is shown that the method is effective for solving multivariable problems and can converge to the optimal solution.

1.3. Contribution

Motivated by these ideas, this paper makes the following contributions:

- To solve optimization problems using the IPM, the search direction plays an important role. We use the idea proposed in [20] for determining the search direction. The main goal of this paper is to extend the strategy proposed in [20] to IPMs. This paper introduces a two-step method that calculates the inverse of the matrix only once to determine the search direction in each iteration.

The next goal of this article is to demonstrate how this method can be integrated with the feasible interior-point algorithm for linear optimization problems to enhance the

efficiency of the IPM. We combine the technique described in [20] with the algorithm presented in [8], resulting in the development of two new algorithms.

- To demonstrate the efficiency of the new method, we implement the algorithms and solve a collection of NETLIB test problems. The results show that the proposed method can improve the algorithm’s efficiency by 27% in terms of the number of iterations and 18% in terms of CPU time.

The implementation on a Graphics Processing Unit (GPU), a specialized processor designed to accelerate graphics rendering and computational tasks, shows a reduction in the number of iterations and GPU time compared to the classical algorithm by 39% and 30%, respectively.

The paper is organized as follows: Section 2 briefly recalls the basics of the IPMs and the central path for solving linear programs. We explain the two-step algorithm proposed here next. Section 3 reports the numerical results on a collection of LP instances from NETLIB for CPU and GPU implementations of the algorithm. We end with concluding remarks in Section 4.

In this paper, we use the following notational conventions. The Euclidean norm of a vector is denoted by $\|\cdot\|$. The non-negative and positive orthants are represented by \mathbb{R}_+^n and \mathbb{R}_{++}^n respectively. The vectors xs and $\frac{x}{s}$ indicate coordinate-wise operations on the vectors x and s , with components $x_i s_i$ and $\frac{x_i}{s_i}$ respectively. We also use diagonal matrices X and S with entries x and s respectively. For any variable v belonging to $\{x, y, s\}$, subscripts denote the n^{th} iteration value. For instance, x_n refers to the value of x after n updates. If the current iteration is n , we use v instead of v_n and v_+ instead of v_{n+1} for $v \in \{x, y, s, \tilde{x}, \tilde{y}, \tilde{s}, \hat{x}, \hat{y}, \hat{s}\}$. Here, x refers to the current point and x_+ refers to the updated value of x .

2. Algorithm

In this paper, without loss of generality, we make the following assumptions:

- A is a full rank matrix, that is $\text{rank } A = m \leq n$.
- Both problems (P) and (D) satisfy the Interior Point Condition (IPC) [8], i.e., there exists $(x^0, s^0) > 0$ and y^0 so that:

$$\begin{aligned} Ax^0 &= b, & x^0 &> 0, \\ A^T y^0 + s^0 &= c, & s^0 &> 0. \end{aligned}$$

An optimal solution for (P) and (D) can be found by solving the following system [10]:

$$\begin{aligned} Ax &= b, & x &\geq 0, \\ A^T y + s &= c, & s &\geq 0, \\ xs &= 0, \end{aligned} \tag{1}$$

where xs stands for the coordinate-wise (Hadamard) product of the vectors x and s . Note that the first and second equations in (1) guarantee the feasibility of x and (y, s) for problems (P) and (D), respectively, and the last equation is satisfied only if the feasible solution is optimal (complementarity conditions).

The algorithm presented here is a two-step primal-dual IPM; therefore, we summarize the key idea behind the generic primal-dual IPMs for solving linear programming problems from the literature, see e.g., [8,10]. The key idea is to replace the complementarity conditions with parameterized equations $xs = \mu e$, where e denotes the all-one vector of length n and μ is a real positive parameter. With these changes, the system of Equation (1) can be expressed as [10]:

$$\begin{aligned} Ax &= b, & x &\geq 0, \\ A^T y + s &= c, & s &\geq 0, \\ xs &= \mu e. \end{aligned} \tag{2}$$

The parametric system (2) has a unique solution for any $\mu > 0$ if the IPC holds [10] (Theorem 1.2.4). For a given $\mu > 0$, denote the unique solution of (2) by $(x(\mu), y(\mu), s(\mu))$, where $x(\mu)$ is the μ -center of (P) and $(y(\mu), s(\mu))$ is called the μ -center of (D). The set of all μ -centers, for all $\mu > 0$, defines a path the so-called *central path* for (P) and (D). It has been shown that, as $\mu \rightarrow 0$, then the limit of the central path exists and converges to an analytic center of the optimal solutions set of (P) and (D) [22,23].

Given a current point (x, y, s) , the goal is to find direction $(\Delta x, \Delta y, \Delta s)$ such that the point $(x + \Delta x, y + \Delta y, s + \Delta s)$ lies on the central path at μ . If it did lie exactly on the central path, then Equation (2) will be satisfied, i.e., [10]:

$$\begin{aligned} A(x + \Delta x) &= b \\ A^T(y + \Delta y) + (s + \Delta s) &= c \\ (X + \Delta x)(S + \Delta s) &= \mu e, \end{aligned} \tag{3}$$

where:

$$X := \text{diag}(x), \quad S := \text{diag}(s)$$

Assuming that the point (x, y, s) is feasible for the LO and dropping the non-linear term $\Delta x \Delta s$ [10], we get the following system of equations:

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y + \Delta s &= 0 \\ S\Delta x + X\Delta s &= \mu e - XSe \end{aligned} \tag{4}$$

System (4) has a unique solution, referred to as the search direction for IPMs. Many interior-point algorithms rely on the Newton method to solve the system (4) and find the search direction. Therefore, improving the speed of convergence of Newton’s method significantly impacts the convergence speed and efficiency of IPMs. To calculate the search direction in Newton’s method, we use the method proposed in [20], which has a convergence speed of $1 + \sqrt{2}$. Accordingly, we introduce a two-step method for the IPM based on the approach proposed in [20]. This change can significantly reduce the number of iterations needed to achieve the optimal solution. To apply the mentioned method, we present the system (4) as a problem of finding roots, followed by an explanation of the new approach.

2.1. Relationship with Root Finding

Define:

$$\zeta = \begin{bmatrix} x \\ y \\ s \end{bmatrix} \text{ and } F(\zeta) = \begin{bmatrix} Ax - b \\ A^T y + s - c \\ \mu e - XSe \end{bmatrix},$$

then a solution to the system (2) is ζ^* such that $F(\zeta^*) = 0$. The root ζ^* can be computed using Newton’s method (in the neighbourhood of the root), which, given ζ , finds $\Delta\zeta$ such that $F(\zeta + \Delta\zeta) = 0$ [20]. Using Taylor series approximation (linear approximation) around ζ , we have [20]:

$$F(\zeta) + F'(\zeta)\Delta\zeta \simeq 0$$

where:

$$\text{Jacobian } F' = \begin{bmatrix} \frac{\partial F_1}{\partial \zeta_1} & \frac{\partial F_1}{\partial \zeta_2} & \frac{\partial F_1}{\partial \zeta_3} \\ \frac{\partial F_2}{\partial \zeta_1} & \frac{\partial F_2}{\partial \zeta_2} & \frac{\partial F_2}{\partial \zeta_3} \\ \frac{\partial F_3}{\partial \zeta_1} & \frac{\partial F_3}{\partial \zeta_2} & \frac{\partial F_3}{\partial \zeta_3} \end{bmatrix} = \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \text{ and } \Delta\zeta = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix}.$$

This is precisely the system that was derived earlier (4). This approximation yields an iterative algorithm, the Newton–Raphson method, for finding roots. Starting with an initial

estimate ζ_0 , the current estimate ζ_n of the root is updated using the following rule for some appropriate step size α :

$$\zeta_{n+1} = \zeta_n - \alpha(F'(\zeta_n))^{-1}F(\zeta_n)$$

The proposed method generalizes the Newton–Raphson method as follows (assume any F). In the first step, an auxiliary point $\tilde{\zeta}_n$ is computed using a linear approximation to the function around the current estimate of the root ζ_n . The average $\hat{\zeta}_{n+1}$ of the auxiliary point and the current estimate is computed in the second step. In the third step, the current estimate is updated using the linear approximation to F around ζ_n , but instead of using the Jacobian at point ζ_n , the Jacobian is evaluated at the average $\hat{\zeta}_{n+1}$. The update rules used in n^{th} iteration can be summarized as [20]:

$$\begin{aligned} \tilde{\zeta}_{n+1} &= \zeta_n - \alpha(F'(\hat{\zeta}_n))^{-1}F(\zeta_n) \\ \hat{\zeta}_{n+1} &= \frac{1}{2}(\tilde{\zeta}_{n+1} + \zeta_n) \\ \zeta_{n+1} &= \zeta_n - \alpha(F'(\hat{\zeta}_{n+1}))^{-1}F(\zeta_n) \end{aligned}$$

This sequence of updates requires two evaluations of the Jacobian, once at point $\hat{\zeta}_n$ and the second time at point $\hat{\zeta}_{n+1}$. This requirement doubles the computation burden. In practice, we use information regarding $F'(\hat{\zeta}_{n+1})$ for the next iteration. Thus only a single evaluation of the Jacobian is needed in each iteration if we save the result from the previous iteration.

2.2. Two-Step IPMs

Here, we extend the idea presented in Section 2.1 to the interior-point algorithm. The extension involves two phases:

- Phase 1: We obtain a feasible starting point (x, y, s) using the method presented in [17]. Then, we create an auxiliary point $(\tilde{x}, \tilde{y}, \tilde{s})$, which initially has the same values as (x, y, s) . The parameter μ is updated by a factor of $1 - \theta$, and $(\tilde{x}, \tilde{y}, \tilde{s})$ is set to (x, y, s) . After that, we set $(\hat{x}, \hat{y}, \hat{s}) = \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$ and solve the following system to determine the search direction:

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y - \Delta s &= 0 \\ \hat{S}\Delta x + \hat{X}\Delta s &= \mu\mathbf{e} - XSe \end{aligned} \tag{5}$$

To satisfy the feasibility condition, the algorithm performs a line search to find the maximum step size. To obtain the new point, the current point is updated in the following way:

$$(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$$

- Phase 2: In this phase, starting from $k \geq 1$, we begin by updating the parameter μ and constructing the auxiliary point. Then, we utilize it to create the principal point. As in Phase 1, we solve the following system to determine the search direction for the auxiliary point:

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y - \Delta s &= 0 \\ \hat{S}\Delta x + \hat{X}\Delta s &= \mu\mathbf{e} - XSe. \end{aligned} \tag{6}$$

Since the left-hand side of (6) is the same as the left-hand side of (5), so there is no need to compute the inverse. Next, a line search is performed to determine the maximum step size, and the auxiliary point is updated as follows:

$$(\tilde{x}, \tilde{y}, \tilde{s}) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$$

Next, the middle point is updated as:

$$(\hat{x}, \hat{y}, \hat{s}) \leftarrow \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$$

Finally, the algorithm updates the principal point using the search direction $(\Delta x, \Delta y, \Delta s)$ is obtained by solving the following system:

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y - \Delta s &= 0 \\ \hat{S}\Delta x + \hat{X}\Delta s &= \mu\mathbf{e} - XSe \end{aligned} \tag{7}$$

As earlier, to ensure that the principal point satisfies the feasibility condition, a line search is performed to determine the maximum step size. The principal point is then updated as follows:

$$(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s).$$

In order to determine the maximum value for the step size α in each iteration, we utilize the method presented by Cai et al. [24]. This method initially calculates an upper bound based on the positivity constraints of the variables. If $(x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$ is not feasible, then α is decreased by a certain multiplicative factor until the point is feasible. This approach is more efficient than determining the optimal value for α , as it avoids an additional computational step. Algorithm 1 outlines the steps of this new approach.

Algorithm 1 A two-step feasible IPM algorithm for LOs

Input: $x_0, y_0, s_0, \mu, \theta, \epsilon$
 $(x, y, s) \leftarrow (x_0, y_0, s_0)$
 $(\tilde{x}, \tilde{y}, \tilde{s}) \leftarrow (x, y, s)$
 $(\hat{x}, \hat{y}, \hat{s}) \leftarrow \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$
 $\mu \leftarrow (1 - \theta)\mu$
 Find search direction $(\Delta x, \Delta y, \Delta s)$ by solving (5).
 Find the maximum value for the step size α .
 Set $(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$
while *stopping criteria is not met* **do**
 $\mu \leftarrow \mu(1 - \theta)$
 Solve (6) to find the search direction.
 Find the maximum value for the step size.
 Set $(\tilde{x}, \tilde{y}, \tilde{s}) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$.
 Set $(\hat{x}, \hat{y}, \hat{s}) \leftarrow \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$
 Find search direction $(\Delta x, \Delta y, \Delta s)$ by solving (7).
 Find the maximum value for the step size α .
 Set $(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$
end

The method used in Algorithm 1 stands out from existing interior point algorithms. It utilizes a two-step Newton method to determine the search direction, resulting in a higher convergence rate of $1 + \sqrt{2}$ compared to traditional methods. This improved rate leads to a significant reduction in the number of iterations needed by the algorithm. Additionally, unlike interior-point predictor-corrector methods that require computing the matrix's inverse twice per iteration, Algorithm 1 only needs to compute it once per iteration. Specifically, it uses the previous iteration's matrix inverse to calculate the search direction for the auxiliary point. This computational characteristic aligns Algorithm 1 with one-step methods but requires fewer iterations to converge.

Incorporating the idea proposed by Wang et al. [25], we will now implement Algorithm 2. To begin, they introduced a fresh variable v , defined below:

$$v := \sqrt{\frac{xs}{\mu}} \tag{8}$$

Next, they introduced a kernel function as $\psi(t) = \frac{(t-1)^2}{2}$. Based on this kernel function and the fact that $(x, s) > 0$, they showed that [25]:

$$v = \mathbf{e} \Leftrightarrow v^2 = \mathbf{e} \Leftrightarrow \frac{xs}{\mu} = \sqrt{\frac{xs}{\mu}}$$

Now, we use this idea and rewrite the (6) and (7) as [25]:

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y - \Delta s &= 0 \\ \hat{S}\Delta x + \hat{X}\Delta s &= \mu\mathbf{e} - \sqrt{\mu}\sqrt{XSe} \end{aligned} \tag{9}$$

and

$$\begin{aligned} A\Delta x &= 0 \\ A^T\Delta y - \Delta s &= 0 \\ \hat{S}\Delta x + \hat{X}\Delta s &= \mu\mathbf{e} - \sqrt{\mu}\sqrt{XSe} \end{aligned} \tag{10}$$

The resulting changes lead to Algorithm 2 which is described next:

Algorithm 2 A two-step feasible IPM algorithm for the algorithm proposed in [25]

Input: $x_0, y_0, s_0, \mu, \theta, \epsilon$
 $(x, y, s) \leftarrow (x_0, y_0, s_0)$
 $(\tilde{x}, \tilde{y}, \tilde{s}) \leftarrow (x, y, s)$
 $(\hat{x}, \hat{y}, \hat{s}) \leftarrow \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$
 $\mu \leftarrow (1 - \theta)\mu$
 Find search direction $(\Delta x, \Delta y, \Delta s)$ by solving (10).
 Find the maximum value for the step size α .
 Set $(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$
while *stopping criteria is not met* **do**
 $\mu \leftarrow \mu(1 - \theta)$
 Solve (9) to find the search direction.
 Find the maximum value for the step size.
 Set $(\tilde{x}, \tilde{y}, \tilde{s}) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$.
 Set $(\hat{x}, \hat{y}, \hat{s}) \leftarrow \frac{1}{2}((\tilde{x}, \tilde{y}, \tilde{s}) + (x, y, s))$
 Find search direction $(\Delta x, \Delta y, \Delta s)$ by solving (10).
 Find the maximum value for the step size α .
 Set $(x, y, s) \leftarrow (x + \alpha\Delta x, y + \alpha\Delta y, s + \alpha\Delta s)$
end

2.3. Discussion

The idea in the Algorithm is also used in [20] for finding roots of a univariate function f using a modification to the Newton–Raphson method. The method requires two steps a predictor step and a corrector step. The predictor step computes the point \tilde{x}_+ based on a linear approximation of the function around the current point x . In the corrector step, the

new point is calculated using the derivative at a point which is the average of the current point and the predictor. This can be expressed as [20]:

$$\tilde{x}_+ = x - \frac{f(x)}{f'(\frac{x+\tilde{x}}{2})}$$

If we call $1/2(x + \tilde{x}_+)$ as \hat{x} and use the derivative at \hat{x} instead at x in the predictor step, then we have the following scheme:

$$\begin{aligned} \hat{x} &= \frac{x + \tilde{x}_+}{2} \\ x_+ &= x - \frac{f(x)}{f'(\hat{x})} \end{aligned}$$

In particular applications $f'(\hat{x})$ is a good estimate of $f'(x)$. Therefore this reduces the number of derivate evaluations. If the predictor-corrector scheme was not used, then the new point is \tilde{x} , and the function value would be $f(\tilde{x}) = f(x - f(x)/f'(\hat{x}))$. Since a corrector is used, the value of f at the new point is $f(x - f(x)/f'(\hat{x}_+))$. If there is a favourable relationship between the derivatives of the predictor and the corrector points, then (assuming a minimization problem) we have:

$$f(x - f(x)/f'(\hat{x})) > f(x - f(x)/f'(\hat{x}_+)) \tag{11}$$

Suppose this inequality is valid in many iterative steps. In that case, there will be a reduction in the number of iterations compared to the Newton–Raphson method. The decrease in iterations required can be credited to the new approach’s ability to attain the optimal point of the Newton method at a rate of $1 + \sqrt{2}$. This higher convergence rate means that the new method needs fewer iterations than the traditional approach.

In general, if the function is multivariate denoted F , then the updates can be written as:

$$\begin{aligned} \tilde{x}_+ &= x - \alpha(\nabla F(\hat{x}))^{-1}F(x) \\ \hat{x}_+ &= 1/2(x + \tilde{x}_+) \\ x_+ &= x - \alpha(\nabla F(\hat{x}_+))^{-1}F(x) \end{aligned}$$

The algorithm presented in this paper is fundamentally distinct from the predictor-corrector method proposed by Mehrotra [17]. Notably, one of the main differences lies in the approach to calculating the search direction. While both algorithms leverage the two-stage search direction concept, Algorithm 1 computes the search direction by utilizing an auxiliary point and taking the average of this auxiliary point and the main point. Another significant contrast arises in the computation of the matrix inverse. Algorithm 1 requires calculating the matrix inverse only once per iteration. Consequently, it falls into the category of one-step methods in terms of time consumption, while achieving efficiency akin to two-step algorithms. These distinctions set Algorithm 1 apart from the predictor-corrector method of [17], highlighting its unique and promising characteristics in solving optimization problems. Moreover, the method in [17] requires tuning many parameters in each iteration, which can be expensive and complex for some problems to find their optimal values. In contrast, our proposed method involves far fewer parameters to tune, simplifying the process and potentially reducing the computational burden.

3. Experimental Evaluation

We use linear programming instances from NETLIB [26]. There are a total of 138 linear programming problems with varying sizes. The smallest one has 22 non-zeros and the largest one has 1,408,073 non-zeros in the coefficient matrix. There are 30 instances that are infeasible. Thirty nine of the instances are not full rank. Further, four test problems have $b = 0$. If $b = 0$, the initialization scheme [27] not produce an initial feasible solution. We do

not use these 73 instances in our experimentation. We define the absolute and relative gap as follows:

$$\begin{aligned} \text{absolute gap: } & |c^T x - b^T y| \\ \text{relative gap: } & \frac{|c^T x - b^T y|}{1 + |c^T x| + |b^T y|}. \end{aligned}$$

For all experiments, we use a relative-gap of at most 10^{-6} , at most 700 iterations as the stopping criteria. Although the absolute gap is generally more stringent than the relative-gap, achieving the absolute gap for problems with large objective function values may require many iterations. In IPMs, significant reductions occur during the initial iterations, but only small reductions happen in the final iterations. Therefore, in this article, we use the relative-gap in our experiments.

3.1. Experiments on CPU

We demonstrate the effectiveness of the new approach by executing Algorithm 1 and comparing the results to the classical interior-point algorithm (the primal-dual logarithmic barrier method Chapter 7) [8]. Note that instead of calculating the search direction using Newton's method mentioned in the classical algorithm [8], Algorithm 1 finds the search direction using the modified Newton approach. This is the primary difference between Algorithm 1 and the classical algorithm. Specifically, in each iteration, Algorithm 1 first updates the auxiliary point. It then uses the auxiliary point, along with the primary principle, to find the search direction. In contrast, the classical algorithm relies solely on the information from the principal point to determine the search direction. In this regard, we coded the three algorithms (Algorithm 1, Algorithm 2 and Classical approach) in Python 3.10 using BLAS routines for the equation-solving step in all the algorithms. Of the remaining 65 instances the three algorithms solved 50 instances within a total time limit of 24 h on alliance canada cluster CEDAR, 2 × Intel E5-2683 v4 Broadwell CPU with 125 G RAM 2.1 GHz (<https://alliancecan.ca/en>, accessed on 27 August 2023) [28]. The remaining fifteen instances large-sized instances were not completed due to the time limit for at least one of the algorithms tested.

To demonstrate the efficacy of the new method, we applied Algorithms 1 and 2 to a test problem. For each step of the two-step method, we computed the objective function. Figure 1 illustrates the difference in the objective function between the two points, \tilde{x}_+ (auxiliary point) and x_+ (principal point), during each iteration. The plot shows that the objective function value in the second stage is consistently lower than in the first stage, indicated by positive values of $c^T \tilde{x}_+ - c^T x_+$. This validates the proposed method, demonstrating its potential to reduce the number of iterations required. Additionally, it indicates that the auxiliary point effectively guides the algorithm in correcting its search direction, leading to more efficient convergence.

Next, we want to see for the same instance `lp_scsd8`, (i) the rate at which the absolute gap decreases, (ii) the decrease in the relative gap between primal and dual solution values as a function of the iterations. Figure 2 shows the two plots. Algorithms 1 and 2 achieve a faster reduction in the relative gap and the absolute gap compared to the classical approach, i.e., a lesser number of iterations are needed overall. Also note a sharp reduction in the latter set of iterations for Algorithms 1 and 2 (this phenomenon is not universal though, sometimes there is a sharp reduction in the initial iteration for some instances).

To demonstrate the effectiveness of the two-step method we observe two quantities, (i) the speedup and (ii) the relative reduction in the number of iterations achieved by Algorithms 1 and 2 compared to the classical algorithm. If Algorithm i takes time $t(A_i, j)$ on instance j , classical algorithm C takes time $t(C, j)$ on the same instance j then the speedup of Algorithm i is defined as $t(C, j)/t(A_i, j)$. Similarly, we define the relative reduction in the number of iterations. If Algorithm i takes $n(A_i, j)$ iterations on instance j , classical

algorithm C takes $n(C, j)$ iterations on the same instance j then the relative reduction of Algorithm i is defined as $n(C, j)/n(A_i, j)$.

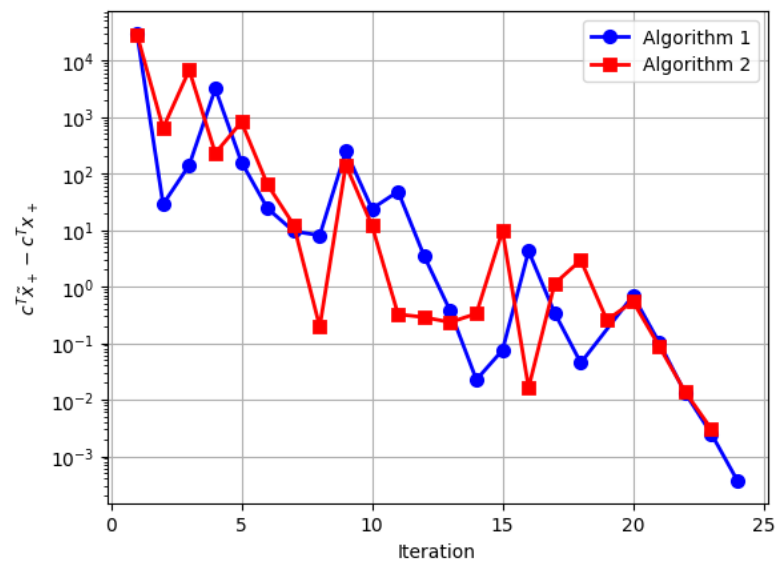


Figure 1. $c^T \tilde{x}_+ - c^T x_+$ for Algorithms 1 and 2 on instance 1p_scscd8.

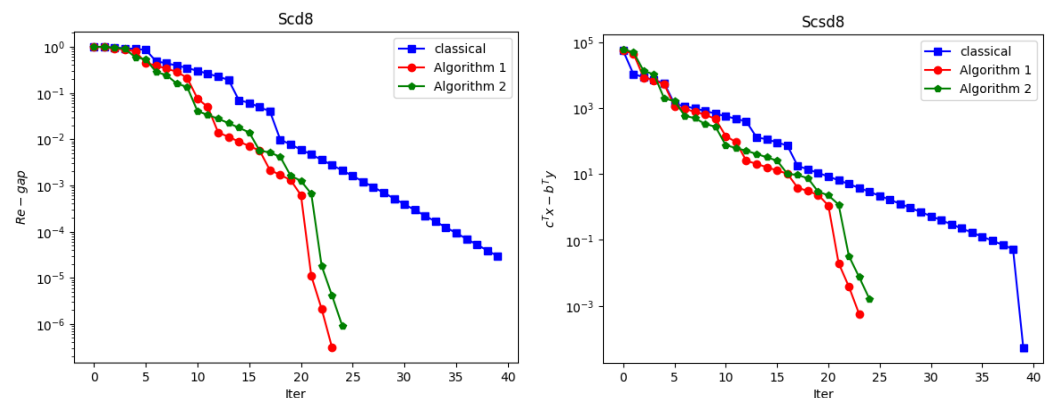


Figure 2. (Left) Relative gap, (Right) absolute gap vs. the iteration number.

Figures 3 and 4 show the relative reduction and the speedup in the number of iterations as a function of the number of instances. The instances are ordered according to the size in array. The average value of speedup, relative reduction if computed for each suffix of the array. If the x -coordinate is n then the y -coordinate is the average speedup, relative reduction for $50 - n$ largest instances. The average speedup over all 50 instances is between 1.3 and 1.4 for both Algorithms 1 and 2. The average relative reduction is between 1.20 and 1.25 for both Algorithms 1 and 2. If we focus only on 5 of the largest instances then the speedup and relative reduction are comparably higher. This indicates that the method scales well. Algorithm 1 has a greater relative reduction but it does not translate into a greater speedup. Algorithm 2 has a better speedup. The instances indexed 40–45 in the plots show anomalous behaviour, it appears that these are specific types of instances (staircase instances) and the performance of Algorithms 1 and 2 seems to be different (compared to the average) on these instances. One of the prominent features of the new proposed algorithm is its ability to find an appropriate search direction compared to traditional interior-point algorithms. As demonstrated in [20], the new modified Newton method exhibits a faster convergence rate, approximately $1 + \sqrt{2}$. Therefore, the new algorithm leverages this property, allowing it to reach the optimal solution in fewer iterations—a particularly valuable advantage when dealing with high-dimensional test problems. In

scenarios where matrix multiplication and inversion can be computationally intensive, the reduced number of iterations provided by the new algorithm significantly decreases the execution time and enhances overall performance.

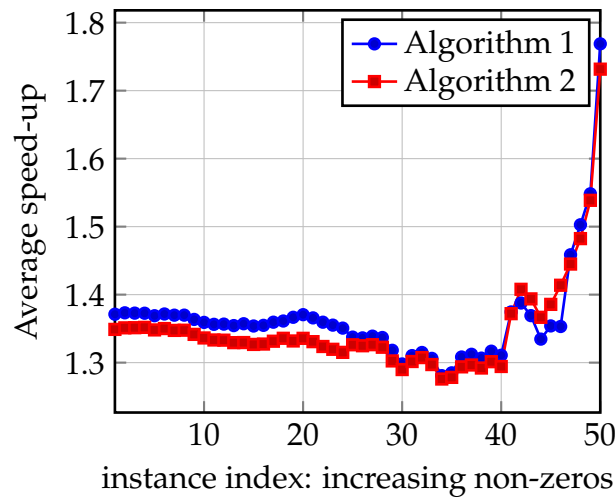


Figure 3. Relative reduction.

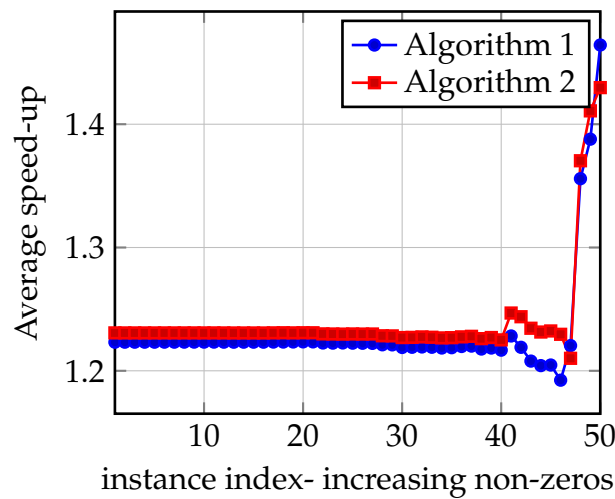


Figure 4. Speedup vs. number of instances.

In Table 1, the average number of iterations and CPU time (measured in seconds) for three different methods—Classical Algorithm, Algorithms 1 and 2—are presented. The Classical Algorithm required an average of 97.82 iterations and 166.71 s of CPU time. In contrast, Algorithm 1 demonstrated improved efficiency with an average of 71.34 iterations and 136.29 s of CPU time. Similarly, Algorithm 2 also exhibited promising results with 72.52 iterations and 135.46 s of CPU time. These findings indicate that both Algorithms 1 and 2 are more effective than the Classical Algorithm in reducing the number of iterations and CPU time. Notably, Algorithm 1 achieved the lowest number of iterations, while Algorithm 2 achieved the lowest CPU time. Moreover, the fourth column presents the standard deviation for the number of iterations for the three algorithms, indicating that Algorithm 1 has the lowest variability, followed by Algorithm 2, with the Classical Algorithm having the highest variability.

The details of the problem instances can be found in [29]. The raw data used to build these figures is in the Appendix A and is self-explanatory. The data in Appendix B is divided into blocks of three rows each. The first row in each block is data for the classical algorithm and the other two rows are the data for Algorithms 1 and 2. Table A2 lists the

relative gap and the norms and shows the convergence of the three algorithms to an optimal solution on several instances from NETLIB.

Table 1. The average number of iterations, CPU time (s) and standard deviation.

Methods	Aver. Iter.	Aver. CPU (s)	Std-Iter.
Classical Algorithm	97.82	166.71	70.96
Algorithm 1	71.34	136.29	55.64
Algorithm 2	72.52	135.46	57.55

Note: Bold numbers indicate the lowest values in each column.

3.2. Experiments on GPU

A study by Świrydowicz et al. [30] investigated the use of GPU-accelerated linear solvers for optimizing large-scale power grid problems. Motivated by their study, we evaluate the performance of the algorithms on massively parallel hardware (GPU with tensor cores) using library routines to perform factoring.

We utilized the Pytorch library to develop a GPU-enabled version of our program. The library assumes that all tensors are located on the same device. During our experiments, we utilized the NVIDIA V100 Volta (on the GRAHAM Compute Canada cluster), which has a maximum size limitation due to its 16GB HBM2 memory. This card features tensor cores that perform certain tensor operations very efficiently in hardware. GPU arithmetic is typically conducted in low precision, ranging from 8 to 32 bits which account for the speed. Unfortunately, the algorithms failed to converge to an optimal solution when we employed low-precision arithmetic on V100. As a result, we opted to use double precision at 64 bits, even though this substantially increased the run time.

To utilize GPU acceleration, we convert all vectors and matrices into tensors, leveraging the PyTorch library instead of Numpy for computations. In contrast, when running on a CPU, we represent these structures as regular vectors and matrices. By performing essential operations like matrix multiplication, inverse calculation and step size computation in tensor mode. This approach enhances the efficiency of calculations and enables faster reaching the optimal point.

The computationally expensive steps in the algorithm proposed here (as in any IPM) are (i) the determination of α and (ii) the computation of the inverse of the Jacobian to determine the step direction. The first expensive step has quadratic complexity ($O((n + m)^2)$ when implemented naturally). The second step is computationally more costly $O((n + m)^3)$. In the approach here, gains are realized asymptotically in the second step. Therefore, we focus on the time the GPU takes to perform matrix inversions. The raw data is shown in Appendix C. The second and third columns are the iterations needed by the classical and the algorithm proposed here. The last two columns in the table are the times (in seconds) needed by the GPU to perform the matrix inversions.

Results for GPU

Table A3 presents the results for Classical Algorithm and Algorithm 1 on GPU. We report the number of iterations and GPU times for 41 test problems. Note that, Classical algorithm could not reach the optimal solution for 5 test problems and Algorithm 1 could not find the optimal solution for 1 test problem based on the stopping conditions. We remove the results for these 6 test problems and calculate the relative reduction and speed-up. Figure 5 presents the relative reduction and speed-up for 35 test problems. Moreover, we denote the average of the number of iterations and GPU time for these 35 test problems in Table 2. Based on this Table, we can conclude that the new proposed method reduced the number of iterations and GPU by 39% and 30% respectively. Moreover, the standard deviation for the number of iterations indicates that Algorithm 1 has the lowest variability at 60.19.

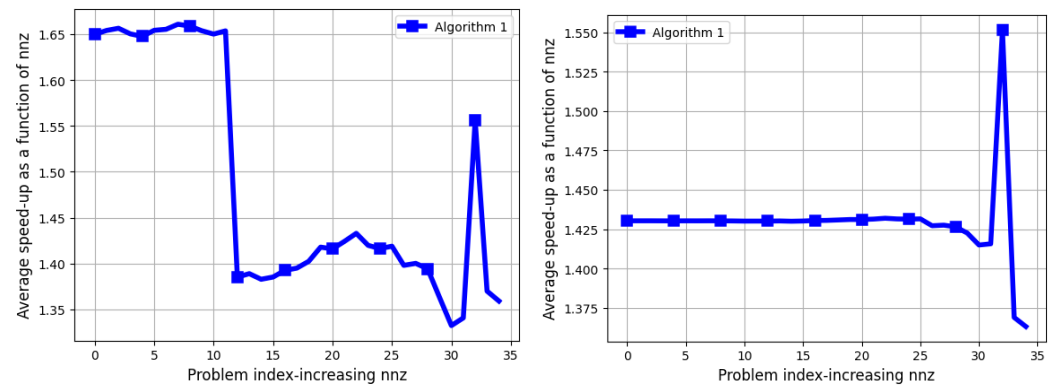


Figure 5. (Left) Relative reduction. (Right) Speedup vs. the number of instances.

Table 2. The average number of iterations, GPU time and standard deviation for the number of iterations.

Methods	Aver. Iter.	Aver. GPU (s)	Std-Iter.
Classical Algorithm	101.77	39.81	113.71
Algorithm 1	61.68	27.83	60.19

Note: Bold numbers indicate the lowest values in each column.

3.3. Discussion

Although it has not been theoretically proven that the convergence rate of the modified Newton method is exactly $1 + \sqrt{2}$, numerical results show that this rate is achievable. This rate may vary slightly depending on the structure of the problem (see example in [20]). As shown in [20], the modified Newton method can reduce the number of iterations and computational time for solving a system of equations by a factor of 1.22, calculated as $(1 + \sqrt{2})/2$, where 2 is the convergence rate of the Newton method and $1 + \sqrt{2}$ is the convergence rate of the modified Newton method. However, the average results in Tables 1 and 2 demonstrate that when we apply the modified Newton method to find the search direction in IPMs, the convergence rate of the proposed method remains close to $1 + \sqrt{2}$.

4. Conclusions

This paper presents a two-step method for optimizing linear programming problems using the interior point method. Building on a previous idea to solve the Karush–Kuhn–Tucker (KKT) conditions, we extend this approach to IPMs. Our method re-examines and enhances the classical algorithm proposed by [8] by incorporating a two-step process. In this method, the search direction is calculated using a modified Newton method that employs an auxiliary point. Like the standard Newton method, the modified Newton method computes the matrix inverse only once. We incorporate this approach into IPM and demonstrate its efficiency through implementations on both CPU and GPU, as well as by solving NETLIB test problems. The results show that the proposed method can improve efficiency by 27% in terms of iterations and 18% in terms of CPU time. These results confirm that the convergence rate of the modified Newton method is equal to $1 + \sqrt{2}$. This work is pioneering in integrating the modified Newton method into IPM. This approach has the potential to be applied to other interior point algorithms, enhancing their performance as well.

One of the key areas for future work is proving the complexity bounds of the algorithm, which could enhance the overall complexity bounds of IPMs. Additionally, demonstrating the effectiveness of the algorithm on real-world problems modeled as linear optimization problems is another important avenue for exploration. Another focus will be evaluating the algorithm’s efficiency on large datasets, which is particularly relevant given the increasing dimensions of real-world problems. Further, we plan to optimize the implementation on GPUs, which includes developing parallel implementations of the algorithm and leveraging

parallel processing to calculate the inverse of the Jacobian matrix. This work can yield significant performance improvements and provide valuable insights into the practical applications of the algorithm.

Author Contributions: Conceptualization, S.F.H. and D.G.; methodology, S.F.H., D.G. and R.B.; visualization, S.F.H. and D.G.; investigation, D.G. and R.B.; writing—original draft preparation, S.F.H. and D.G.; writing—review and editing, D.G. and R.B.; supervision, D.G. and R.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets used in this paper are available from the following link <https://sparse.tamu.edu/LPnetlib>, accessed on 25 August 2023.

Acknowledgments: This research was enabled in part by support provided by the Digital Research Alliance of Canada (<https://alliancecan.ca>, accessed on 27 August 2023).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LO	Linear optimization
IPM	Interior-point method
SDO	Semidefinite optimization
NCP	Non-linear complementarity problem
SOCO	Second-order cone optimization
KKT	Karush–Kuhn–Tucke
IPC	Interior point condition

Appendix A. Raw Tables—CPU Data

Table A1. Iteration number and CPU times.

Name	Iteration			CPU Time (s)		
	Classical	Algo 1	Algo 2	Classical	Algo 1	Algo 2
adlittle	54	35	33	0.09	0.07	0.06
afiro	16	17	18	0.46	0.49	0.24
agg	91	66	66	6.20	4.64	4.63
agg2	95	60	58	7.99	5.24	5.03
agg3	124	60	66	10.51	5.31	5.85
bandm	105	69	71	3.97	2.67	2.74
beaconfd	105	48	46	1.39	0.76	0.64
blend	50	26	24	0.07	0.05	0.04
bnl2	203	183	153	1089.95	1000.31	827.93
capri	85	74	55	3.11	2.84	2.06
chemcom	58	45	45	3.69	2.97	2.94
czprob	186	137	146	434.42	313.03	342.57
e226	130	72	120	3.76	2.20	3.66
ffff800	118	121	112	22.88	24.03	22.13
fit1p	80	38	42	43.91	21.15	23.39
forest6	43	22	22	0.07	0.05	0.04
ganges	50	43	42	49.60	42.40	42.73

Table A1. Cont.

Name	Iteration			CPU Time (s)		
	Classical	Algo 1	Algo 2	Classical	Algo 1	Algo 2
gfrd_pnc	118	74	74	28.88	18.68	18.56
israel	150	100	102	1.31	0.98	0.97
lotfi	74	61	56	1.28	1.16	1.04
maros_r7	75	6	65	47.15	41.12	41.72
pilot87	329	186	190	842.15	575.14	589.04
pilot	168	135	133	472.34	371.98	342.67
pilot_we	357	325	371	612.35	558.41	632.02
sc50a	28	20	22	0.03	0.03	0.02
sc50b	29	20	22	0.03	0.03	0.02
sc105	24	24	23	0.06	0.16	0.13
sc205	42	24	24	0.308	0.30	0.27
scagr25	43	42	42	3.75	3.62	3.55
scfxm1	83	53	54	4.45	4.33	3.17
scfxm2	80	68	66	20.06	17.84	17.34
scfxm3	86	77	77	49.00	44.49	44.68
scrs8	149	108	106	85.76	63.90	61.95
scagr7	30	22	23	0.11	0.21	0.11
scsd1	26	27	28	1.94	2.08	2.12
scsd6	24	25	27	5.91	6.24	6.77
scsd8	40	25	24	42.27	25.68	24.58
sctap1	71	56	63	3.79	3.16	3.50
sctap2	82	56	58	99.29	68.78	71.00
sctap3	99	60	62	232.41	144.41	149.83
share1b	127	86	89	0.85	0.71	0.71
share2b	41	31	30	1.65	1.06	1.21
stair	102	97	97	4.82	4.81	4.71
standata	105	83	82	28.15	22.48	23.12
standmps	211	129	142	58.05	36.08	39.97
stocfor1	91	51	52	0.27	0.32	0.25
stocfor2	230	176	191	848.32	651.89	668.12
truss	70	59	58	2917.26	2517.39	2560.80
lp-vtp-base	64	42	39	0.89	0.59	0.57
woodw	50	43	46	239.01	198.67	201.92

Appendix B. Relative Gap and the Norm—CPU Data

Table A2. Relative gap and the norm.

Name	Regap	$\ Ax - b\ $	$\ c - s - A^T y\ $
adlittle	8.69×10^{-7}	5.62×10^{-14}	1.29×10^{-10}
adlittle	8.82×10^{-7}	3.32×10^{-12}	1.29×10^{-10}
adlittle	1.27×10^{-7}	1.31×10^{-13}	1.83×10^{-5}
afiro	8.37×10^{-7}	4.70×10^{-17}	1.61×10^{-4}
afiro	7.18×10^{-7}	2.13×10^{-9}	1.61×10^{-4}
afiro	9.54×10^{-7}	1.64×10^{-9}	3.69×10^{-4}
agg	1.41×10^{-7}	2.38×10^{-12}	1.78×10^{-5}
agg	7.57×10^{-7}	2.02×10^{-11}	1.78×10^{-5}
agg	7.57×10^{-7}	2.02×10^{-11}	1.78×10^{-5}

Table A2. Cont.

Name	Regap	$\ Ax - b\ $	$\ c - s - A^T y\ $
agg2	3.23×10^{-8}	9.15×10^{-12}	3.34×10^{-6}
agg2	7.21×10^{-7}	5.27×10^{-13}	3.34×10^{-6}
agg2	2.03×10^{-9}	3.29×10^{-14}	3.37×10^{-6}
agg3	5.27×10^{-7}	1.13×10^{-10}	2.82×10^{-6}
agg3	5.77×10^{-7}	4.40×10^{-11}	2.82×10^{-6}
agg3	2.62×10^{-7}	1.01×10^{-11}	3.24×10^{-6}
bandm	8.64×10^{-7}	7.02×10^{-13}	1.76×10^{-3}
bandm	1.50×10^{-7}	4.43×10^{-15}	1.76×10^{-7}
bandm	7.66×10^{-7}	2.86×10^{-13}	8.31×10^{-9}
beaconfd	6.09×10^{-9}	7.13×10^{-5}	3.75×10^{-8}
beaconfd	4.26×10^{-8}	1.25×10^{-8}	3.75×10^{-8}
beaconfd	2.405×10^{-8}	6.97×10^{-9}	1.52×10^{-8}
blend	5.83×10^{-7}	2.87×10^{-16}	2.78×10^{-6}
blend	5.25×10^{-7}	6.43×10^{-10}	2.78×10^{-6}
blend	9.32×10^{-7}	7.54×10^{-14}	7.10×10^{-5}
bnl2	9.79×10^{-7}	1.65×10^{-16}	3.37×10^{-3}
bnl2	8.55×10^{-7}	2.27×10^{-16}	3.37×10^{-3}
bnl2	6.19×10^{-7}	1.97×10^{-16}	2.66×10^{-2}
capri	2.34×10^{-9}	2.57×10^{-16}	2.41×10^{-7}
capri	1.57×10^{-7}	1.45×10^{-10}	2.41×10^{-7}
capri	4.77×10^{-7}	6.37×10^{-10}	7.62×10^{-6}
chemcom	3.88×10^{-9}	1.48×10^{-16}	6.08×10^{-9}
chemcom	1.52×10^{-7}	1.66×10^{-12}	6.08×10^{-9}
chemcom	1.98×10^{-7}	1.72×10^{-12}	6.11×10^{-8}
czprob	9.42×10^{-7}	1.01×10^{-12}	1.59×10^{-3}
czprob	8.18×10^{-7}	8.94×10^{-13}	1.59×10^{-3}
czprob	7.07×10^{-9}	2.98×10^{-13}	3.81×10^{-5}
e226	2.03×10^{-7}	3.85×10^{-15}	5.40×10^{-4}
e226	2.95×10^{-7}	2.60×10^{-14}	5.40×10^{-4}
e226	7.65×10^{-7}	3.31×10^{-16}	4.22×10^{-3}
ffff800	9.51×10^{-7}	1.16×10^{-10}	3.45×10^{-7}
ffff800	3.96×10^{-7}	8.72×10^{-11}	3.45×10^{-7}
ffff800	1.31×10^{-8}	1.65×10^{-10}	6.25×10^{-7}
fit1p	8.80×10^{-7}	1.48×10^{-16}	1.02×10^{-5}
fit1p	1.25×10^{-7}	1.39×10^{-11}	1.02×10^{-5}
fit1p	1.01×10^{-7}	1.26×10^{-11}	7.49×10^{-10}
forest6	6.54×10^{-8}	6.95×10^{-16}	1.71×10^{-6}
forest6	8.14×10^{-7}	5.54×10^{-12}	1.71×10^{-6}
forest6	8.59×10^{-7}	5.57×10^{-12}	3.51×10^{-5}
ganges	7.87×10^{-7}	1.48×10^{-16}	2.40×10^{-7}
ganges	5.00×10^{-7}	6.77×10^{-12}	2.40×10^{-7}
ganges	4.25×10^{-7}	4.97×10^{-12}	3.41×10^{-8}
gfrd_pnc	1.88×10^{-8}	9.77×10^{-17}	1.24×10^{-6}
gfrd_pnc	2.99×10^{-7}	1.69×10^{-16}	1.24×10^{-6}
gfrd_pnc	2.11×10^{-7}	1.57×10^{-16}	3.98×10^{-6}
israel	8.63×10^{-7}	3.42×10^{-16}	4.00×10^{-6}
israel	2.37×10^{-7}	7.63×10^{-14}	4.00×10^{-6}
israel	2.33×10^{-7}	7.32×10^{-14}	3.99×10^{-6}

Table A2. *Cont.*

Name	Regap	$\ Ax - b\ $	$\ c - s - A^T y\ $
lotfi	1.07×10^{-8}	2.34×10^{-14}	1.89×10^{-8}
lotfi	4.72×10^{-7}	4.05×10^{-12}	1.89×10^{-8}
lotfi	2.23×10^{-7}	1.84×10^{-12}	5.32×10^{-7}
maros_r7	2.36×10^{-10}	2.94×10^{-16}	7.53×10^{-9}
maros_r7	6.67×10^{-8}	2.92×10^{-12}	7.53×10^{-9}
maros_r7	5.15×10^{-8}	2.059×10^{-12}	1.82×10^{-9}
pilot87	2.10×10^{-5}	7.50×10^{-12}	3.58×10^{-7}
pilot87	3.86×10^{-5}	2.46×10^{-12}	3.58×10^{-7}
pilot87	1.35×10^{-5}	2.76×10^{-12}	2.85×10^{-7}
pilot	2.26×10^{-7}	7.30×10^{-13}	5.62×10^{-7}
pilot	1.62×10^{-6}	1.00×10^{-13}	5.62×10^{-7}
pilot	1.53×10^{-6}	7.56×10^{-14}	1.55×10^{-6}
pilot_we	9.70×10^{-7}	6.77×10^{-15}	9.06×10^{-4}
pilot_we	2.87×10^{-7}	4.01×10^{-15}	9.06×10^{-4}
pilot_we	8.19×10^{-7}	6.18×10^{-15}	1.27×10^{-3}
sc50a	9.73×10^{-7}	7.74×10^{-17}	7.90×10^{-3}
sc50a	6.08×10^{-7}	1.39×10^{-12}	7.901×10^{-3}
sc50a	7.08×10^{-7}	7.57×10^{-16}	3.92×10^{-2}
sc50b	9.94×10^{-7}	1.36×10^{-16}	5.45×10^{-3}
sc50b	6.46×10^{-7}	3.58×10^{-14}	5.45×10^{-3}
sc50b	4.58×10^{-7}	1.90×10^{-16}	2.71×10^{-3}
sc105	1.02×10^{-7}	1.02×10^{-16}	1.20×10^{-2}
sc105	4.64×10^{-7}	8.09×10^{-13}	1.20×10^{-2}
sc105	6.63×10^{-7}	2.12×10^{-16}	9.76×10^{-3}
sc205	5.80×10^{-9}	1.12×10^{-16}	3.80×10^{-6}
sc205	3.35×10^{-7}	9.02×10^{-13}	3.80×10^{-6}
sc205	5.83×10^{-7}	1.69×10^{-16}	9.60×10^{-3}
scagr25	6.25×10^{-7}	1.32×10^{-16}	2.79×10^{-6}
scagr25	2.11×10^{-7}	5.24×10^{-12}	2.79×10^{-6}
scagr25	2.80×10^{-7}	8.46×10^{-11}	6.52×10^{-5}
scfxm1	9.01×10^{-7}	1.82×10^{-10}	4.43×10^{-6}
scfxm1	1.74×10^{-7}	9.71×10^{-10}	4.43×10^{-6}
scfxm1	9.94×10^{-7}	1.45×10^{-9}	7.66×10^{-6}
scfxm2	7.87×10^{-7}	5.19×10^{-10}	1.62×10^{-6}
scfxm2	4.25×10^{-7}	1.41×10^{-10}	1.62×10^{-6}
scfxm2	9.46×10^{-7}	1.25×10^{-10}	3.06×10^{-6}
scfxm3	9.61×10^{-7}	8.06×10^{-9}	7.08×10^{-7}
scfxm3	1.57×10^{-7}	3.51×10^{-10}	7.08×10^{-7}
scfxm3	1.60×10^{-7}	3.86×10^{-10}	6.77×10^{-7}
scrs8	8.13×10^{-7}	5.25×10^{-13}	4.79×10^{-8}
scrs8	7.91×10^{-7}	1.84×10^{-13}	4.79×10^{-8}
scrs8	2.03×10^{-7}	1.82×10^{-14}	2.15×10^{-8}
scagr7	9.96×10^{-7}	1.33×10^{-16}	1.01×10^{-6}
scagr7	7.30×10^{-7}	1.71×10^{-16}	1.01×10^{-6}
scagr7	3.32×10^{-7}	4.26×10^{-14}	6.08×10^{-6}
scsd1	6.18×10^{-7}	1.18×10^{-16}	1.18×10^{-7}
scsd1	6.90×10^{-7}	1.04×10^{-16}	1.18×10^{-7}
scsd1	7.51×10^{-7}	8.19×10^{-17}	1.08×10^{-5}

Table A2. *Cont.*

Name	Regap	$\ Ax - b\ $	$\ c - s - A^T y\ $
scsd6	7.89×10^{-7}	1.56×10^{-16}	3.14×10^{-5}
scsd6	8.16×10^{-7}	2.03×10^{-16}	3.14×10^{-5}
scsd6	6.87×10^{-7}	1.83×10^{-16}	3.64×10^{-5}
scsd8	2.94×10^{-8}	2.52×10^{-16}	2.64×10^{-5}
scsd8	3.12×10^{-7}	2.76×10^{-16}	2.64×10^{-5}
scsd8	5.75×10^{-7}	2.41×10^{-16}	5.35×10^{-6}
sctap1	5.77×10^{-9}	2.84×10^{-16}	4.66×10^{-6}
sctap1	4.76×10^{-7}	1.28×10^{-9}	4.66×10^{-6}
sctap1	2.18×10^{-9}	4.64×10^{-12}	3.66×10^{-8}
sctap2	6.17×10^{-9}	5.70×10^{-11}	1.33×10^{-7}
sctap2	1.49×10^{-7}	4.38×10^{-12}	1.33×10^{-7}
sctap2	1.79×10^{-7}	1.94×10^{-11}	2.16×10^{-6}
sctap3	1.96×10^{-8}	8.318×10^{-11}	2.61×10^{-6}
sctap3	5.08×10^{-7}	2.86×10^{-10}	2.61×10^{-6}
sctap3	2.63×10^{-7}	1.41×10^{-10}	2.41×10^{-6}
share1b	9.68×10^{-7}	1.27×10^{-16}	1.40×10^{-4}
share1b	9.29×10^{-8}	3.31×10^{-15}	1.40×10^{-4}
share1b	8.53×10^{-8}	2.59×10^{-15}	1.38×10^{-5}
share2b	9.60×10^{-7}	6.11×10^{-15}	4.34×10^{-6}
share2b	8.65×10^{-7}	1.23×10^{-9}	4.34×10^{-6}
share2b	1.14×10^{-7}	3.10×10^{-10}	1.41×10^{-5}
stair	9.30×10^{-7}	1.90×10^{-11}	1.48×10^{-5}
stair	6.96×10^{-7}	2.04×10^{-10}	1.48×10^{-5}
stair	2.84×10^{-7}	1.22×10^{-10}	2.15×10^{-5}
standata	7.12×10^{-7}	2.66×10^{-12}	3.02×10^{-12}
standata	3.87×10^{-7}	1.22×10^{-13}	3.02×10^{-12}
standata	9.88×10^{-7}	2.35×10^{-15}	4.45×10^{-11}
standmps	9.68×10^{-7}	8.69×10^{-9}	5.74×10^{-7}
standmps	1.75×10^{-7}	1.10×10^{-9}	5.74×10^{-7}
standmps	3.92×10^{-8}	6.92×10^{-11}	4.53×10^{-7}
stocfor1	1.20×10^{-8}	2.06×10^{-16}	5.68×10^{-6}
stocfor1	1.30×10^{-7}	2.69×10^{-14}	5.68×10^{-6}
stocfor1	1.96×10^{-7}	1.98×10^{-15}	1.28×10^{-5}
stocfor2	1.39×10^{-10}	2.15×10^{-16}	6.40×10^{-7}
stocfor2	5.61×10^{-10}	6.68×10^{-16}	6.40×10^{-7}
stocfor2	2.08×10^{-8}	6.16×10^{-15}	5.51×10^{-6}
truss	5.72×10^{-11}	1.40×10^{-15}	6.37×10^{-7}
truss	2.12×10^{-9}	6.87×10^{-16}	6.37×10^{-7}
truss	1.99×10^{-9}	6.67×10^{-16}	1.22×10^{-5}
lp-vtp-base	9.02×10^{-9}	2.12×10^{-9}	3.55×10^{-6}
lp-vtp-base	3.82×10^{-7}	4.49×10^{-9}	3.55×10^{-6}
lp-vtp-base	5.09×10^{-7}	2.42×10^{-8}	4.26×10^{-6}
woodw	3.85×10^{-5}	2.01×10^{-4}	4.05×10^{-4}
woodw	4.29×10^{-5}	1.51×10^{-5}	4.05×10^{-4}
woodw	1.33×10^{-4}	7.42×10^{-6}	2.09×10^{-4}

Appendix C. Data Tables for GPU

Table A3. Results for GPU.

Name	Iteration	Iteration	GPU Time (s)	GPU Time (s)
	Classical	Algorithm 1	Classical	Algorithm 1
adlittle	49	31	0.151	0.099
afiro	19	17	0.028	0.027
agg	74	60	1.635	1.339
agg2	84	47	2.108	1.181
agg3	81	54	2.108	1.378
bandm	700	105	10.05	1.529
beaconfd	700	50	5.318	0.392
blend	53	26	0.148	0.073
bnl2	222	236	182.065	192.634
capri	67	51	0.963	0.733
chemcom	76	47	1.719	1.075
d2q06c	306	166	467.293	252.344
fit1p	41	35	3.137	2.670
forest6	42	22	0.125	0.071
ganges	700	41	71.770	4.190
israel	157	128	1.311	1.109
lotfi	68	54	0.642	0.522
pilot87	246	181	462.709	339.440
pilot	128	92	105.27	75.457
pilot_we	399	269	84.727	56.946
sc50a	46	20	0.089	0.042
sc50b	28	20	0.052	0.039
sc105	21	21	0.085	0.082
sc205	45	23	0.397	0.207
scagr25	47	37	1.133	0.890
scfxm1	75	51	1.399	0.960
scfxm2	72	53	3.480	2.551
scfxm3	76	56	7.496	5.498
scagr7	32	26	0.141	0.117
scsd1	26	27	0.526	0.551
scsd6	24	25	1.071	1.110
scsd8	37	22	6.320	3.732
sctap1	60	52	1.229	1.073
3sctap2	90	47	16.300	8.485
sctap3	105	61	36.785	21.257
share1b	700	69	4.096	0.417
share2b	38	27	0.144	0.104
stair	700	82	13.832	1.640
standmps	183	700	8.892	34.161
stocfor1	67	38	0.255	0.151
lp-vtp-base	558	37	0.521	0.354

References

- Zhu, J.; Rosset, S.; Tibshirani, R.; Hastie, T. 1-norm support vector machines. *Adv. Neural Inf. Process. Syst.* **2003**, *16*, 49–56.
- Yang, J.; Zhang, Y. Alternating direction algorithms for ℓ_1 -problems in compressive sensing. *SIAM J. Sci. Comput.* **2011**, *33*, 250–278. [[CrossRef](#)]
- Yuan, M. High dimensional inverse covariance matrix estimation via linear programming. *J. Mach. Learn. Res.* **2010**, *11*, 2261–2286.
- Recht, B.; Re, C.; Tropp, J.; Bittorf, V. Factoring nonnegative matrices with linear programs. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1214–1222.
- Meshi, O.; Globerson, A. An alternating direction method for dual MAP LP relaxation. In Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, 5–9 September 2011; Proceedings, Part II; Springer: Berlin/Heidelberg, Germany, 2011; Volume 22, pp. 470–483.

6. Karmarkar, N. A new polynomial-time algorithm for linear programming. In Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, Washington, DC, USA, 30 April–2 May 1984; pp. 302–311.
7. Nesterov, Y.; Nemirovskii, A. *Interior-Point Polynomial Algorithms in Convex Programming*; SIAM: Philadelphia, PA, USA, 1994.
8. Roos, C.; Terlaky, T.; Vial, J.-P. *Theory and Algorithms for Linear Optimization: An Interior Point Approach*; Wiley: Chichester, UK, 1997.
9. Roos, C. A full-Newton step $O(n)$ infeasible interior-point algorithm for linear optimization. *SIAM J. Optim.* **2006**, *16*, 1110–1136. [[CrossRef](#)]
10. Terlaky, T.; Roos, C.; Peng, J. *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Algorithms*; Princeton University Press: Princeton, NJ, USA, 2002.
11. Li, M.M.; Zhang, M.W.; Huang, Z.W. A primal-dual interior-point method for linear optimization based on a new parameterized kernel function. *J. Nonlinear Funct. Anal.* **2019**, *38*, 1–20.
12. Fathi-Hafshejani, S.; Peyghami, R.M. An interior point algorithm for solving linear optimization problems using a new trigonometric kernel function. *Filomat* **2020**, *34*, 1471–1486. [[CrossRef](#)]
13. Kheirfam, B.; Haghighi, M. A wide neighborhood interior-point algorithm based on the trigonometric kernel function. *J. Appl. Math. Comput.* **2020**, *64*, 119–135. [[CrossRef](#)]
14. Touil, I.; Chikouche, W. Polynomial-time algorithm for linear programming based on a kernel function with hyperbolic-logarithmic barrier term. *Palest. J. Math.* **2022**, *11*, 127–135.
15. Touil, I.; Chikouche, W. Novel kernel function with a hyperbolic barrier term to primal-dual interior point algorithm for SDP problems. *Acta Math. Appl. Sin. Engl. Ser.* **2022**, *38*, 44–67. [[CrossRef](#)]
16. Dexter, G.; Chowdhury, A.; Avron, H.; Drineas, P. On the convergence of inexact predictor-corrector methods for linear programming. In Proceedings of the International Conference on Machine Learning, Baltimore, MD, USA, 17–23 July 2022; pp. 5007–5038.
17. Mehrotra, S. On the implementation of a primal-dual interior point method. *SIAM J. Optim.* **1992**, *2*, 575–601. [[CrossRef](#)]
18. Darvay, Z.; Illés, T.; Kheirfam, B.; Rigó, P.R. A corrector–predictor interior-point method with new search direction for linear optimization. *Cent. Eur. J. Oper. Res.* **2020**, *28*, 1123–1140. [[CrossRef](#)]
19. Yang, X.; Yin, H. A Mizuno-Todd-Ye Predictor-Corrector Infeasible-Interior-Point Method with the l_1 -Norm Wide Neighborhood for Linear Programming. *Appl. Math. Sci.* **2022**, *16*, 511–528. [[CrossRef](#)]
20. McDougall, T.J.; Wotherspoon, S.J. A simple modification of Newton’s method to achieve convergence of order $1 + \sqrt{2}$. *Appl. Math. Lett.* **2014**, *29*, 20–25. [[CrossRef](#)]
21. Argyros, I.K.; Deep, G.; Regmi, S. Extended Newton-like midpoint method for solving equations in Banach space. *Foundations* **2023**, *3*, 82–98. [[CrossRef](#)]
22. Monteiro, R.D.; Adler, I. Interior path following primal-dual algorithms. Part I: Linear programming. *Math. Program.* **1989**, *44*, 27–41. [[CrossRef](#)]
23. Sonnevend, G. An “analytical centre” for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In *System Modelling and Optimization: Proceedings of 12th IFIP Conference, Budapest, Hungary, 2–6 September 1985*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 866–875.
24. Cai, X.; Wang, G.; Zhang, Z. Complexity analysis and numerical implementation of primal-dual interior-point methods for convex quadratic optimization based on a finite barrier. *Numer. Algorithms* **2013**, *62*, 289–306. [[CrossRef](#)]
25. Wang, W.; Bi, H.; Liu, H. A full-newton step interior-point algorithm for linear optimization based on a finite barrier. *Oper. Res. Lett.* **2016**, *44*, 750–753. [[CrossRef](#)]
26. Gay, D.M. Electronic mail distribution of linear programming test problems. *Math. Program. Soc. Coal Newsl.* **1985**, *13*, 10–12.
27. Andersen, E.D.; Roos, C.; Terlaky, T. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Program.* **2003**, *95*, 249–277. [[CrossRef](#)]
28. Baldwin, S. Compute Canada: Advancing computational research. *J. Phys. Conf. Ser.* **2012**, *341*, 012001. [[CrossRef](#)]
29. Koch, T. The Final NETLIB-LP Results. *Oper. Res. Lett.* **2003**, *32*, 138–142. [[CrossRef](#)]
30. Świrydowicz, K.; Darve, E.; Jones, W.; Maack, J.; Regev, S.; Saunders, M.A.; Thomas, S.J.; Peleš, S. Linear solvers for power grid optimization problems: A review of GPU-accelerated linear solvers. *Parallel Comput.* **2022**, *111*, 102870. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.