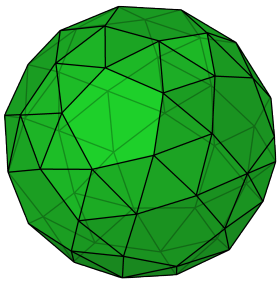
Projects in Mathematics and Applications

Interior Point Method

August 08, 2024

Pham Quoc Binh* [†]Bao Quy Dinh Tan
Nguyen Tran Ha Phuong [‡] [§]Le Minh Quy



September 10, 2024

*Thong Nhat B High School - Dong Nai Province, Vietnam
[†]Le Quy Don High School for the Gifted - Khanh Hoa Province, Vietnam
[‡]Wilbraham and Monson Academy, USA
[§]Gia Dinh High School - Ho Chi Minh City, Vietnam

Acknowledgments

First and foremost, we would like to extend our sincerest thanks to the founders of the summer camp: Mr. Can Tran Thanh Trung, Mr. Tran Hoang Bao Linh, and Mr. Le Viet Hai, along with the organizing committee of PiMA for establishing, maintaining, and developing the program for eight consecutive years, providing us the opportunity to experience such a beneficial and interesting summer camp on Mathematics and Applications.

We also wish to thank the instructors who taught us knowledge on Linear Algebra, Calculus, and Linear Programming in the first part of the camp. Your enthusiasm gave us the energy to push ourselves during the camp period.

Next, we would like to extend a special thank you to our mentors, Mr. Tran Phan Anh Danh, Mr. Nguyen Hoang Khang, and Mr. Vu Le The Anh, who closely monitored and helped our group throughout the completion of the project. Our topic would not have been completed without your assistance.

We are also grateful to the University of Science, Vietnam National University - Ho Chi Minh City for providing us with the classroom and equipment, allowing us to study in the best possible conditions. We are honored to have participated in the summer camp at one of the leading research universities in Vietnam.

Over the past summer days, we have had the opportunity to bond with campers and mentors from all over the country and from different parts of the world. We want to thank everyone for these meaningful and unforgettable experiences. We hope PiMA will continue to develop and inspire other high school students in the coming years.

Thank you.

August 08, 2024.
Group 6.

Summary

This report focuses on studying and applying the interior point method, specifically the primal-dual version, to solve linear programming problems. The interior point method differs from the traditional simplex method by moving inside the feasible region of the problem. This reduces the number of steps needed to find the optimal solution.

In the report, we present the theoretical foundations of the interior point method, implementation, and comparison with other algorithms to demonstrate its effectiveness. The final part of the report also discusses practical applications of the interior point method through the transportation problem and illustrates the efficiency of the algorithm with specific examples.

Contents

1 Introduction

1.1 Linear programming problem

Linear programming is an important tool for solving optimization problems. It is used to find the optimal value of a linear objective function, while satisfying the associated linear constraints.

The linear programming problem in standard form has the form

$$\begin{aligned} \text{Maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall\, i = 1, \ldots, m \\ & x_j \geq 0, \quad \forall\, j = 1, \ldots, n \end{aligned}$$

With

- $\sum_{j=1}^{n} c_j x_j$ is objective function
- $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ and $x_j \geq 0$ are constraints

1.2 Interior Point Method

The interior point method is an advanced method for solving linear programming problems. Unlike the simplex method - moving from vertex to vertex on the boundary of the feasible solution set of the problem - the interior point method moves inside this solution set. This helps to minimize the number of steps to find the optimal solution of the problem, thereby handling large and complex linear programming problems. That is also the advantage of the interior point method compared to other methods such as the simplex method or the elliptic method.

The basic approach of the interior point method is to move in the solution domain of the problem and gradually approach the optimal solution. While moving, it avoids the boundary of the solution domain created by the constraints of the original problem, thereby avoiding problems related to the boundary of the problem, which the simplex method often encounters. This is the highlight of the interior point method.

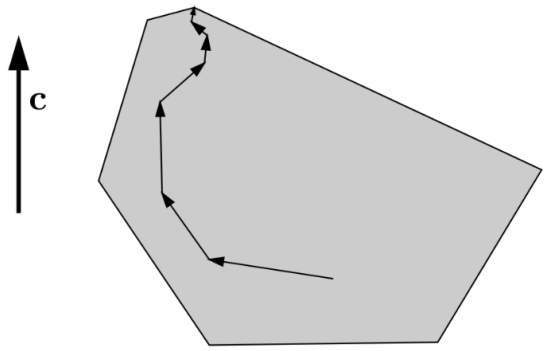


Figure 1: Illustration of the interior point method approach

The interior point method was first introduced in 1984 by Narendra Kramarkar, a researcher at IBM. Since then, many versions and improvements of this method have been developed to improve its effectiveness in many practical applications. In this paper, we focus on the centerline method, since it is the most computationally efficient method. At the same time, we focus on the primal-dual version, since it combines information from both the original and dual problems, which is important for solving large problems.

2 Algorithm design

2.1 Barrier problem

First, suppose we have a linear programming problem in standard form as follows:

$$\begin{aligned} &\text{Maximize } c^\intercal x \\ &\text{Subject to } Ax \leq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

and correspondingly we have the duality problem:

$$\begin{aligned} &\text{Minimize } b^\intercal y \\ &\text{Subject to } A^\intercal y \geq c \\ &\qquad\qquad y \geq 0 \end{aligned}$$

We will add the corresponding complementary variables w and z to the two problems above to bring them into canonical form, with the original problem becoming:

$$\begin{aligned} &\text{Maximize } c^\intercal x \\ &\text{Subject to } Ax + w = b \\ &\qquad\qquad x, w \geq 0 \end{aligned} \tag{1}$$

Similarly, the duality problem is then:

$$\begin{aligned} &\text{Minimize } b^\intercal x \\ &\text{Subject to } A^\intercal y - z = c \\ &\qquad\qquad y, z \geq 0 \end{aligned}$$

We wish to be able to remove the constraint on the inequality sign of the variables in(1), and we can do so by considering a problem with a different objective function, such as:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \sum_{i=1}^{n} \mathrm{I}(x_i) + \sum_{i=1}^{m} \mathrm{I}(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{2}$$

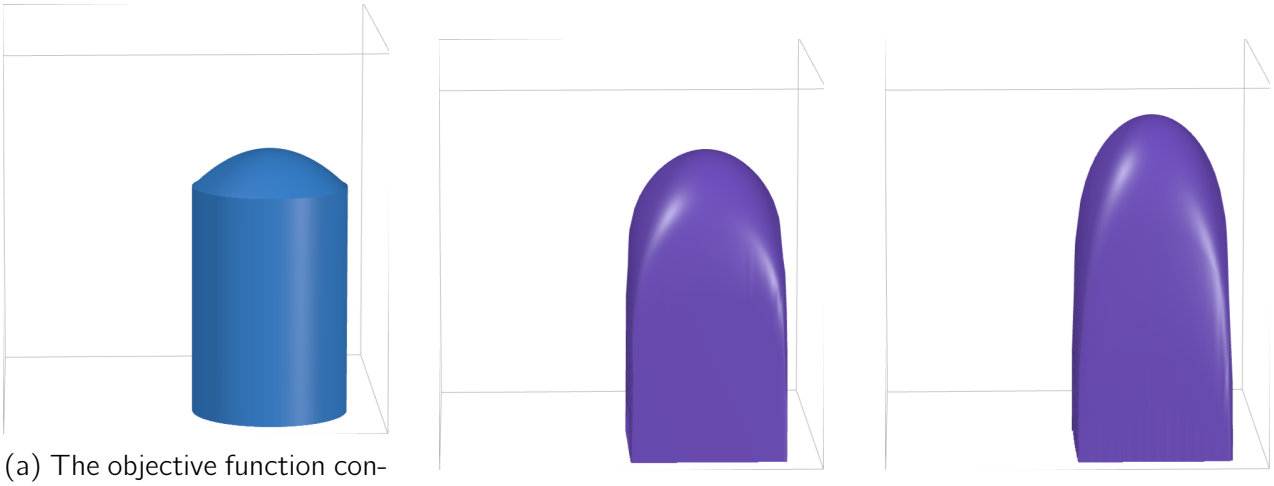
In which the function $I(x)$ is defined as follows:

$$I(x) = \begin{cases} -\infty & , x < 0 \\ 0 & , x \geq 0 \end{cases}$$

Consider the problem (2), we see that if x, $w \geq 0$ then this problem will become the problem (1), on the contrary, if there exists a component in the vectors x, vector w smaller than 0, the objective function immediately approaches $-\infty$ contrary to the fact that we are maximizing the objective function. Therefore, the problem (2) contains the problem (1), but the constraints on the signs of x and w have been removed.
But with such a function, it is very difficult for us to study the problem, because the objective function is not continuous so there is no derivative on $\mathbb{R}$, so we cannot directly apply analytical methods to study it.
However, let's consider another option as follows:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{i=1}^{m} \ln(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{3}$$

where $\mu > 0$ is a real parameter.
Our new objective function has a logarithm function which is a function with multiple derivatives, which makes it easy to apply calculus directly to the study. On the other hand, the objective function as above includes the condition that the variables $x, w > 0$ and prevents x and w from being zero points, because assuming that x_i approaches 0, the objective function will gradually approach $-\infty$ which is contrary to maximizing the objective function. So for our new objective function, the maximum point of each problem corresponding to each μ will always lie inside and not above or beyond the acceptable solution boundary. See (Figure: 2) to better understand objective functions containing $I(x)$ and logarithms.



(a) The objective function containing I(x)

(b) $\mu = 0.1$

(c) $\mu = 0.2$

Figure 2: In **picture a** is an illustration of the objective function containing the function $I(x)$, and **figure b**, **figure c** are illustrations of the objective function with the logarithm function corresponding to $\mu = 0.1$ and $\mu = 0.2$. In all three figures, the acceptable solution region is represented by a circle, the shaded part represents the value of the objective function, the value of the objective function becomes smaller at lower points. We can see that around the acceptable solution region for the objective function containing $I(x)$ is a barrier, which if you just step over, you will fall to extremely small values. As for **figure b** and **figure c**, when approaching the boundary of the acceptable solution region, there will be a gentle slope down to deeper value domains but not falling as suddenly as the objective function containing $I(x)$. On the other hand, with μ having a smaller value, we can see that the peak of the logarithmic objective function becomes closer to the peak of the objective function containing $I(x)$.

The problem (3) above is called the corresponding "barrier problem" of problem (3). And the problem (3) above is not equivalent to problem (3). Corresponding to each μ we will get a separate problem. Although not equivalent to the numerical problem (1), if we choose μ small enough, from solving the numerical problem (3) we can approximate the optimal solution of the problem (1)

In the specific numerical problem (3), we used the logarithm function to represent the objective function, but there will be many other ways to choose the function to create an objective function representation with the same purpose of limiting the problem to always be within the acceptable solution domain. Such functions are called **barrier functions**. The function used in the numerical problem (3) is called "**logarithm barrier function**".

We have a way to visualize the barrier function as follows, assuming the acceptable solution set of the problem is a polygon. The value of the objective function will approach $-\infty$ as it approaches the edge of the polygon. Corresponding to each different μ we will find the maximum value of the objective function achieved at a certain point inside the polygon (see figure 3)

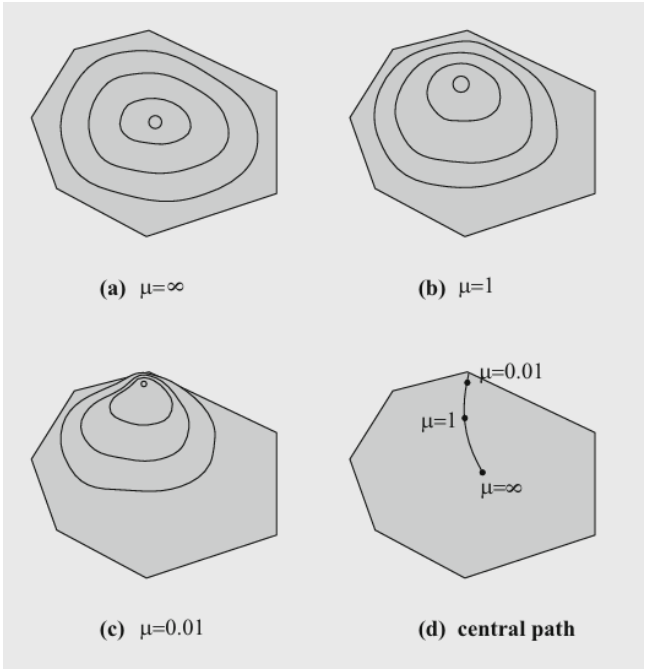


Figure 3: Figures (a), (b), (c) illustrate the values of the objective function on the set of acceptable solutions to the problem. The maximum values will be achieved at some point inside the polygon. Figure (d) retraces the paths of the extreme points when μ changes from ∞ to 0

This also shows why the method we are studying is called *Interior Point Method*, because for each μ above, we will determine the largest point that only lies in the acceptable solution set. But as μ gradually approaches 0, we will approximate the optimal value of the initial objective function. And if we connect the points where the objective function is largest in problems with different μ, we will get a path, which is called **central path**.

2.2 Center Path

The basic idea of the center method is that we start from the center of the solution domain (or an acceptable solution to the problem) and determine the optimal solution of each deterministic problem. After solving the optimal solution of n deterministic problems - each deterministic problem corresponding to each value of μ, we arrive at the optimal solution of the original problem. The set of optimal solutions of each deterministic problem corresponding to each value of μ is the center path.

The problem we need to solve is:

$$\begin{aligned} &\text{Maximize } c^{\intercal}x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{i=1}^{m} \ln(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{4}$$

We need to maximize the objective function, with the conditions $Ax + w = b$ and $x, w \geq 0$. Then, our problem to be solved is transformed into a blocking problem of the form

$$\max_{x} \left(c^{T}x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\ x, w > 0 \right)$$

Besides, as mentioned, the center path is a set of points. Therefore, we will use the arg function to return the x values we need to find, with x being the optimal solution of each blocking problem.

$$\arg\max_x \left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\ x, w > 0 \right)$$

The mathematical representation of the center path is

$$x^*(\mu) = \arg\max_x \left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\ x, w > 0 \right)$$

With:

- $c^T x$: Objective function of primal problem.
- $Ax = b$: Constraints of the original problem.
- $\sum_{i=1}^{n} \ln(x_i)$ and $\sum_{j=1}^{n} \ln(w_j)$: Barrier function
- μ: The weight of the barrier function. Initially, μ is very large, all the barrier functions simultaneously prevent the optimal solution of the barrier problem from moving to the boundary, the optimal solution of the barrier problem at this time is located in the center of the solution domain. When μ approaches 0^+, the barrier function becomes extremely small, the barrier problem will gradually become equivalent to the original problem. From then on, the optimal solutions of the two problems will be almost equivalent.

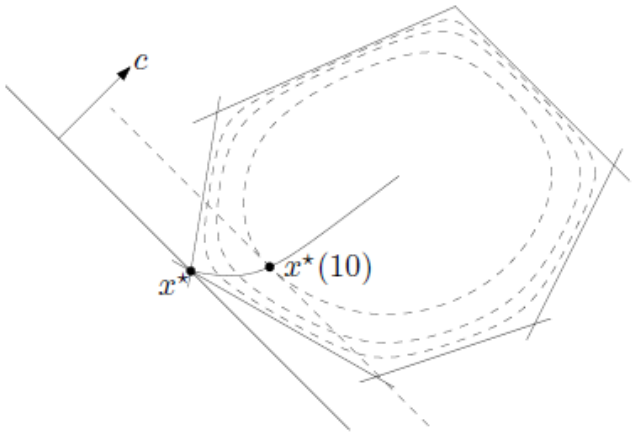


Figure 4: As μ approaches 0, the influence of the barrier function will decrease, leading to easier access to the optimal value. Then, the barrier problem (4) will gradually return to the problem (1)

For each μ, we find x and w so that

$$\left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax = b,\ x, w > 0 \right)$$

achieves the maximum value. To find the optimal solution of each blocking problem, based on each value of μ, we use Newton's method presented in Section 2.4.

2.3 Lagrangian function

For convenience, we will use the matrix convention x **in lowercase**:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

And the matrix X **in uppercase** will be:

$$X = \begin{pmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & x_n \end{pmatrix}$$

So the matrix X^{-1} will be:

$$X^{-1} = \begin{pmatrix} \frac{1}{x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \frac{1}{x_n} \end{pmatrix}$$

There are also e matrices of size $n \times 1$ which are of the form:

$$e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

This matrix will have a size depending on where it is written to validate matrix multiplication.

We can prove that for each μ, there is only one unique solution to the blocking problem. And as μ gets closer to 0, the solution to the blocking problem gets closer to the set of solutions to the original problem. Let's consider the blocking problem:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \mu \sum_j \ln x_j + \mu \sum_i \ln w_i \\ &\text{Subject to } Ax + w = b. \end{aligned} \tag{5}$$

This is an optimization problem with constraints that are equations, so we can use the Lagrange multiplier method to solve it. The Lagrangian function of the problem is:

$$L(x, w, y) = c^\intercal x + \mu \sum_j \ln x_j + \mu \sum_i \ln w_i + y^\intercal (b - Ax - w). \tag{6}$$

Since the stationary points of the Lagrangian function will also be local extrema of the original problem, we can derive a system of equations for the solution of the blocking problem. Consider the derivative with respect to each variable of the Lagrangian function:

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= c_j + \mu \frac{1}{x_j} - \sum_i y_i a_{ij} = 0, j = 1, 2, \ldots, n, \\ \frac{\partial L}{\partial w_i} &= \mu \frac{1}{w_i} - y_i = 0, i = 1, 2, \ldots, m, \\ \frac{\partial L}{\partial y_i} &= b_i - \sum_j a_{ij} x_j - w_i = 0, i = 1, 2, \ldots, m. \end{aligned} \tag{7}$$

Writing the above derivatives in matrix form, we have the system of equations:

$$\begin{aligned} A^T y - \mu X^{-1} e &= c \\ y &= \mu W^{-1} e \\ Ax + w &= b \end{aligned} \tag{8}$$

Let $z = \mu X^{-1} e$ where X is a square matrix in which all numbers are 0 and the variables x are on the main diagonal. Substituting z in, we will have the system of equations:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ z &= \mu X^{-1} e \\ y &= \mu W^{-1} e \end{aligned} \tag{9}$$

We multiply X into both sides of the $3rdequation$, W into both sides of the $4thequation$ to get the system of equations:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned} \tag{10}$$

Looking at the first 2 equations of the above system of equations, we see that equations 1 and 2 are the conditions of the original main problem and its corresponding dual problem, respectively. As μ gradually decreases to 0, equations 3 and 4 will gradually return to the form:

$$\begin{aligned} XZ &= 0 \\ YW &= 0 \end{aligned} \tag{11}$$

These two equations, paying attention to the complementary deviations of the main problem and the dual problem, are the conditions for the optimal solution of the problem. Therefore, the system of equations (10) has a solution that both optimizes the objective function and satisfies the requirements of the original problem.

2.4 Algorithm

2.4.1 Overall idea

1. Start with a valid (x, w, y, z) set.
2. At each iteration, go to a new point $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ on the center line to the optimal solution.
3. The algorithm has 2 phases: Find the starting point and follow the center line

2.4.2 Follow the central path

1. (Phase I) Set $\mu = 1$ and choose the set $(x, w, y, z) > 0$ that satisfies the system (10).
2. (Phase II) While $\mu \geq \epsilon$, repeat steps 3 and 4. If $\mu < \epsilon$, return x as an optimal solution and stop the loop
3. Replace μ with $\left(1 - \frac{1}{2\sqrt{n}}\right) \mu$ [1]
4. (Newton's Step) Find $(\Delta x, \Delta w, \Delta y, \Delta z)$ as a solution to the system (10):

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned} \tag{12}$$

5. Replace (x, w, y, z) with $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ and repeat the loop.

[1] This method of reducing μ is referred to in [3] section **7.2 Interior Point Methods**

2.4.3 Phase II

In Phase II, the Newton step is an important step for us to solve the problem. Therefore, to find the solution $(\Delta x, \Delta w, \Delta y, \Delta z)$ we will use **Newton's method**.

In numerical analysis, the Newton method is a method for finding the approximate value of a function with real parameters by linear approximation. We often use the Newton method to find the solution of the function or $F(\xi) = 0$.

We can generalize the Newton method to find the solution of the non-linear system of equations $F(\xi) = 0$ by updating ξ. We consider the function $F : \mathbb{R}^n \to \mathbb{R}^n$ with

$$F(\xi) = \begin{bmatrix} F_1(\xi) \\ F_2(\xi) \\ \vdots \\ F_N(\xi) \end{bmatrix}, \quad \xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix}$$

Finding the solution of the system of equations means we need to find ξ^* such that $F(\xi^*) = 0$. Using the linear approximation method $F(\xi)$ using the Taylor Expansion, we have:

$$F(\xi + \Delta\xi) \approx F(\xi) + F'(\xi)\Delta\xi,$$

where $F'(\xi)$ is the Jacobian matrix of $F(\xi)$:

$$J_F(X) = \nabla F = F'(\xi) = \begin{bmatrix} \frac{\partial F_1}{\partial \xi_1} & \frac{\partial F_1}{\partial \xi_2} & \cdots & \frac{\partial F_1}{\partial \xi_N} \\ \frac{\partial F_2}{\partial \xi_1} & \frac{\partial F_2}{\partial \xi_2} & \cdots & \frac{\partial F_2}{\partial \xi_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_N}{\partial \xi_1} & \frac{\partial F_N}{\partial \xi_2} & \cdots & \frac{\partial F_N}{\partial \xi_N} \end{bmatrix}$$

Solving the equation $F(\xi) + F'(\xi)\Delta\xi = 0$, we find $\Delta\xi$; replace ξ with $\xi + \Delta\xi$ and repeat the above process until the function $F(\xi)$ reaches approximately 0.

Returning to the central line problem, we call:

$$\xi = \begin{bmatrix} x \\ w \\ y \\ z \end{bmatrix} \qquad F(\xi) = \begin{bmatrix} Ax + w - b \\ A^T y - z - c \\ XZe - \mu e \\ YWe - \mu e \end{bmatrix}.$$

We find the Jacobi matrix $F'(\xi) = \begin{bmatrix} A & I & 0 & 0 \\ 0 & 0 & A^T & -I \\ Z & 0 & 0 & X \\ 0 & Y & W & 0 \end{bmatrix}$

Solve the system of equations $F'(\xi)\Delta\xi = -F(\xi)$:

$$\begin{aligned} A\Delta x + \Delta w &= b - Ax - w \\ A^T\Delta y - \Delta z &= z + c - A^T y \\ Z\Delta x + X\Delta z &= \mu e - XZe \\ Y\Delta w + W\Delta y &= \mu e - YWe \end{aligned} \tag{13}$$

ta tìm c $\Delta\xi$ vi $\Delta\xi = \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \\ \Delta z \end{bmatrix}$

2.4.4 Phase I

In Phase I, we need to find a set of $(x, w, y, z) > 0$ that satisfies the system (10) with $\mu = 1$. We will start with any set of $(x, w, y, z) > 0$, for example, a matrix of all numbers 1, then perform Newton's method to solve, turning the initial set we take into any satisfying set. The reason we can do this is because the Newton orientation of the system (10) is also oriented into the domain, so from a point outside the domain, we can gradually go to a point that satisfies the system (10) and lies in the solution domain. This has been proven by Lustig [2], specifically in sections 3 and 4.

2.5 Karush-Kuhn-Tucker (KKT) Conditional

As discussed in the previous section, the conditional system (10) was used to solve the linear programming problem, which is actually a consequence of a more general problem in convex optimization.
In convex optimization, the constraint functions and the objective function are convex and differentiable, which is more general than the linear programming problem where the conditional systems are just linear constraints (which are also convex constraints). A notable result in convex optimization is **Karush-Kuhn-Tucker (KKT) conditions**[2], this result has great significance because this is the sufficient condition for a feasible solution to be confirmed as the optimal solution of the problem, from which we can build an algorithm to find optimal solutions for convex optimization problems and therefore we will simultaneously solve linear programming problems.

2.5.1 Karush-Kuhn-Tucker (KKT) conditions in convex optimization problems

We consider the original problem as follows in convex optimization:

$$\begin{aligned} &\text{Minimize } f_0(x) \\ &\text{Subject to } f_i(x) \le 0, \quad i = 1, \dots, m \\ &\qquad\qquad h_i(x) = 0, \quad i = 1, \dots, p. \end{aligned}$$

The Lagrangian dual problem with the objective function being the Lagrangian dual function $g(\lambda, \nu)$ [3] The corresponding Lagrangian dual problem of the above problem in convex optimization is:

$$\begin{aligned} &\text{Maximize } g(\lambda, \nu) = \inf_{x \in D} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \\ &\text{Subject to } \lambda \ge 0 \end{aligned}$$

Assume that our original problem is a convex problem, i.e. the functions f_i, h_i are both convex and differentiable, then if we find a set $(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$ satisfying the following set of conditions:

$$\begin{aligned} f_i(\tilde{x}) &\le 0, \quad i = 1, \dots, m \\ h_i(\tilde{x}) &= 0, \quad i = 1, \dots, p \\ \tilde{\lambda}_i &\ge 0, \quad i = 1, \dots, m \\ \tilde{\lambda}_i f_i(\tilde{x}) &= 0, \quad i = 1, \dots, m \\ \nabla f_0(\tilde{x}) + \sum_{i=1}^{m} \tilde{\lambda}_i \nabla f_i(\tilde{x}) + \sum_{i=1}^{p} \tilde{\nu}_i \nabla h_i(\tilde{x}) &= 0 \end{aligned} \tag{14}$$

[2] See more about this condition in [1] section **5.5.3 KKT optimality conditions**
[3] See also [1] section **5.2 The Lagrangian dual problem**

then $\tilde{x}$ is the optimal solution of the original problem, and $(\tilde{\lambda}, \tilde{\nu})$ is the optimal solution of the dual problem.

The above set of conditions (14) is called the **Karush-Kuhn-Tucker (KKT) set of conditions**.

In this system of conditions, the first two lines are the conditions to satisfy $\tilde{x}$ as a feasible solution of the original problem, the third line is the condition for $\tilde{\lambda_i}$ to be a feasible solution of the dual problem, the fourth line is the complementary deviation of the optimal solution of the original problem and the dual problem is equal to 0, the fifth line is the condition that the gradient of the Lagrangian function $L(x, \lambda, \nu)$ is equal to 0.

Because we have assumed that the original problem we are considering is a convex problem, and the gradient of the Lagrangian function $L(\tilde{x}, \tilde{\lambda}, \tilde{\nu}) = 0$, we can assert that $\tilde{x}$ is the optimal solution of the original problem. Accordingly, we will have $(\tilde{\lambda}, \tilde{\nu})$ is the optimal solution of the dual problem. [4]

2.5.2 Karush-Kuhn-Tucker conditional system in linear programming problems

Returning to the conditional system (10) and the linear optimization problem (also a subclass of convex problems) we are considering:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned}$$

We see that if (x, w, y, z) is a solution to the above system of equations, then the condition is also satisfied: the gradient of the corresponding Lagrangian function of the original problem is 0 (corresponding to condition number 5 of the 14 system), and at the same time, conditions number 1 and number 2 are also satisfied (corresponding to conditions 1, 2, 3 of the 14 system).

Throughout the process of our algorithm mentioned above, we have gradually reduced μ to 0, without breaking the given conditions, then conditions 3 and 4 in the system (10) corresponding to condition 4 in the system (14) have been adjusted (instead of XZe = 0 and YWe = 0, it is XZe = μe and YWe = μe).

This leads to the fact that our set (x, w, y, z) will gradually satisfy the above KKT system and become the optimal solution of the original problem.

From the above results, it can be seen that the conditional system (10) is essentially the **Karush-Kuhn-Tucker conditional system** in the case where the convex functions of the problem are linear functions and the parameter μ approaches 0.

[4] For detailed proof, see [1] **Chapter 5 Duality**

3 Algorithm implementation and comparison

Here is the pseudocode of the algorithm:

1: Initialize the initial variables: $\mu \leftarrow 1$, $\varepsilon \leftarrow$ A value small enough to approximate the solution well.
2: From the results in phase 1 of the algorithm, we get the initial values of the set (x, y, z, w) that satisfy the system of equations: $\begin{cases} Ax + w = b \\ A^T y - z = c \\ XZe = \mu e \\ YWe = \mu e. \end{cases}$
3: **while** $\mu \geq \varepsilon$ **do**
4: $\quad \Delta x, \Delta y, \Delta z, \Delta w \leftarrow \text{NewtonMethod}(\mu, x, y, z, w)$
5: $\quad x \leftarrow x + \Delta x$
6: $\quad y \leftarrow y + \Delta y$
7: $\quad z \leftarrow z + \Delta z$
8: $\quad w \leftarrow w + \Delta w$
9: $\quad \mu \leftarrow \left(1 - \frac{1}{2\sqrt{n}}\right) \mu$
10: **end while**
11: The above iteration process will end after a finite number of steps, then return the result that approximates the optimal solution.
12: **return** x

Here is how to implement the algorithm in Python and plot the path our algorithm takes to reach the optimal point:

```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output

def convertDiag(x):
    X = np.zeros((len(x), len(x)))
    for i in range(len(x)):
        X[i][i] = x[i, 0]  # Extract the single element from the array
    return X

def Newton_Method(x, w, y, z, mu, c, A, b):
    X = convertDiag(x)
    Y = convertDiag(y)
    W = convertDiag(w)
    Z = convertDiag(z)

    AT = np.transpose(A)
    m, n = A.shape

    F_e = np.block([[np.dot(A, x) + w - b],
                    [np.dot(AT, y) - z - c],
                    [np.dot(X, z) - mu * np.ones((n, 1))],
                    [np.dot(Y, w) - mu * np.ones((m, 1))]])

    dF_e = np.block([
        [A, np.identity(m), np.zeros((m, m)), np.zeros((m, n))],
        [np.zeros((n, n)), np.zeros((n, m)), AT, -np.identity(n)],
        [Z, np.zeros((n, m)), np.zeros((n, m)), X],
        [np.zeros((m, n)), Y, W, np.zeros((m, n))]
    ])

    res = np.linalg.solve(dF_e, -1 * F_e)

    delta_x = res[:n]
    delta_w = res[n:n + m]
    delta_y = res[n + m:n + m + m]
    delta_z = res[n + m + m:]

    return delta_x, delta_w, delta_y, delta_z

epsilon = 1e-10
def phase_II(x, w, y, z, c, A, b):
    mu = 1

    AT = np.transpose(A)
    m, n = A.shape

    path = []

    path.append([float(x[0, 0]), float(x[1, 0])])  # Extract single elements

    while mu >= epsilon:
        mu = (1 - 1 / (2 * np.sqrt(n))) * mu

        delta_x, delta_w, delta_y, delta_z = Newton_Method(x, w, y, z, mu, c
    , A, b)

        x = x + delta_x
```

```
        y = y + delta_y
        w = w + delta_w
        z = z + delta_z

        path.append([float(x[0, 0]), float(x[1, 0])])  # Extract single elements


    return x, path

def phase_I(c, A, b):
    m, n = A.shape

    x = np.ones((n, 1))
    w = np.ones((m, 1))
    y = np.ones((m, 1))
    z = np.ones((n, 1))

    for i in range(100):
        delta_x, delta_w, delta_y, delta_z = Newton_Method(x, w, y, z, 1, c, A, b)

        x = x + delta_x
        y = y + delta_y
        w = w + delta_w
        z = z + delta_z

    return x, w, y, z

def solve(cT, A, bT):
    c = np.transpose(cT)
    b = np.transpose(bT)

    x_start, w_start, y_start, z_start = phase_I(c, A, b)

    x_optimal, path = phase_II(x_start, w_start, y_start, z_start, c, A, b)

    return np.round(np.dot(cT, x_optimal), 3), path

def visualize(cT, A, bT, cpath_points, lim_x1=10, lim_x2=10):
    x1 = np.linspace(0, lim_x1, 400)
    x2 = np.linspace(0, lim_x2, 400)
    X1, X2 = np.meshgrid(x1, x2)

    m, n = A.shape

    constraint = []
    for i in range(m):
        constraint.append(A[i][0]*X1 + A[i][1]*X2 - bT[i])

    constraint = np.array(constraint)

    mu = 0.01
    epsilon = 1e-6

    barrier_function = np.log(-np.minimum(constraint[0], -epsilon))
    for i in range(1, m):
        barrier_function = barrier_function + np.log(-np.minimum(constraint[i], -epsilon))
    barrier_function = mu * barrier_function
    barrier_function = barrier_function + mu * (np.log(x1 + epsilon) + np.
```

```
    log(x2 + epsilon))

    goal_function = cT[0]*X1 + cT[1]*X2 + barrier_function

    feasible_mask = (constraint[0] <= 0)
    for i in range(1, m):
        feasible_mask = feasible_mask & (constraint[i] <= 0)
    goal_function = np.where(feasible_mask, goal_function, np.nan)

    plt.figure(figsize=(10, 10))
    contour = plt.contourf(X1, X2, goal_function, cmap='viridis', levels
    =100, alpha=0.7, extend='both')

    cbar = plt.colorbar(contour)
    cbar.set_label('Objective function value')

    for i in range(m):
        plt.contour(X1, X2, constraint[i], levels=[0], colors='r',
    linestyles='--')

    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.title('Feasible solution range')

    arrow_size = 0.03
    for i in range(len(cpath_points) - 1):
        arrow_size = arrow_size * 0.5
        vector_start = (cpath_points[i][0], cpath_points[i][1])
        vector_end = (cpath_points[i + 1][0], cpath_points[i + 1][1])
        plt.arrow(vector_start[0], vector_start[1],
                  vector_end[0] - vector_start[0],
                  vector_end[1] - vector_start[1], width=arrow_size,
                  head_width=arrow_size * 3, head_length=arrow_size * 1.5,
                  fc='black', ec='black')

    plt.grid(True)
    plt.axhline(0, color='black', linewidth=0.5)
    plt.axvline(0, color='black', linewidth=0.5)

    plt.show()

def solveAndVisualize(cT, A, bT, lim_x1 = 10, lim_x2 = 10):
  ans, path = solve(cT, A, bT)

  cT = np.asarray(cT)[0]
  A = np.array(A)
  bT = np.asarray(bT)[0]

  visualize(cT, A, bT, path, lim_x1, lim_x2)
```

Test the code with the following problem:

$$
\begin{aligned}
\text{Maximize } & x_1 + x_2 \\
\text{Subject to } & x_1 \leq 4 \\
& x_2 \leq 1,7 \\
& 2x_1 + 3x_2 \leq 10 \\
& x_1 - 3x_2 \leq 3 \\
& -2x_1 + 6x_2 \leq 8 \\
& -3x_1 - 6x_2 \leq -10
\end{aligned}
\tag{15}
$$

Enter the corresponding parameters and run the code as follows:

```
cT = np.matrix([1, 1])
A = np.matrix([[1, 0],
               [0, 1],
              [2, 3],
              [1, -3],
              [-2, 6],
              [-3, -6]])
bT = np.matrix([4, 1.7, 10, 3, 8, -10])
solveAndVisualize(cT, A, bT, 5, 5)
```

After running the code, we will get the image:

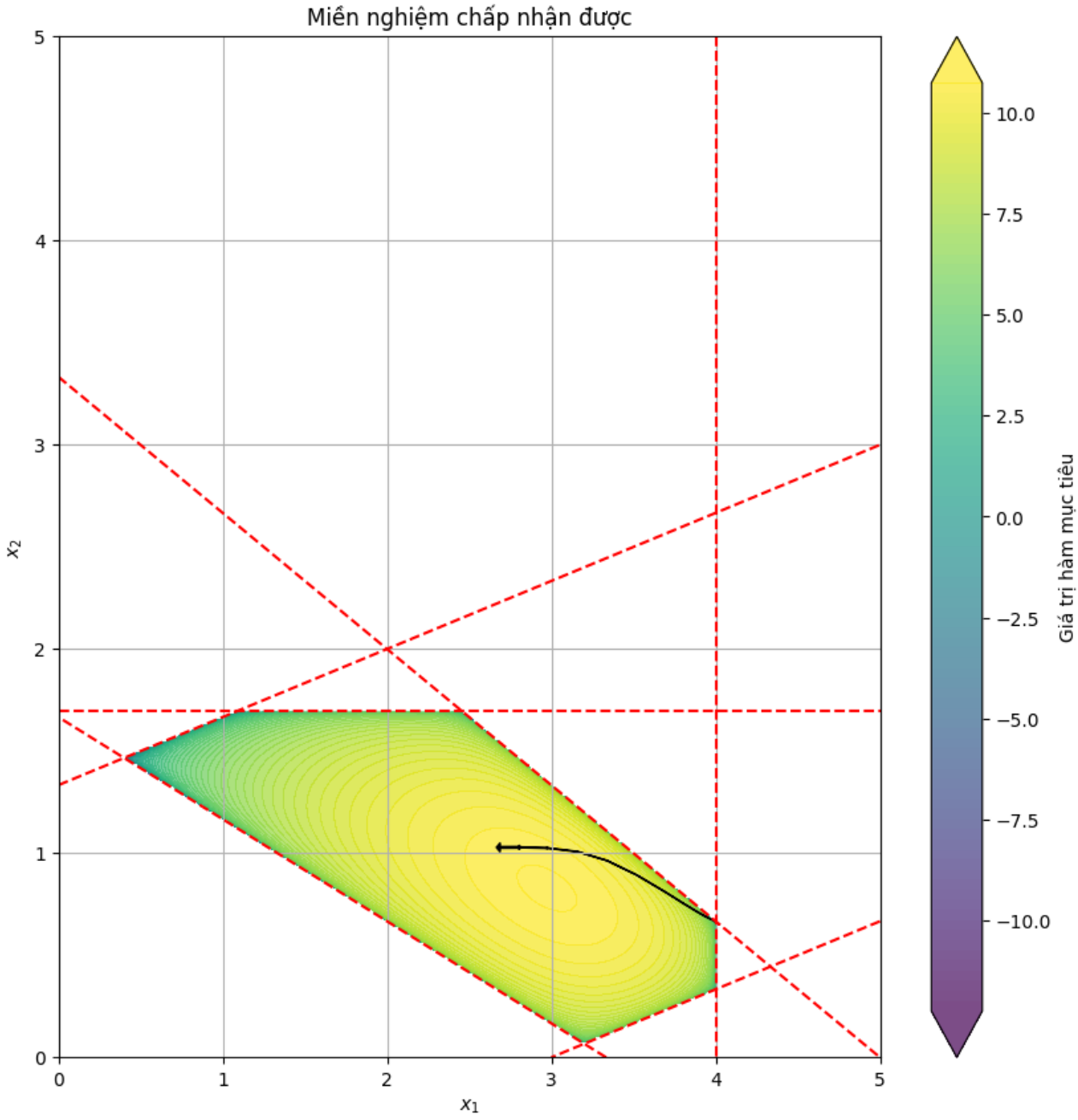


Figure 5: $\mu = 1$

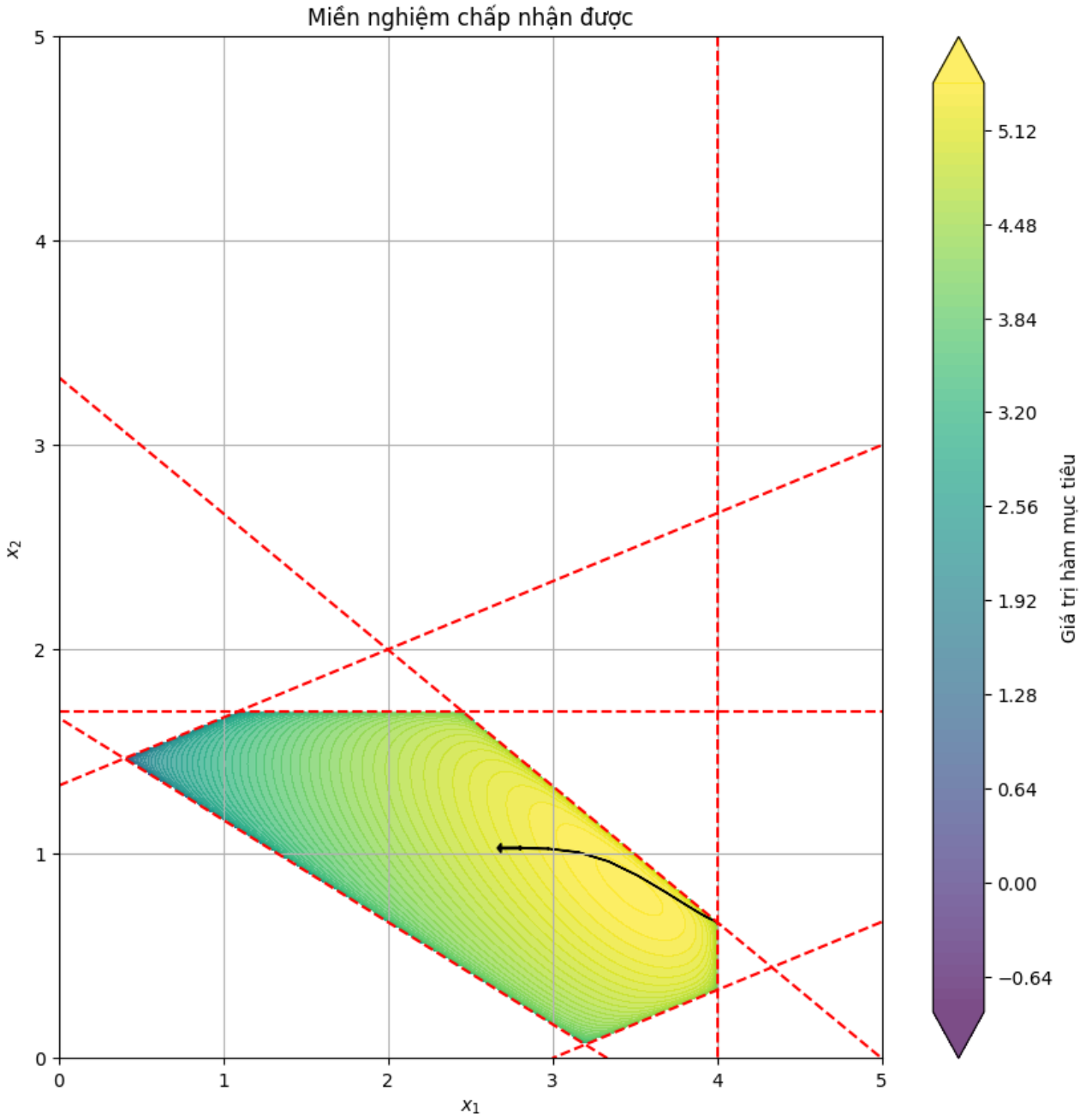


Figure 6: $\mu = 0.2$

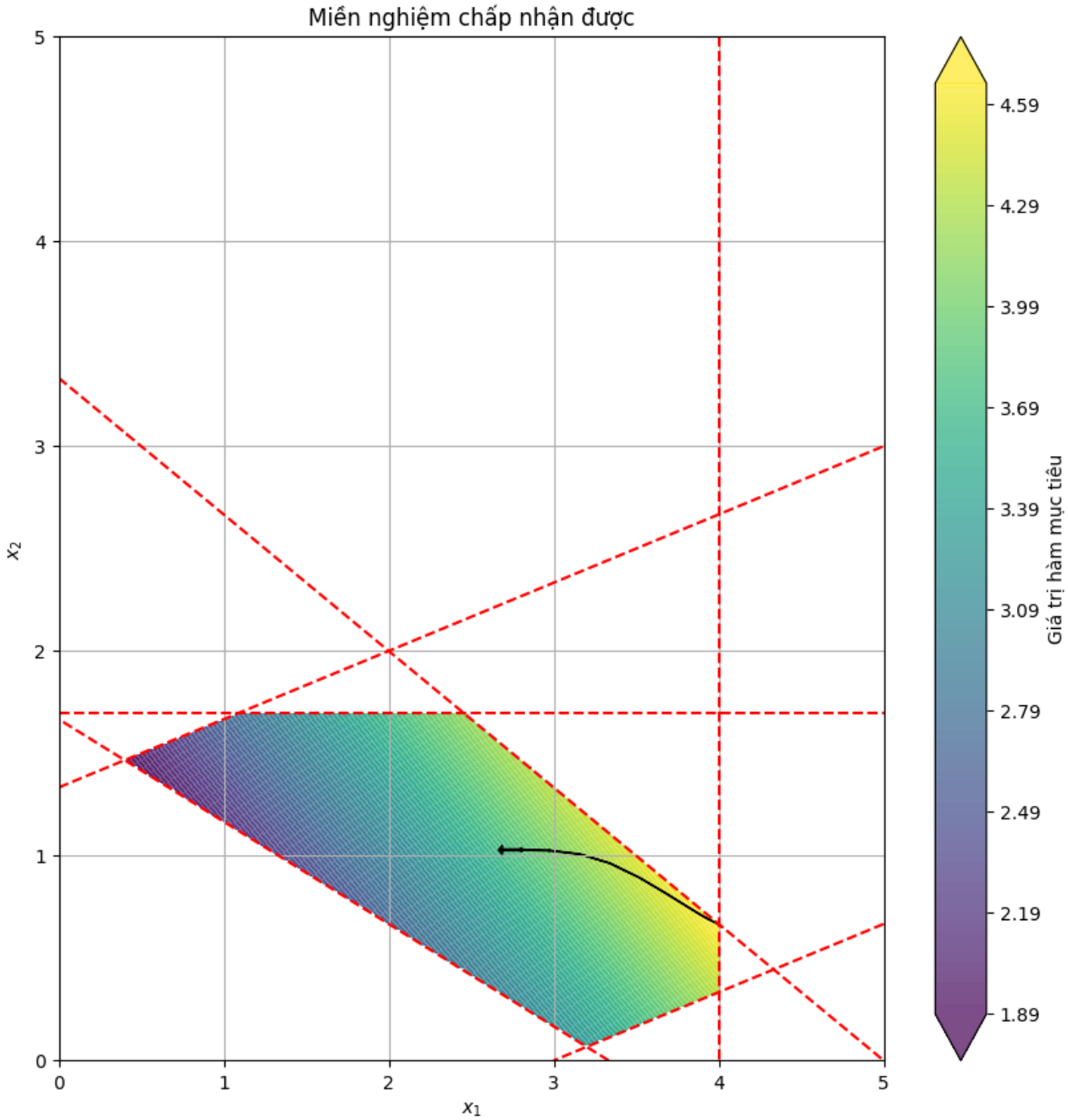


Figure 7: $\mu = 0.001$

In the figure, we draw a path connected from the points that our algorithm passes through and the valid solution region with the color filled as light if at that point the objective function value of the barrier problem is large, and the color is filled in. is bold if at that point the objective function value of the blocking problem is small. We see that as μ gradually decreases (from 1 to near 0), the light colored area will tend to concentrate near the optimal point of the original problem, proving that as μ gets smaller then the impact of the barrier function will be less, the solution of the barrier problem will be closer to the solution of the original problem and our algorithm will gradually reach the optimal point we need to find.

4 Algorithm application

4.1 Application of the interior point method through the transportation problem

4.1.1 Problem introduction

We consider a practical application of the interior point method through the transportation problem with the following context:
In preparation for the 2024 Spring Festival of Nha Trang University - NTU, n classes have been requisitioned to prepare food for the festival. Class i plans to prepare F_i units of raw materials, the raw materials can be purchased from m supermarkets in the city: supermarket j currently has C_j units of raw materials. The cost of purchasing and delivering a unit of raw material from supermarket j to class i's house is P_{ij}.
Summary:

- There are n classes. Each class i needs F_i units of raw material
- There are m supermarkets. Each supermarket j has C_j units of raw material
- The cost of purchasing and delivering 1 unit of raw material from supermarket j to class i is P_{ij}

The goal of the problem is to

(1) purchase and deliver raw materials so that the total cost is minimized

(2) satisfy the needs of all classes

(3) does not exceed the amount of raw materials at each supermarket

4.1.2 Problem Modeling

The above transportation problem is represented as a linear programming problem as follows:

$$\text{Minimize } Z = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}x_{ij}$$

$$\text{Subject to } \sum_{j=1}^{m} x_{ij} = F_i \forall i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} \leq C_j \forall j = 1, ..., m$$

$$x_{ij} \geq 0$$

Where:

- x_{ij} is the number of raw material units purchased from supermarket j to class i with $x_{ij} \geq 0$
- Minimize the objective function $Z = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}x_{ij}$ is the objective (1) of the problem
- Constraint $\sum_{j=1}^{m} x_{ij} = F_i \forall i = 1, ..., n$ is the objective (2) of the problem
- Constraint $\sum_{i=1}^{n} x_{ij} \leq C_j \forall j = 1, ..., m$ is the objective (3) of the problem

4.2 Illustrative problem

4.2.1 Illustrative problem introduction

Suppose there are 2 classes and 2 supermarkets.

The classes need the following raw materials: Class 1 needs 20 raw material units, class 2 needs 30 raw material units.

The amount of raw materials available at the supermarkets is as follows: Supermarket 1 has 40 raw material units, supermarket 2 has 60 raw material units.

The purchasing and transportation costs are as follows

	Supermarket 1	Supermarket 2
Class No.1	30	20
Class No.2	25	15

Table 1: Purchase and shipping costs (unit: thousand VND))

Hence,

- $F_1 = 20$ and $F_2 = 30$
- $C_1 = 40$ and $F_2 = 60$
- $P_{11} = 30$, $P_{12} = 20$, $P_{21} = 25$,$P_{22} = 15$

Mathematical representation of the above illustrated problem:

$$\begin{aligned} \text{Minimize } Z &= 30x_{11} + 20x_{12} + 25x_{21} + 15x_{22} \\ \text{Subject to } & x_{11} + x_{12} = 20 \\ & x_{21} + x_{22} = 30 \\ & x_{11} + x_{21} \leq 40 \\ & x_{12} + x_{22} \leq 60 \\ & x_{ij} \geq 0 \end{aligned}$$

Putting the above representation into the form of an objective function maximization problem, we have:

$$\begin{aligned} \text{Maximize } Z &= -30x_{11} - 20x_{12} - 25x_{21} - 15x_{22} \\ \text{Subject to } & x_{11} + x_{12} = 20 \\ & x_{21} + x_{22} = 30 \\ & x_{11} + x_{21} \leq 40 \\ & x_{12} + x_{22} \leq 60 \\ & x_{ij} \geq 0 \end{aligned}$$

4.2.2 Apply the installed algorithm and solve the illustrated problem

We check the solution of the above problem using the linprog function of the `scipy` library:

```
import scipy.optimize as so
from scipy.optimize import linprog
import numpy as np

# Transportation problem
c = np.matrix([30, 20, 25, 15])
A_ub = np.matrix([
    [1, 0, 1, 0],
```

```
    [0, 1, 0, 1]
], dtype=float)
b_ub = np.matrix([40, 60])

# Define A_eq and B_eq
A_eq = np.matrix([
    [1, 1, 0, 0],
    [0, 0, 1, 1]
])
b_eq = np.matrix([20,30])

# Defind bounds
# Because the bounds is the same as default setting, so we don't have to
    define them
bounds = [
    (0, None)
    for i in range(len(c))
]

# With enough parameters, we use function scipy.optimize.linprog
# to solve the above linear programming problem
result = linprog(
    c=c,
    A_ub=A_ub,
    b_ub=b_ub,
    A_eq=A_eq,
    b_eq=b_eq,
    bounds=bounds,
)

if result.success:
    print("Optimal value:", result.fun)
    print("Optimal solution:", result.x)
else:
    print("Error:", result.message)
```

After running the above code, we get the result:

```
    Optimal value: 850.0
    Optimal solution: [ 0.  20.   0.  30.]
```

Here we use the scipy library to solve the problem, since our implementation does not work well for linear programming problems with two or more variables. This will be a future goal of the team.

References

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[2] I.J. Lustig. Feasibility issues in a primal-dual interior-point method for linear programming. *Mathematical Programming*, 49(1-3):145–162, 1991.

[3] Jirí Matoušek and Bernd Gärtner. *Understanding and Using Linear Programming*. January 2007.

[4] Joel Sobel. Linear programming notes viii: The transportation problem. https://econweb.ucsd.edu/~jsobel/172aw02/notes8.pdf, 2002.