Algorithms Research 2014, 3(1): 1-25
DOI: 10.5923/j.algorithms.20140301.01

T-Forward Method: A Closed-Form Solution and Polynomial Time Approach for Convex Nonlinear Programming

Gang Liu

Technology Research Department, Macrofrontier, Elmhurst, New York

Abstract We present a closed-form solution for convex Nonlinear Programming (NLP). It is closed-form solution if all the constraints are linear, quadratic, or homogeneous. It is polynomial when applied to convex NLP. It gives exact optimal solution when applied to LP. The T-forward method moves forward inside the feasible region with a T-shape path toward the increasing direction of the objective function. Each T-forward move reduces the residual feasible region at least by half. The T-forward method can solve an LP with 10,000 variables within seconds, while other existing LP algorithms will take millions of years to solve the same problem if run on a computer with 1GHz processor.

Keywords Nonlinear programming, Linear programming, Quadratic programming, Interior point, Simplex, Polyhedron, polynomial, Closed-form solution, Basic feasible solution

1. Introduction

A standard nonlinear optimization problem (NLP) can be formulated as:

$$\text{Standard NLP: } \begin{cases} \textit{Maximize}: & f(\vec{x}) \\ \textit{subject to}: & h_i(\vec{x}) \le 0 \quad i = 1, 2, \cdots, N \\ \textit{subject to}: & x_k \ge 0 \quad k = 1, 2, \cdots, M \end{cases} \quad (1)$$

Which involves M variables and N constraints.

Generic NLP is hard to solve. It has been divided into several sub-set problems. Linear programming (LP) is one of the simplest sub-set of NLP, and has got concentrations for researchers.

During World War II, Dantzig and Motzkin first developed the simplex method, which has become one of the most famous algorithm since then [12]. The Journal Computing in Science and Engineering listed the simplex algorithm as one of the top 10 algorithms of the twentieth century [5].

Khachiyan proposed ellipsoid method in 1979[10, 11]. In 1984, Karmarkar started the so called interior-point revolution (See e.g. [9, 17]). There are also other successful algorithms for LP, such as primal-dual interior point method [13], logarithmic method [14], etc.

Most of the algorithms that are successful in LP have been extended to NLP, especially the interior-point method [3], primal-dual method [6, 8], barrier methods[7, 15], all have found their roles for solving NLP.

Currently, most of the LP or NLP algorithms are varieties of simplex or interior-point methods [4, 16].

Most of the current existing LP algorithms as well as NLP algorithms are path oriented and matrix oriented [1, 2]. Path oriented algorithm tries to search the optimal solution in the set of Basic Feasible Solution (BFS), which is the set of all feasible region defined by all of the constraints. However, the BFS is not obvious, thus needs to be calculated or maintained repeatedly. To maintain or to calculate BFS is one of the major difficulties and takes most of the running time for those algorithms. Also, the path oriented algorithm needs to search vertex by vertex along the edges. It could result a long path or ends up with a loop. Another common property for those existing algorithms is matrix oriented, which includes a lot of matrix operations in finding the basic feasible solutions.

Convergence and exactness are two of the most important factors for an LP or NLP algorithm. Simplex method can give exact optimal solution in general. However, it is not guaranteed to give solution at polynomial time. Ellipsoid method guarantees a polynomial upper bound on convergence at the order $O(N^6 L^2 \ln L \ln\ln L)$ [10], and interior point method improved the convergence time to $O(N^{3.5} L^2 \ln L \ln\ln L)$ [9]. However, these polynomial time algorithms may give ε-approximated solution, instead of exact optimal solution. People are still looking for new methods with better convergence and can give exact optimal

* Corresponding author:
gang.liu.1989@gmail.com (Gang Liu)
Published online at http://journal.sapub.org/algorithms

solution.

We present a new approach, T-forward method, which can solve NLP in polynomial time with convergence speed $O(NL\ln L)$, and can give exact optimal solution when applied to LP. We first construct the closed-form solution for the feasible region defined by all the constraints. Having the closed-form basic feasible solution in hand, there is no need to search paths and no need to do any matrix operations, and there is no need to save any feasible solutions in computer system. We can always get it through the closed-form formula.

The closed-form solution for the BFS plays key role in the methods proposed by this paper. In fact, once we have the closed-form solution for BFS, we can transform the original optimization problem into an unconstrained problem or an equation problem.

The rest of the paper is organized as follows. Section 2 discusses how to convert a generic NLP to an NLP with linear objective function. The constraints in an NLP define a feasible region. Section 3 analyzes basic concepts and properties related to a region. Section 4 analyzes the feasible region defined by a single constraint. Section 5 further analyzes the feasible region defined by all constraints and gives the closed-form solution for the feasible region for convex NLP. Section 6 gives the gradient function of the feasible region. The closed-form formulae for the feasible region and infeasible region are summarized in section 7. Section 8 introduces the concepts of the dual-point and the dual-direction. Section 9 introduces the T-Forward method, which gives closed-form solution for convex NLP. Section 10 introduces the greedy T-Forward method, which is a simplified version of the T-Forward method. Section 11 introduces the Facet-Forward method, which can solve LP in polynomial time. Section 12 gives closed-form solution for LP, Quadratic Programming, and NLP with homogeneous constraints. Section 13 presents method for solving nonconvex NLP. Section 14 gives the generic algorithm for T-Forward method. In Section 15, we apply the T-forward method to solve some example optimization problems. In Section 16, we analyze the complexity of the T-forward method and compare it with some existing LP algorithms. Conclusions are presented in Section 17.

2. Converting Generic NLP into an NLP with Linear Objective

A standard NLP can always be converted into an NLP with linear objective function through the following transform:

$$\text{NLP:}\quad \begin{cases} Maximize: & f(\vec{x}) = x_{M+1} \\ Subject\ to: & x_{M+1} - f(\vec{x}) \le 0 \\ Subject\ to: & h_i(\vec{x}) \le 0 \qquad for\ i = 1, 2, \cdots, N \end{cases} \tag{2}$$

In fact, there should have another constraint: $f(\vec{x}) - x_{M+1} \le 0$. However, this constraint is redundant when we try to maximize x_{M+1}. The NLP problem listed in Equation (2) is a standard NLP with Linear (NLPL) objective function. So, we only need to deal with NLP in the following format:

$$\text{NLPL:}\quad \begin{cases} Maximize: & f(\vec{x}) = \vec{c} \bullet \vec{x} \quad a\ linear\ function \\ Subject\ to: & g_i(\vec{x}) \le 0 \quad for\ i = 1, 2, \cdots, N \end{cases} \quad \text{for} \quad \vec{x} \in R^M \tag{3}$$

Here, we have dropped the constraints $x_k \ge 0$ to make the problem more generic. If we want to include the constraints $x_k \ge 0$, we can replace $\vec{x} \in R^M$ with $\vec{x} \in R_+^M$, where R^M is the set of all M –dimensional real numbers, and R_+^M is the set of all M –dimensional real positive numbers. We may also use the following notations:

$$R_+^M = \{\vec{x}: \quad x_k \ge 0,\ for\ k = 1, 2, \cdots, M\} \tag{4}$$

$$R_-^M = \{\vec{x}: \quad x_k < 0,\ for\ k = 1, 2, \cdots, M\} \tag{5}$$

$$R_\pm^M = R_+^M \bigcup R_-^M \tag{6}$$

3. Definitions Regarding a Region

The basic theory and formula in this paper are mainly based on the feasible and infeasible regions defined by the constraints within an NLPL. We first define some useful concepts or notations regarding a region $A \in R^M$.

DEFINITION: **Valid path**: Given a path $C \in R^M$, path C is called valid path if $C \in A$.

DEFINITION: **Valid straight path**: A valid path and the path is a straight line.

DEFINITION: **Connected region**: A region $A \in R^M$ is called connected region if all points in that region can be connected through a valid path.

DEFINITION: **Convex region**: A region $A \in R^M$ is called a convex region if any two points in region A can be connected through a valid straight path. A convex region can be expressed in mathematics format as:

$$\forall \vec{x}^1 \in A, \forall \vec{x}^2 \in A, \forall \mu \in R_+^1, \mu \in [0,1] \;\Rightarrow\; \mu\vec{x}^1 + (1-\mu)\vec{x}^2 \in A \tag{7}$$

Starting from any point in a convex region, we can reach any other points in that region through a valid straight path.

DEFINITION: **Mathematic formulation for boundary curve**: A function $G(\vec{x})$ is called a mathematic formulation for region $A \in R^M$ if it can be used to formulate region A in the following format:

$$A = \{\, \vec{x} : G(\vec{x}) \le 0, \vec{x} \in R^M \,\} \tag{8}$$

In the rest of this section, we assume $G(\vec{x})$ is a mathematic formulation for region A.

DEFINITION: **Boundary**: If A is formulated as the above equation, the following set Ω is called the boundary of A:

$$\Omega = \{\, \vec{x} : G(\vec{x}) = 0, \vec{x} \in R^M \,\} \tag{9}$$

DEFINITION: **Shine region of a point**: Given a point $\vec{x} \in A$, there is a region $A(\vec{x}) \subseteq A$ in which all points can be connected to $\vec{x}$ through valid straight path. $A(\vec{x})$ is called the shine region of point $\vec{x}$.

DEFINITION: **Sun shine set:** Given a point set $\vec{x}^k \in A$ for $k = 1, 2, \cdots, K$ the set $\{\vec{x}^k\}$ is called a sun-shine set if:

$$\bigcup_{k=1}^{K} A(\vec{x}) = A \tag{10}$$

That means the shine regions of a sun-shine set can cover all points in region A.

DEFINITION: **Order of direct connectedness:** The smallest number of points of a sun-set. For example, any convex region can be shined by any point in the region, and thus the order of direct connectedness is 1. The order of direct connectedness is 1 for a circle region, and infinity for a circle line curve.

DEFINITION: **Ray set:** Given a point $\vec{x}^0 \in R^M$, a unit vector $\hat{\vec{z}} \in R^M$ with $\left|\hat{\vec{z}}\right| = 1$, the ray line starting from point $\vec{x}^0$ along the direction $\hat{\vec{z}}$ is called an ray set and denoted as $\Xi(\vec{x}^0, \hat{\vec{z}})$. $\Xi(\vec{x}^0, \hat{\vec{z}})$ can be formulated as:

$$\Xi(\vec{x}^0, \hat{\vec{z}}) = \{\vec{x} : \vec{x} = \vec{x}^0 + \lambda\hat{\vec{z}} \;\; for \; \forall \lambda \in R_+^1\} \tag{11}$$

DEFINITION: **Ray-boundary point:** given a ray set $\Xi(\vec{x}^0, \hat{\vec{z}})$, the shortest (ordered by λ) intersection point of $\Xi(\vec{x}^0, \hat{\vec{z}})$ and the boundary Ω is called the ray-boundary point of the pair ($\vec{x}^0, \hat{\vec{z}}$) and denoted as $\vec{\Omega}(\vec{x}^0, \hat{\vec{z}})$. Within the ray-boundary point function, $\vec{x}^0$ is called the start point, $\hat{\vec{z}}$ is called the moving direction. $\vec{\Omega}(\vec{x}^0, \hat{\vec{z}})$ can be formulated as:

$$\vec{\Omega}(\vec{x}^0, \hat{\vec{z}}) = \vec{x}^0 + \hat{\vec{z}}\breve{G}(\vec{x}^0, \hat{\vec{z}}) \tag{12}$$

Where $\breve{G}(\vec{x}^0, \hat{\vec{z}})$ denotes the smallest non-zero positive root of λ in equation $G(\vec{x}^0 + \lambda\hat{\vec{z}}) = 0$, or $+\infty$ if it doesn't have a positive solution. It can be expressed in mathematics format as:

$$\breve{G}(\vec{x}^0, \hat{\vec{z}}) = \min\left(+\infty, \quad \min\{t : G(\vec{x}^0 + t\hat{\vec{z}}) = 0 \;\; for \; t \in R_+^1 \;\; and \; t > 0\}\right) \tag{13}$$

DEFINITION: **Line region:** The intersection set of $\Xi(\vec{x}^0, \hat{\vec{z}})$ and A is called the line region associated to the pair of

($\vec{x}^0, \hat{\vec{z}}$), and denoted as $A(\vec{x}^0, \hat{\vec{z}})$. $A(\vec{x}^0, \hat{\vec{z}})$ can be formulated as:

$$A(\vec{x}^0, \hat{\vec{z}}) = \{\vec{x} : \vec{x} = \vec{x}^0 + \lambda \hat{\vec{z}} \ for \ \forall \lambda \in R_+^1 \ and \ \lambda \le \breve{G}(\vec{x}^0, \hat{\vec{z}})\} \tag{14}$$

The ray-boundary point $\vec{\Omega}(\vec{x}^0, \hat{\vec{z}})$ is the boundary point that can be reached and connected from point $\vec{x}^0$ and along the direction $\hat{\vec{z}}$. $\vec{\Omega}(\vec{x}^0, \hat{\vec{z}})$ is the basic function and representation for the boundary. We will try to build the $\vec{\Omega}(\vec{x}^0, \hat{\vec{z}})$ for NLPL in following sections.

4. Regions Defined by a Constraint

The i[th] constraint in the NLPL as shown in Equation (3) defines a feasible region in R^M. Let Θ^i denote the feasible region defined by the i[th] constraint in Equation (3), $\overline{\Theta}^i$ denote the infeasible region (with the boundary included even they are feasible) defined by the same constraint, and Ω^i denote the common boundary of Θ^i and $\overline{\Theta}^i$. Note that, as a convention, we allow the boundary Ω^i to be included in both the feasible region Θ^i and infeasible region $\overline{\Theta}^i$. We may frequently use Ω^i to represent the i[th] constraint. By definition, Θ^i, $\overline{\Theta}^i$, and Ω^i can be formulated as:

$$\Theta^i = \{\vec{x} : g_i(\vec{x}) \le 0, \ for \ \forall \vec{x} \in R^M\} \tag{15}$$

$$\overline{\Theta}^i = \{\vec{x} : g_i(\vec{x}) \ge 0, \ for \ \forall \vec{x} \in R^M\} \tag{16}$$

$$\Omega^i = \{\vec{x} : g_i(\vec{x}) = 0, \ for \ \forall \vec{x} \in R^M\} \tag{17}$$

$$\Theta^i \bigcup \overline{\Theta}^i = R^M \tag{18}$$

$$\Theta^i \bigcap \overline{\Theta}^i = \Omega^i \tag{19}$$

Given a point $\vec{x} \in R^M$, if $g_i(\vec{x}) = 0$ and thus $\vec{x} \in \Omega^i$, $\vec{x}$ is called a **boundary point** of constraint Ω^i, and the constraint Ω^i is called a **boundary constraint** at point $\vec{x}$. If $g_i(\vec{x}) < 0$ and thus $\vec{x} \in \Theta^i$, $\vec{x}$ is called a **feasible point** to constraint Ω^i, and the constraint Ω^i is called a **valid constraint** at $\vec{x}$. If $g_i(\vec{x}) > 0$ and thus $\vec{x} \in \overline{\Theta}^i$, $\vec{x}$ is called an **infeasible point** to constraint Ω^i, and the constraint Ω^i is called an **invalid constraint** at $\vec{x}$.

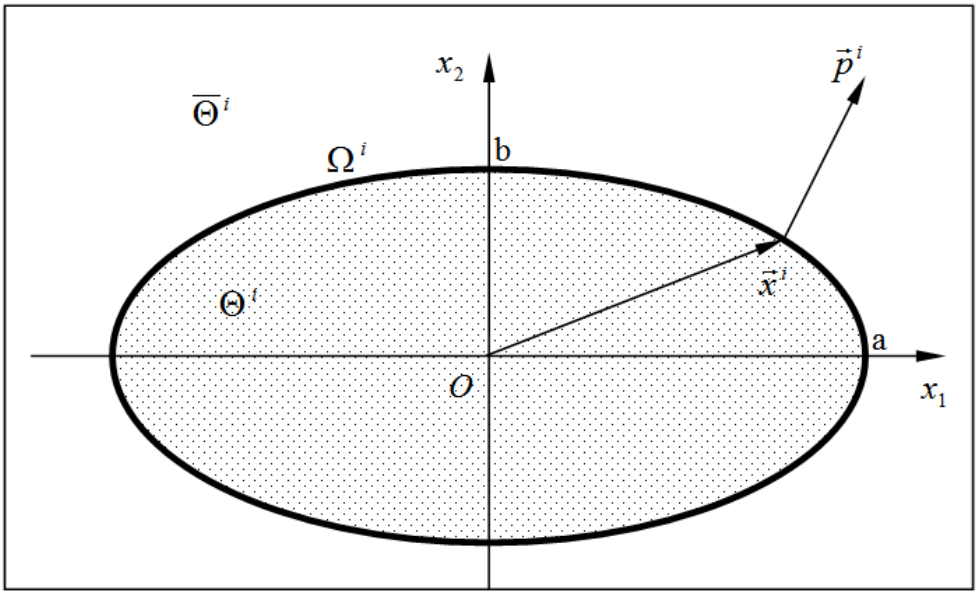

Figure 1. The feasible region Θ^i and boundary Ω^i defined by a constraint in ellipsoid type. $\vec{p}^i$ is the gradient vector of Ω^i at point $\vec{x}^i$

A constraint Ω^i is called a **simple constraint** if it has only one feasible region OR only one infeasible region. In this paper, we assume all constraints are simple constraints.

The ray-boundary point $\vec{\Omega}^i(\vec{x}^0, \hat{\vec{z}})$ can be formulated as:

$$\vec{\Omega}^i(\vec{x}^0, \hat{\vec{z}}) = \vec{x}^0 + \hat{\vec{z}}\breve{g}_i(\vec{x}^0, \hat{\vec{z}}), \quad \text{if } \vec{x}^0 \in \Theta^i \tag{20}$$

The above equation gives the feasible boundary point linked to point $\vec{x}^0$ and along the direction $\hat{\vec{z}}$. Figure 1. illustrates the feasible region Θ^i and feasible boundary Ω^i defined by a constraint in ellipsoid type.

Similarly, given a point $\vec{x} \in \overline{\Theta}^i$, there should exist a region around $\vec{x}$ in which all points are infeasible and can be connected to $\vec{x}$ through straight line valid to $\overline{\Theta}^i$. This region is called infeasible region connected to $\vec{x}$ defined by constraint Ω^i. Let $\overline{\Theta}^i(\vec{x})$ denote this region and $\overline{\Omega}^i(\vec{x})$ denote its boundary. The ray-boundary point $\vec{\overline{\Omega}}^i(\vec{x}^0, \hat{\vec{z}})$ can be formulated as:

$$\vec{\overline{\Omega}}^i(\vec{x}^0, \hat{\vec{z}}) = \vec{x}^0 + \hat{\vec{z}}\breve{g}_i(\vec{x}^0, \hat{\vec{z}}), \quad \text{if } \vec{x}^0 \in \overline{\Theta}^i \tag{21}$$

5. Feasible Region Defined by All the Constraints in NLPL

All the constraints in an NLPL as shown in Equation (3) define an M –dimensional polytope in R^M, which contains some piece-wise curves as its facets, and each piece-wise curve must be a part of one of the feasible boundary Ω^i defined in NLPL. All the points in the feasible region are also called Basic Feasible Solution (BFS).

Let Θ denote the feasible region defined by all the constraints in NLPL, $\overline{\Theta}$ denote the infeasible region defined by all the constraints in the same NLPL, Ω denote the common boundary of Θ and $\overline{\Theta}$. Θ is the set in R^M that have all the constraints satisfied. By definition, we should have:

$$\Theta = \bigcap_{i=1}^{N} \Theta^i \tag{22}$$

$$\overline{\Theta} = \bigcup_{i=1}^{N} \overline{\Theta}^i \tag{23}$$

$$\Theta \bigcup \overline{\Theta}. = R^M \tag{24}$$

$$\Theta \bigcap \overline{\Theta} = \Omega \tag{25}$$

Given a point $\vec{x}^\Theta \in \Theta$, there should have a region that all points can be connected to $\vec{x}$ through a straight line valid to Θ. Let $\Theta(\vec{x}^\Theta)$ denote this region and $\Omega(\vec{x}^\Theta)$ denote its boundary. Similarly, given a point $\vec{x}^{\overline{\Theta}} \in \overline{\Theta}$, there should have a region that all points can be connected to $\vec{x}$ through a straight line valid to $\overline{\Theta}$. Let $\overline{\Theta}(\vec{x}^{\overline{\Theta}})$ denote this region and $\overline{\Omega}(\vec{x}^{\overline{\Theta}})$ denote its boundary. Based on our definition, both $\Theta(\vec{x}^\Theta)$ and $\overline{\Theta}(\vec{x}^{\overline{\Theta}})$ are convex, and

$$\Theta(\vec{x}^\Theta) \subseteq \Theta \tag{26}$$

$$\overline{\Theta}(\vec{x}^{\overline{\Theta}}) \subseteq \overline{\Theta} \tag{27}$$

The line-region $\Theta(\vec{x}^\Theta, \hat{\vec{z}})$ can be formulated as:

$$\Theta(\vec{x}^\Theta, \hat{\vec{z}}) = \Theta \bigcap \Xi(\vec{x}^\Theta, \hat{\vec{z}}) = \bigcap_{i=1}^{N} \Theta^i(\vec{x}^\Theta, \hat{\vec{z}}) \tag{28}$$

$$\Theta(\vec{x}^{\Theta},\hat{\vec{z}})=\bigcap_{i=1}^{N}\Theta^{i}(\vec{x}^{\Theta},\hat{\vec{z}})=\{\vec{x}:\ \vec{x}=\vec{x}^{\Theta}+\hat{\vec{z}}\min_{1\le i\le N}\breve{g}_{i}(\vec{x}^{\Theta},\hat{\vec{z}})\} \tag{29}$$

Thus, the ray-boundary point $\vec{\Omega}(\vec{x}^{\Theta},\hat{\vec{z}})$ can be formulated as:

$$\vec{\Omega}(\vec{x}^{\Theta},\hat{\vec{z}})=\vec{x}^{\Theta}+\hat{\vec{z}}\breve{g}(\vec{x}^{\Theta},\hat{\vec{z}}) \tag{30}$$

where

$$\breve{g}(\vec{x}^{\Theta},\hat{\vec{z}})=\min_{1\le i\le N}\breve{g}_{i}(\vec{x}^{\Theta},\hat{\vec{z}}) \tag{31}$$

Note that all items within the min operation in the above equation are positive and none-zeros. Then the above equation can be equivalently transformed into:

$$\breve{g}(\vec{x}^{\Theta},\hat{\vec{z}})=\hat{\vec{z}}\sum_{i=1}^{N}\breve{g}_{i}(\vec{x}^{\Theta},\hat{\vec{z}})\Phi\left(\left\{\left(\breve{g}_{i}(\vec{x}^{\Theta},\hat{\vec{z}})\right)^{-1}\right\}\right) \tag{32}$$

Where $\Phi(\{x_i\})$ is the following function:

$$\Phi\left(\{x_i\}\right)=\frac{\delta\left(\max\limits_{1\le i\le N}(x_i)-x_i\right)}{\sum\limits_{i=1}^{N}\delta\left(\max\limits_{1\le i\le N}(x_i)-x_i\right)}\ ;\ x_i>0\ \text{ for }\ i=1,2,\cdots,N \tag{33}$$

And $\delta(x)$ is an $R^1\to R^1$ function defined as:

$$\delta(x)=\begin{cases}1 & if\quad x=0\\ 0 & otherwise\end{cases} \tag{34}$$

Function $\Phi(\{x_i\})$ has the following property:

$$\sum_{i=1}^{N}\Phi\left(\{x_i\}\right)=1 \tag{35}$$

Then, $\Phi(\{x_i\})$ can be interpreted as a probability partition function of the set $\{x_i\}$ with non-zero probability when $x_i=\max\limits_{i}\{x_i\}$, and 0 probability for other cases. $\Phi(\{x_i\})$ can also be expressed in the following format:

$$\Phi\left(\{x_i\}\right)=\lim_{\beta\to+\infty}\frac{\exp(\beta x_i)}{\sum\limits_{i=1}^{N}\exp(\beta x_i)} \tag{36}$$

To make it easy for use, we can simply drop the "lim" operation by replacing β with a large number. Then, the above equation can be written as:

$$\Phi(\{x_i\})\approx\overline{\Phi}(\{x_i\}),\quad \overline{\Phi}\left(\{x_i\}\right)=\frac{\exp(\beta x_i)}{\sum\limits_{i=1}^{N}\exp(\beta x_i)};\ \beta=\overline{\beta}\Big/\max_{1\le i\le N}(x_i) \tag{37}$$

$\overline{\beta}$ is a large number such that $\exp(\overline{\beta})$ is acceptable for computer system without data over flow. Normally, $\overline{\beta}=500$ is good enough for numerical calculations.

$\Phi(\{x_i\})$ is called **exact-partition** function, whereas $\overline{\Phi}(\{x_i\})$ is called **smooth-partition** function. Here we use

$\overline{\Phi}(\{x_i\})$ to denote the smooth format partition function, so that it can be distinguished from the exact format. In fact, they give almost the same values once $\overline{\beta}$ is big enough. In the rest of the paper, as long as there is no confusion, we will use the same notation $\Phi(\{x_i\})$ for both smooth and exact format partition functions.

Using the partition function, the ray-boundary point $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ in both exact and smooth formats becomes:

$$\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \vec{x}^{\Theta} + \hat{\vec{z}} \sum_{i=1}^{N} \breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}}) \Phi\left(\left\{ \left(\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}}) \right)^{-1} \right\} \right), \text{ if } \vec{x}^{\Theta} \in \Theta \tag{38}$$

Suppose $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ has been solved from the equation $g^i(\vec{x}^{\Theta} + \lambda^i \hat{\vec{z}}) = 0$ for all $i = 1, 2, \cdots, N$, then both Equation (30) and (38) give closed-form solution for a feasible point $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$, which is the ray-boundary point of $(\vec{x}^{\Theta}, \hat{\vec{z}})$. $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ will give all points of the shine-region $\Omega(\vec{x}^{\Theta})$ once point $\hat{\vec{z}}$ goes through all possible directions. Further, once point $\vec{x}^{\Theta}$ goes through all points of a sun set, $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ will give all the points on feasible boundary Ω. Therefore, Equation (30), or its smooth format Equation (38), is a closed-form solution for the feasible boundary Ω or BFS defined by all constraints in an NLPL.

6. Gradient Function of the Feasible Boundary

In real applications, we may frequently need the gradient of the feasible boundary Ω. Since we already have the closed-form solution for the feasible boundary, especially its smooth format, the gradient function can be calculated by differentiating the boundary function. However, here we give another simple method to calculate the gradient function.

Keeping in mind that partition function $\Phi\left(\left\{ \left(\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}}) \right)^{-1} \right\} \right)$ is the probability distribution function for any constraint Ω^i to be on the boundary Ω. Then, we can use the partition function as a weight or probability distribution to calculate the gradient function. Let $\Lambda(\vec{x}^{\Theta}, \hat{\vec{z}})$ denote the gradient of the feasible boundary curve Ω at the point $\Omega(\vec{x}^{\Theta}, \hat{\vec{z}})$. Then it can be formulated as:

$$\vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \sum_{i=1}^{N} \vec{\Lambda}^i(\vec{x}^{\Theta}, \hat{\vec{z}}) \Phi\left(\left\{ \left(\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}}) \right)^{-1} \right\} \right) \tag{39}$$

Where $\vec{\Lambda}^i(\vec{x}^{\Theta}, \hat{\vec{z}})$ is the unit gradient vector of the constraint curve Ω^i at the point $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$. $\vec{\Lambda}^i(\vec{x}^{\Theta}, \hat{\vec{z}})$ can be formulated as:

$$\vec{\Lambda}^i(\vec{x}^{\Theta}, \hat{\vec{z}}) = \vec{p} / |\vec{p}|, \text{ where } \vec{p}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \nabla g_i(\vec{x})\big|_{\vec{x}=\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})}, \tag{40}$$

$\Lambda(\vec{x}^{\Theta}, \hat{\vec{z}})$ gives the average gradient of all the curves passing through the point $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ on the feasible boundary Ω. It could be slightly different from the gradient calculated by differentiating the boundary function. However, they will be the same as long as the corresponding boundary point contains only one constraint, which is most of the case.

Although the smooth format might be slightly different from the exact format, it always gives a good approximation when $\overline{\beta}$ is big enough. The exact format gradient function may not be continuous, while the smooth format is always smooth and continuous. When calculating the gradient, it would be better to use the smooth format partition function.

7. Summary of Equations for Boudaries Defined by All Constraints

We have formulated the closed-form solution for the feasible boundary Ω and its gradient. Similarly, we can construct the infeasible boundary $\overline{\Omega}$ and its gradient by formulating the ray point $\vec{\overline{\Omega}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$ starting from an infeasible point

$\vec{x}^{\overline{\Theta}} \in \overline{\Omega}$ and along the moving direction $\hat{\vec{z}}$. As a summary, let us list the boundary functions and the gradient functions for Ω and $\overline{\Omega}$ as following:

$$\text{Feasible boundary:} \quad \vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \vec{x}^{\Theta} + \hat{\vec{z}} \sum_{i=1}^{N} t_i \Phi\left(\left\{(t_i)^{-1}\right\}\right), \quad \text{if } \vec{x}^{\Theta} \in \Theta \tag{41}$$

$$\text{Infeasible boundary:} \quad \vec{\overline{\Omega}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}}) = \vec{x}^{\overline{\Theta}} + \hat{\vec{z}} \sum_{i=1}^{N} t_i \Phi\left(\{t_i\}\right), \quad \text{if } \vec{x}^{\overline{\Theta}} \in \overline{\Theta} \tag{42}$$

$$\text{Feasible gradient:} \quad \vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \sum_{i=1}^{N} \vec{\Lambda}^i \Phi\left(\left\{(t_i)^{-1}\right\}\right), \quad \text{if } \vec{x}^{\Theta} \in \Theta \tag{43}$$

$$\text{Infeasible gradient:} \quad \vec{\overline{\Lambda}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}}) = \sum_{i=1}^{N} \vec{\Lambda}^i \Phi\left(\{t_i\}\right), \quad \text{if } \vec{x}^{\overline{\Theta}} \in \overline{\Theta} \tag{44}$$

Supplementary functions:

$$\text{Positive root of constraint:} \quad t_i = \breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}}) = \min\{t : g_i(\vec{x}^{\Theta} + t\hat{\vec{z}}) = 0, t > 0\} \tag{45}$$

$$\text{Exact partition function:} \quad \Phi\left(\{t_i\}\right) = \frac{\delta\left(\max\limits_{1 \le i \le N}(t_i) - t_i\right)}{\sum\limits_{i=1}^{N} \delta\left(\max\limits_{1 \le i \le N}(t_i) - t_i\right)}; \; t_i > 0; \text{ for } i = 1,2,\cdots,N \tag{46}$$

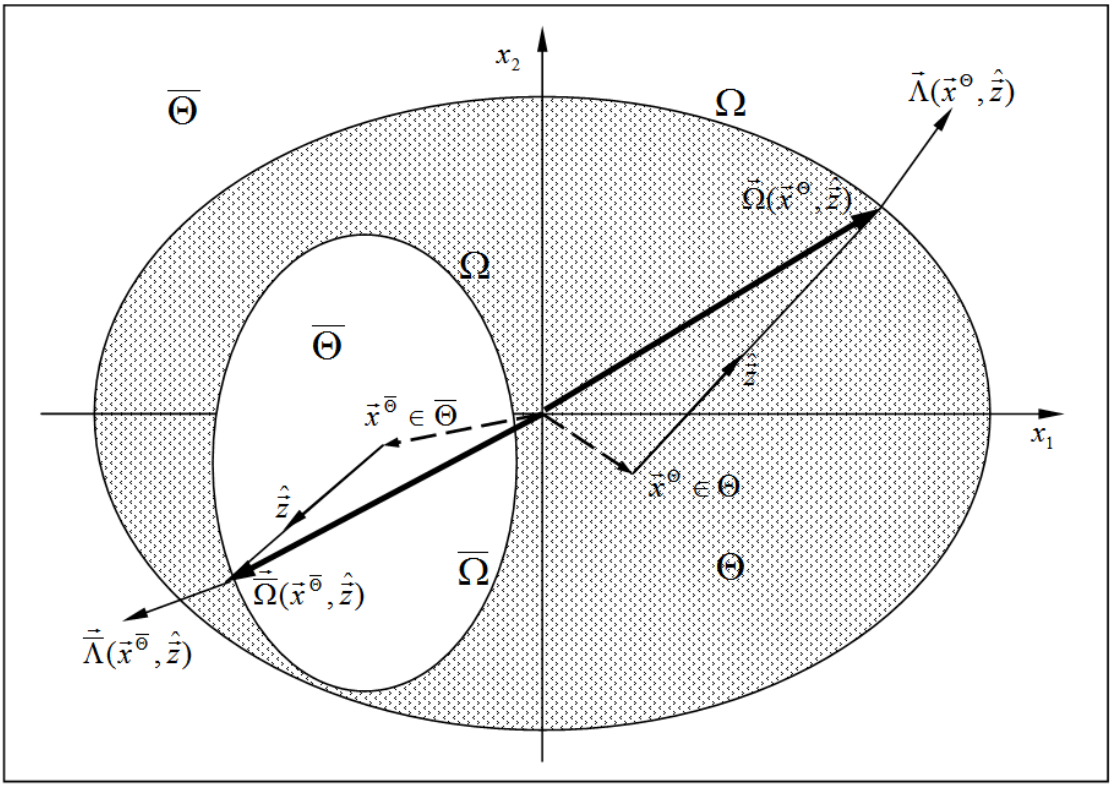

Figure 2. The right side shows the feasible boundary Ω, one of its ray-boundary point $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$, and its gradient $\vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}})$. The left side shows the infeasible boundary $\overline{\Omega}$, one of its ray-boundary point $\vec{\overline{\Omega}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$, and its gradient $\vec{\overline{\Lambda}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$

Smooth partition function: $$\Phi\left(\{t_i\}\right) \approx \frac{\exp\left(\beta t_i\right)}{\sum_{i=1}^{N} \exp\left(\beta t_i\right)} \tag{47}$$

Gradient of constraint Ω^i : $$\vec{\Lambda}^i = \vec{p}/|\vec{p}|, \quad \vec{p}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \nabla g_i(\vec{x})\big|_{\vec{x}=\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})} \tag{48}$$

These are the major functions we are going to use in the rest of this paper. Figure 2. illustrates some vectors or points related to the boundary curve and can be calculated by the above equations.

8. Dual-point and Dual-direction

Using the boundary functions given in the previous section, we can start from a point $\vec{x}^{\Theta}$ to find a feasible ray-boundary point. If $\vec{x}^{\overline{\Theta}}$ is an infeasible point, we can apply function $\vec{\Omega}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$ to get a feasible point.

Suppose $\vec{x}^{\Omega} \in \Omega$ is a feasible boundary point we have found. Let B denote the boundary point and the ending point of $\vec{x}^{\Omega}$ as shown in Figure 3.

Let Π_f^0 denote the contour plane of the objective function passing through point B, which can be formulated as $f(\vec{x}) = f(\vec{x}^0)$. Let $\vec{n}_f$ denote the unit gradient vector of Π_f^0, Ω^0 denote the tangent plane of the boundary Ω at point B, and $\vec{n}_{\Omega}^0$ denote the unit gradient vector of Ω at point B. Let vector $\vec{v}$ be the projected vector of $\vec{n}_{\Omega}^0$ onto Π_f^0, point *D* be the intersection point of $-\vec{v}$ with Ω, *C* be the mid point between B and *D*.

Figure 3. Illustrates the points and curves mentioned above.

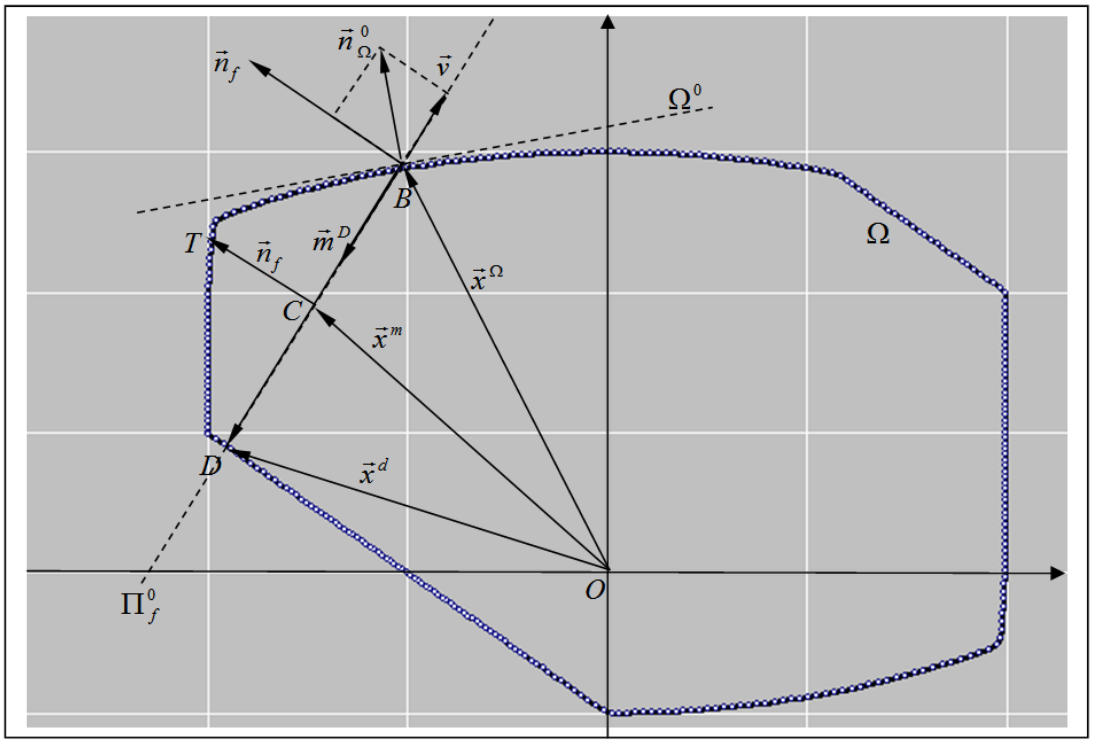

Figure 3. A feasible boundary point $\vec{x}^{\Omega}$, its dual-point $\vec{x}^d$, the mid-interior-point $\vec{x}^m$, and the dual-direction $\vec{m}^D$

***DEFINITION:* dual-direction**. Vector $\vec{m}^D = -\vec{v}/|\vec{v}|$ is called the dual-direction. If $\vec{v} = \vec{0}$, we set the dual-direction to $-\vec{n}_{\Omega}^0$.

***DEFINITION:* dual-point.** Given a boundary point $\vec{x}^{\Omega}$, its ray-boundary point along the dual-direction is called the dual point of $\vec{x}^{\Omega}$, which is the point *D* or $\vec{x}^{d}$ as shown in Figure 3.

***DEFINITION:* mid-interior-point**. The middle point between a boundary point $\vec{x}^{\Omega}$ and its dual point $\vec{x}^{d}$ is called the mid-interior-point of $\vec{x}^{\Omega}$.

Here we borrow the words "interior point" and "dual point" to indicate some special points related to a boundary point and the boundary curves. However, both of them are very different form the interior-point method and the dual method that are frequently used in the optimization world.

Using the boundary gradient function, the gradient vector $\vec{n}_{\Omega}^{0}$ can be calculated as:

$$\vec{n}_{\Omega}^{0} = \vec{n}/|\vec{n}|, \text{ where } \vec{n} = \vec{\Lambda}(\vec{x}^{\Omega}, \hat{\vec{z}}) = \sum_{i=1}^{N} \vec{\Lambda}^{i} \Phi\left(\left\{(t_i)^{-1}\right\}\right) \tag{49}$$

The dual-direction can be easily formulated as:

$$\vec{m}^{D} = \begin{cases} -\vec{n}_{\Omega}^{0} & if \quad |\vec{v}| = 0 \\ -\vec{v}/|\vec{v}|, \quad \vec{v} = \vec{n}_{\Omega}^{0} - \vec{n}_{f}\left(\vec{n}_{f} \bullet \vec{n}_{\Omega}^{0}\right) & if \quad |\vec{v}| \neq 0 \end{cases} \tag{50}$$

The dual-point can be formulated as:

$$\vec{x}^{d} = \vec{\Omega}(\vec{x}^{\Omega}, \vec{m}^{D}) \tag{51}$$

The mid-interior-point can be formulated as:

$$\vec{x}^{m} = \vec{x}^{\Omega} + \frac{\vec{m}^{D}}{2}\left|\vec{x}^{\Omega} - \vec{x}^{d}\right| = \frac{1}{2}\left(\vec{x}^{\Omega} + \vec{x}^{d}\right) \tag{52}$$

9. T-Forward Method for Nonlinear Programming Problems

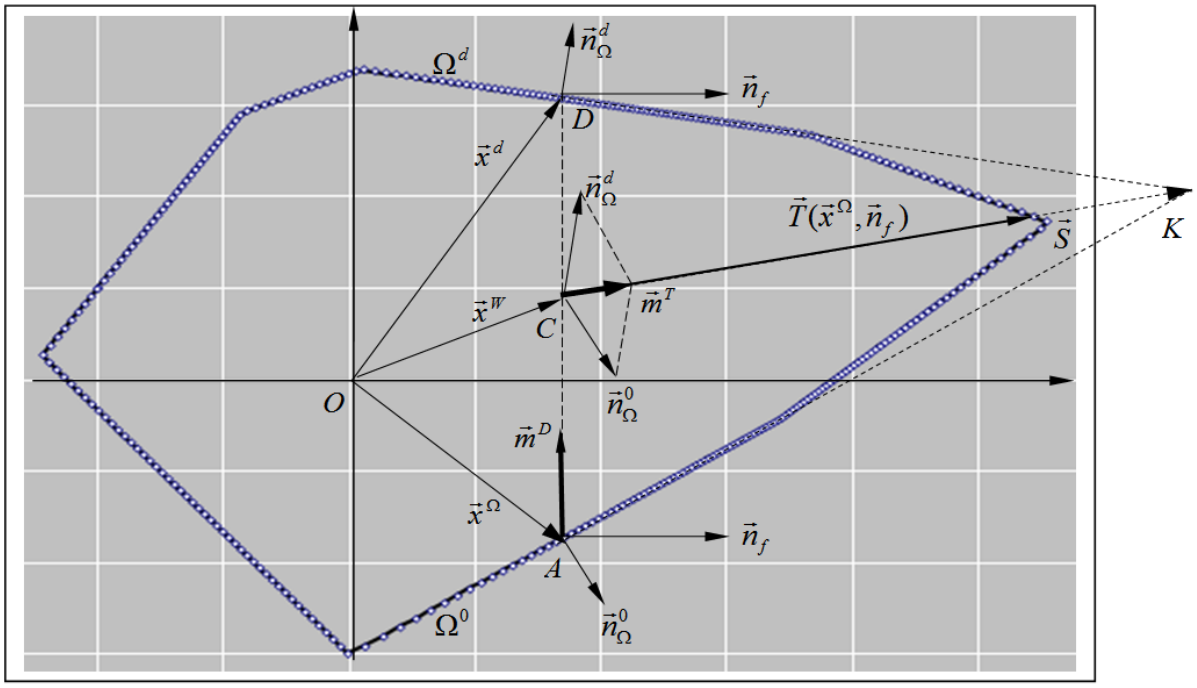

Figure 4. Illustration of the T-Forward vector $\vec{T}(\vec{x}^{\Omega}, \vec{n}_f)$, and how it related to the feasible boundary point $\vec{x}^{\Omega}$, the dual-point $\vec{x}^{d}$, the mid-interior-point $\vec{x}^{m}$, the dual-direction $\vec{m}^{D}$, and the T-forward direction $\vec{m}^{T}$. Point $\vec{T}(\vec{x}^{\Omega}, \vec{n}_f)$ is the T-forward point. Point K represents the common edge of Ω^0 and Ω^d

Building on the closed-form solution for the feasible boundary and for the boundary gradient, we can give a near-closed-form solution for NLP problems. Figure 4. illustrates some of the notations we are going to use in this paper and their relationships.

The following notations will be used in the rest of the paper:

$\vec{x}^{\Theta}$: A feasible point, $\vec{x}^{\Theta} \in \Theta$.

$\vec{x}^{\overline{\Theta}}$: An infeasible point, $\vec{x}^{\overline{\Theta}} \in \Theta$.

$\vec{x}^{\Omega}$: A feasible boundary point, $\vec{x}^{\Omega} \in \Omega$.

$\vec{n}_f$: The gradient of the objective function.

$\vec{x}^{d}$: The dual point of $\vec{x}^{\Omega}$, $\vec{x}^{d} \in \Omega$.

$\vec{x}^{m}$: The mid-interior point of $\vec{x}^{\Omega}$;

Ω^0 : The tangent curve of Ω at the point $\vec{x}^{\Omega}$;

Ω^d : The tangent curve of Ω at the dual-point $\vec{x}^{d}$;

K : The common edge of Ω^0 and Ω^d;

$\vec{n}_{\Omega}^{0}$: The gradient of Ω^0;

$\vec{n}_{\Omega}^{d}$: The gradient of Ω^d;

Π_f^0 : The objective contour plane passing through point $\vec{x}^{\Omega}$ and $\vec{x}^{d}$;

$\vec{m}^{D}$: Dual point direction, $\vec{m}^{D} = \vec{x}^{d} - \vec{x}^{\Omega}$;

$\vec{m}^{T}$: T-forward direction, $\vec{m}^{T} = \vec{n} / |\vec{n}|$, $\vec{n} = (\vec{n}_{\Omega}^{0} + \vec{n}_{\Omega}^{d}) sign\left((\vec{n}_{\Omega}^{0} + \vec{n}_{\Omega}^{d}) \bullet \vec{n}_f\right)$;

$\vec{S}$: The point that gives optimal solution to the NLPL problem; We will use ($\vec{x}^{\Omega}$, $\vec{n}_f$) as the basic variables and have other variables expressed as a function of the pair ($\vec{x}^{\Omega}$, $\vec{n}_f$).

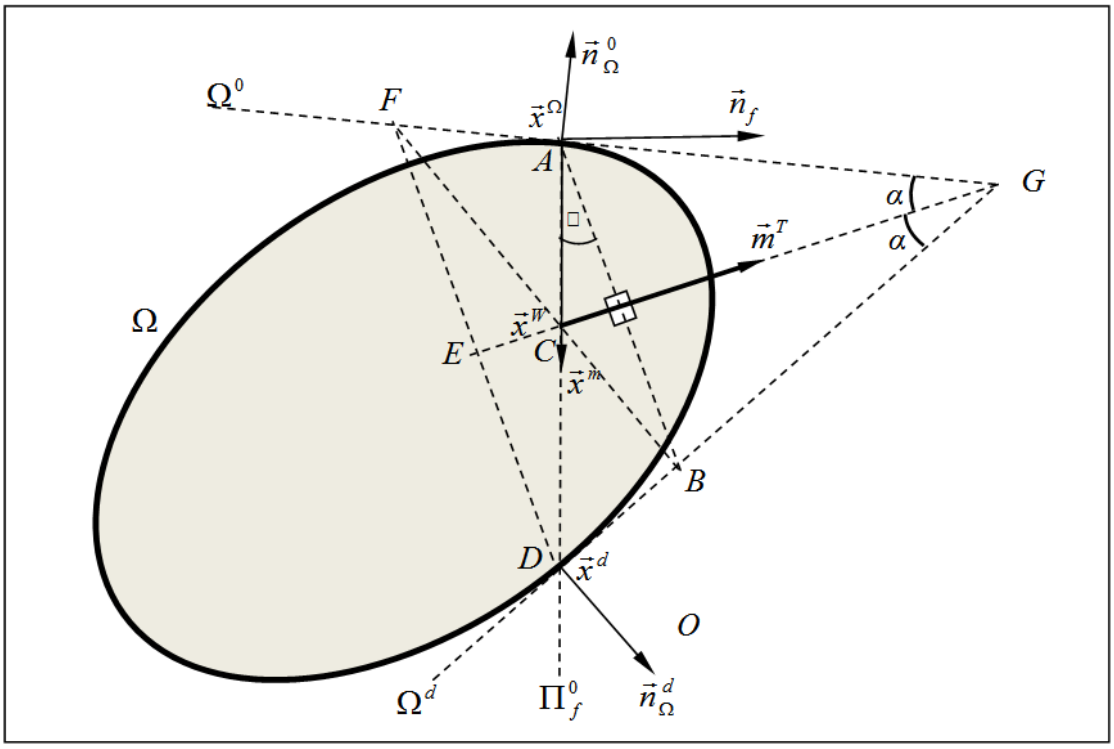

Figure 5. Illustration of the weighted-interior-point $\vec{x}^{W}$ and the mid-interior-point $\vec{x}^{m}$. *A* is the initial point $\vec{x}^{\Omega}$. D is the dual point $\vec{x}^{d}$. The mid-interior-point $\vec{x}^{m}$ is at the middle of the line $\overline{AD}$. *C* is the weighted-interior-point $\vec{x}^{W}$, which divides the line $\overline{AD}$ with μ and $1-\mu$ as the weight factors. Starting from the point C (the end point of $\vec{x}^{W}$), the T-forward direction $\vec{m}^{T}$ will point to the point G, which is the common edge of the tangent curve Ω^0 and the dual tangent curve Ω^d

Figure 5. Illustrates weighted-interior-point $\vec{x}^W$ and how to calculate the weight μ.

DEFINITION: **T-forward-direction.** T-forward direction is defined as:

$$\vec{m}^T = \vec{n}/|\vec{n}|,\ \vec{n} = (\vec{n}_\Omega^0 + \vec{n}_\Omega^d)sign\left((\vec{n}_\Omega^0 + \vec{n}_\Omega^d) \bullet \vec{n}_f\right) \tag{53}$$

DEFINITION: **Weighted-interior-point $\vec{x}^W$.** The weighted-interior-point is defined as:

$$\vec{x}^W = (1-\mu)\vec{x}^\Omega + \mu\vec{x}^d = \vec{x}^\Omega + \mu(\vec{x}^d - \vec{x}^\Omega) \tag{54}$$

$$\text{where } \mu = \frac{\cos(\theta+\alpha)}{2\cos\alpha\cos\theta};\ \alpha = \frac{1}{2}\left(\pi - \arccos\left(\vec{n}_\Omega^0 \bullet \vec{n}_\Omega^d\right)\right);\ \theta = \alpha - \arcsin\left(\vec{n}_\Omega^0 \bullet \vec{n}_f\right) \tag{55}$$

DEFINITION: **T-forward-point.** The ray-boundary point starting from the weighted-interior-point $\vec{x}^W$ and along the T-forward direction $\vec{m}^T$ is called the T-forward-point, denoted as $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$.

Using the weighted-interior-point as starting point and the T-forward-direction as the forward move direction, the ray-boundary-point $\vec{\Omega}(\vec{x}^W, \vec{m}^T)$ will point to the point *K*, which is the common edge of the tangent curve Ω^0 and the dual tangent curve Ω^d. The T-forward-point will try to reach the edge or vertex whenever possible.

DEFINITION: **T-forward-function:** The function maps from the pair $(\vec{x}^\Omega, \vec{n}_f)$ to the T-forward-point $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$ is called **T-forward-function**, which is also denoted as $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$.

By definition, the T-forward-point $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$ can be formulated as:

$$\vec{T}(\vec{x}^\Omega, \vec{n}_f) = \vec{\Omega}(\vec{x}^W, \vec{m}^T) = \vec{\Omega}\left((1-\mu)\vec{x}^\Omega + \mu\vec{x}^d,\ \vec{m}^T\right) = \vec{\Omega}\left((1-\mu)\vec{x}^\Omega + \mu\vec{\Omega}(\vec{x}^\Omega, \vec{m}^D),\ \vec{m}^T\right)$$

Thus, the T-forward-point and the T-forward-function $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$ can be written as:

$$\vec{T}(\vec{x}^\Omega, \vec{n}_f) = \vec{\Omega}\left((1-\mu)\vec{x}^\Omega + \mu\vec{\Omega}(\vec{x}^\Omega, \vec{m}^D),\ \vec{m}^T\right) \tag{56}$$

The T-forward point $\vec{T}(\vec{x}^\Omega, \vec{n}_f)$ is also a boundary point. We can apply the T-forward-function to it to get another T-forward-point. By applying the T-forward-function repeatedly, we will eventually reach a point where the T-forward-point cannot move further. That must be a point where the objective contour plane and the feasible boundary Ω have just one common contact point. That is the optimal solution for the NLP in the shine-region $\Omega(\vec{x}^\Omega)$.

DEFINITION: **T-forward-solution:** Given a feasible boundary point $\vec{x}^\Omega \in \Omega$ and the objective gradient $\vec{n}_f$, let $\vec{S}(\vec{x}^\Omega, \vec{n}_f)$ denote the optimal solution for the NLPL problem in the shine-region $\Theta(\vec{x}^\Omega)$ (with $\Omega(\vec{x}^\Omega)$ as its boundary). By repeatedly applying the T-forward-function, we can reach a solution for the NLPL. This solution is called **T-forward-solution** and denoted as $\vec{S}^T(\vec{x}^\Omega, \vec{n}_f)$.

The T-forward solution can be formulated as:

$$\vec{S}^T(\vec{x}^\Omega, \vec{n}_f) = \vec{T}\left(\cdots\vec{T}\left(\vec{T}(\vec{x}^\Omega, \vec{n}_f), \vec{n}_f\right), \cdots, \vec{n}_f\right) \tag{57}$$

Or expressed in iterative format:

$$\begin{aligned} &\vec{S}^T(\vec{x}^\Omega, \vec{n}_f) = \vec{T}(\vec{x}^K, \vec{n}_f) \\ &\vec{x}^k = \vec{T}(\vec{x}^{k-1}, \vec{n}_f), \quad for \quad k = 1, 2, \cdots, K \end{aligned} \tag{58}$$

DEFINITION: **T-forward point method:** Using the T-forward-function to solve NLPL problem is called T-forward point method, or simply T-forward method.

DEFINITION: **T-forward path:** The series of the T-forward points starting from the initial point $\vec{x}^\Omega$ and ending at the optimal point $\vec{S}(\vec{x}^\Omega, \vec{n}_f)$ make up a path, which is called T-forward path.

DEFINITION: **T-forward accuracy:** The relative difference between $\vec{S}^T(\vec{x}^\Omega, \vec{n}_f)$ and $\vec{S}(\vec{x}^\Omega, \vec{n}_f)$ is called T-forward accuracy, denoted as $\Delta^T(\vec{x}^\Omega, \vec{n}_f)$. That is:

$$\Delta^T(\vec{x}^\Omega, \vec{n}_f) = \left|\vec{S}(\vec{x}^\Omega, \vec{n}_f) - \vec{S}^T(\vec{x}^\Omega, \vec{n}_f)\right| \Big/ \left|\vec{S}(\vec{x}^\Omega, \vec{n}_f)\right| \tag{59}$$

Given a feasible boundary point $\vec{x}^\Omega \in \Omega$, we can find its dual-point $\vec{x}^d$. Both $\vec{x}^\Omega$ and $\vec{x}^d$ are on the same contour plane Π_f^0, and thus give the same objective value. If the feasible region is convex and continuous, all the points between $\vec{x}^\Omega$ and $\vec{x}^d$ should be interior points and thus have some points between the contour plane and the feasible boundary Ω. For those points that are outside the contour plane Π_f^0 and along the direction $\vec{n}_f$, which is the normal direction of Π_f^0, will give better objective values. Particularly, the T-forward-point between $\vec{x}^W$, will possibly give the longest move within the feasible boundary Ω if we move from $\vec{x}^W$ along the direction $\vec{m}^T$. As long as the point $\vec{x}^W$ is a strict interior point, the ray-boundary-point $\vec{\Omega}(\vec{x}^W, \vec{m}^T)$ will definitely give a larger objective value than the point $\vec{x}^0$.

Normally, T-forward-point method gives ε-approximated solution for NLP. For LP problems, the T-forward-point method is designed to move to the edge or vertex if possible. The final solution is always on the optimal edge or vertex. As long as the LP has a solution, the T-forward-point method guarantees to give the exact optimal solution.

10. Greedy T-Forward Method for Nonlinear Programming Problems

The Greedy T-forward method a simplified version of the T-forward method. Everything is the same as the T-forward method except it takes the mid-interior-point $\vec{x}^m$ as the starting point in each T-forward step. So, if we replace $\vec{x}^W$ with $\vec{x}^m$, the T-forward method will become greedy T-forward method. Compared with the T-forward method, the greedy T-forward method doesn't have any advantages except its simplicity. It may work as efficient as the T-forward method. However, it may not give exact optimal solution when applied to LP problems. Everything in the previous section is applicable to the greedy T-forward method. There is no need to repeat them. Here we just list the greedy T-forward function. By definition, the greedy T-forward-point $\vec{G}(\vec{x}^\Omega, \vec{n}_f)$ can be formulated as:

$$\vec{G}(\vec{x}^\Omega, \vec{n}_f) = \vec{\Omega}(\vec{x}^m, \vec{m}^T) = \vec{\Omega}\left(\vec{\Omega}\left((\vec{x}^\Omega + \vec{x}^d)/2, \vec{m}^D\right), \vec{m}^T\right)$$

Thus, the greedy T-forward-point and the greedy T-forward-function $\vec{G}(\vec{x}^\Omega, \vec{n}_f)$ can be written as:

$$\vec{G}(\vec{x}^\Omega, \vec{n}_f) = \vec{\Omega}\left(\vec{\Omega}\left((\vec{x}^\Omega + \vec{x}^d)/2, \vec{m}^D\right), \vec{m}^T\right) \tag{60}$$

The greedy T-forward solution can be formulated as:

$$\begin{aligned} &\vec{S}^G(\vec{x}^\Omega, \vec{n}_f) = \vec{G}(\vec{x}^K, \vec{n}_f) \\ &\vec{x}^k = \vec{G}(\vec{x}^{k-1}, \vec{n}_f), \quad for \quad k = 1, 2, \cdots, K \end{aligned} \tag{61}$$

11. Facet-forward Method for Linear Programming

The T-forward method gives ε-approximated solution for NLP, and gives exact optimal solution for LP, both in polynomial time if the feasible region is convex. However, the complexity for the T-forward method is little bit difficult to prove. In this section, we introduce facet-forward method, which simply applies the T-forward method on facet.

Given a feasible boundary $\vec{x}^\Omega \in \Omega$, we can find its constraint curve Ω^0. Let Π_Ω^0 be the facet on Ω and contains the point $\vec{x}^0$, that is:

$$\Pi_\Omega^0 = \Omega \bigcap \Omega^0 \tag{62}$$

The facet-forward method first finds the local optimal solution of the LP problem restrained in the facet Π_Ω^0, let it be $\vec{S}^0$.

Then we cut the feasible boundary Ω using the objective contour plane Π_f^0, which passes the point $\vec{S}^0$. This can be easily done by adding the function $f(\vec{S}^0) - f(\vec{x}) \le 0$ as an extra constraint to the original LP listed in Equation (3). Let us call the feasible boundary of the LP problem as the original feasible boundary. Once we add an objective contour plane $f(\vec{S}^0) - f(\vec{x}) \le 0$ to the original problem LP, the original feasible boundary Ω will be cut by the contour plane Π_f^0 into two parts.

DEFINITION: **NLPL on residual feasible region.** The NLPL problem of the original problem restrained in the residual feasible boundary. In other words, it is the following NLPL problem:

$$\text{Residual NLPL:} \begin{cases} \textit{Maximize}: & f(\vec{x}) = \vec{c} \bullet \vec{x} \\ \textit{Subject to}: & g_i(\vec{x}) \le 0 \qquad \textit{for } i = 1, 2, \cdots, N \\ \textit{Subject to}: & f(\vec{S}^0) - f(\vec{x}) \le 0 \end{cases} \tag{63}$$

DEFINITION: **Residual Feasible Boundary.** The feasible boundary of the residual NLPL problem. Actually, it is the set of the original feasible boundary with objective value greater than $f(\vec{S}^0)$.

Since $f(\vec{S}^0)$ is the largest value on the facet Π_f^0, then the constraint Ω^0 becomes completely redundant to the LP on residual feasible boundary. The constraint Ω^0 can be completely removed from the problem by substituting with the new constraint $f(\vec{S}^0) - f(\vec{x}) \le 0$. The new constraint doesn't have any impact when we search on another facet and try to find objective values greater than $f(\vec{S}^0)$. So, the problem becomes a LP with the number of constraint reduced by at least 1. There are N constraints in total, so, at most of N steps of such facet forward search, we will find the optimal solution.

One method for finding the local optimal solution within a facet is called **on-boundary T-forward** method. The on-boundary T-forward method simply applies the T-forward method in a facet, which means the dual point, the dual direction and the T-forward direction are all restrained in the facet.

Another method is called edge-forward method. Given a staring feasible point, we can apply the ray-boundary point function $\vec{\Omega}(\vec{x}^{\Omega}, \hat{\vec{z}})$ repeatedly to the starting point $\vec{x}^{\Omega}$, but with the moving direction vector $\hat{\vec{z}}$ constrained in the facet Π_{Ω}^0, sometimes may be along the edge of that facet. By repeating this process, we can eventually reach the local optimal solution of the specified facet.

12. Closed-form Solutions

Everything in the optimal solutions $\vec{S}^T(\vec{x}^{\Omega}, \vec{n}_f)$ as described in the previous sections are in closed-form format except the root value $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$. If $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ can also be expressed in closed-form format, then all these solutions $S^x(\vec{x}^{\Omega}, \vec{n}_f)$ give closed-form solution for NLPL or LP in the region $\Theta(\vec{x}^{\Omega})$. Actually, we can give closed-form solution for $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ for some special cases. In this section, we give closed form solution for LP, QCQP, and NLP with Homogeneous (NLPH) constraints.

12.1. Closed-Form Solution for Linear Programming

If the problem is LP, then all constraints are linear functions. The constraint $g_i(\vec{x})$ can be expressed as:

$$g_i(\vec{x}) = \vec{b}^i \bullet \vec{x} + c^i \tag{64}$$

Then $g_i(\vec{x}^{\Omega} + \lambda^i \hat{\vec{z}}) = 0$ becomes:

$$g_i(\vec{x}) = \vec{b}^i \bullet (\vec{x}^{\Omega} + \lambda^i \hat{\vec{z}}) + c^i = 0 \tag{65}$$

From which we can easily solve λ^i as:

$$\lambda^i = \frac{-c^i - \vec{b}^i \bullet \vec{x}^\Omega}{\vec{b}^i \bullet \hat{\vec{z}}} \tag{66}$$

Based on the definition for $\breve{g}_i(\vec{x}^\Omega, \hat{\vec{z}})$, we have:

$$\breve{g}_i(\vec{x}^\Omega, \hat{\vec{z}}) = \begin{cases} 0 & if \quad \vec{b}^i \bullet \hat{\vec{z}} = 0 \\ \lambda^i & else\ if \quad \lambda^i > 0 \\ +\infty & else \end{cases} \tag{67}$$

It is in closed-form format, and thus Equation (58) gives closed-form solution for LP. Also, the feasible boundary for LP must be convex and have just one region. Then, Equation (58) gives global optimal solution for LP.

12.2. Closed-Form Solution for Quadratically Constrained Quadratic Programming

In the case of QCQP, all the constraints are quadratic. The constraint $g_i(\vec{x})$ can be expressed as:

$$g_i(\vec{x}) = \vec{x} A^i \vec{x}^T + B^i \vec{x}^T + C^i \tag{68}$$

Then $g_i(\vec{x}^0 + \lambda \hat{\vec{z}}) = 0$ becomes:

$$g_i(\vec{x}^0 + \lambda \hat{\vec{z}}) = a\lambda^2 + b\lambda + c = 0 \tag{69}$$

where

$$a = \sum_{k=1}^{M}\sum_{j=1}^{M} A^i_{jk} \hat{z}_j \hat{z}_k \ ; \ b = \sum_{k=1}^{M}\sum_{j=1}^{M}\left(A^i_{jk} \hat{z}_j x^0_k + A^i_{kj} \hat{z}_k x^0_j \right) + \sum_{j=1}^{M} B^i_j \hat{z}_j \ ; \ c = \sum_{k=1}^{M}\sum_{j=1}^{M} A^i_{jk} x^\Theta_k x^\Theta_{jk} + C^i \tag{70}$$

λ can be solved through the well known closed-form solution for quadratic equation:

$$\lambda^i_1 = \frac{b + \sqrt{\Delta}}{2a}, \quad \lambda^i_2 = \frac{b - \sqrt{\Delta}}{2a}, \quad \Delta = b^2 - 4ac \tag{71}$$

By definition, $\breve{g}_i(\vec{x}^\Theta, \hat{\vec{z}})$ can be formulated as:

$$\breve{g}_i(\vec{x}^\Theta, \hat{\vec{z}}) = \begin{cases} +\infty & if \quad \Delta < 0 \ or \ (\lambda^i_1 \le 0 \ and \ \lambda^i_2 \le 0) \\ \lambda^i_1 & else\ if \quad \lambda^i_1 > 0 \ and \ (\lambda^i_1 \le \lambda^i_2 \ or \ \lambda^i_2 \le 0) \\ \lambda^i_2 & else\ if \quad \lambda^i_2 > 0 \end{cases} \tag{72}$$

Then Equation (58) gives closed-form solution for QCQP.

12.3. Closed-Form Solution for NLP with Homogeneous Constraints

DEFINITION: **Homogeneous function.** A function $f(\vec{x})$ is called homogeneous with degree γ if:

$$f(\mu\vec{x}) = \mu^\gamma f(\vec{x}), \quad \forall \mu \in R^1 \tag{73}$$

Here γ is a positive integer number.

DEFINITION: **NLP with homogeneous constraint (NLPH).** A nonlinear programming in which all the constraints are homogeneous.

For an NLPH, the i[th] constraint can be expressed as:

$$g_i(\vec{x}) \le 1 \quad \text{for } i = 1, 2, \cdots, N \tag{74}$$

And $g_i(\vec{x})$ has the following property:

$$g_i(\mu\vec{x}) = \mu^{\gamma_i} g_i(\vec{x}) \quad \text{for } i = 1, 2, \cdots, N \tag{75}$$

For an NLPH, the origin point is always a feasible point. The $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ is the root of the equation $g_i(\vec{x}^{\Theta} + \lambda\hat{\vec{z}}) = 1$. We can choose $\vec{x}^{\Theta} = \vec{0}$. Then $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ is the root of $g_i(\lambda\hat{\vec{z}}) = 1$. With the homogeneous property, λ can be solved as:

$$\lambda^i = \left(g_i(\hat{\vec{z}})\right)^{-\frac{1}{\gamma_i}} \tag{76}$$

By definition, $\breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$ can be formulated as:

$$\breve{g}_i(\vec{x}^{0}, \hat{\vec{z}}) = \begin{cases} \left(g_i(\hat{\vec{z}})\right)^{-\frac{1}{\gamma_i}} & if \quad g_i(\hat{\vec{z}}) > 0 \\ +\infty & else \end{cases} \tag{77}$$

Therefore, Equation (58) gives the closed-form solution for NLPH.

13. Find the Global Optimal Solution for NLPL

Given a feasible point $\vec{x}^{\Theta} \in \Theta$, we can find the feasible boundary point $\vec{x}^{\Omega} \in \Omega$ by applying $\vec{x}^{\Omega} = \vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{x}}^{\Theta})$. Then, Equation (58) and (61) give the optimal solution for NLPL in the shine region $\Theta(\vec{x}^{\Theta})$. If $\Theta(\vec{x}^{\Theta})$ happened to be the whole feasible region Θ, then the solution $\vec{S}^T(\vec{x}^{\Omega}, \vec{n}_f)$ given by Equation (58) is the global optimal solution for NLPL.

For example, when Θ is convex, we should have $\Theta(\vec{x}^{\Theta}) = \Theta$. Then $\vec{S}^T(\vec{x}^{\Omega}, \vec{n}_f)$ is the global optimal solution for NLPL. Particularly, the feasible region of an LP is always convex, we can always give global optimal solution through the closed-form solution $\vec{S}^T(\vec{x}^{\Omega}, \vec{n}_f)$ for LP.

However, if $\Theta(\vec{x}^{\Theta})$ cannot cover all the feasible region Θ, $\vec{S}^T(\vec{x}^{\Omega}, \vec{n}_f)$ might be a local optimal solution. In that case, we need to find a sun-shine-set that can cover all points in the feasible region through direct path connection. The infeasible boundary function $\vec{\overline{\Omega}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$ given in Equation (42) can be useful in finding the sun-shine set. Suppose $\vec{x}^{\overline{\Theta}} \in \overline{\Theta}$, we can apply $\vec{\overline{\Omega}}(\vec{x}^{\overline{\Theta}}, \hat{\vec{z}})$ with some randomly generated directions $\hat{\vec{z}}$ until we find a feasible point that is not in the shine regions that have been searched.

14. Algorithm for T-Forward Method

Based on the previous sections, the algorithm for T-Forward method can be summarized as follows.

Main Procedure: T-forward-method

Input: ε: requested precision; N: number of the constraints; M: Number of variables; $\vec{c}$: gradient of the objective function; $\vec{x}^{\Theta}$: an feasible interior point; and all the coefficients in each of the constraint $g_i(\vec{x})$.

Output: A feasible boundary point $\vec{x}^{\Omega}$ that maximizes the objective function.

1. Formulate the original NLP in the following NLPL as shown in Equation (3):

$$\text{NLPL:} \begin{cases} Maximize: & f(\vec{x}) = \vec{c} \bullet \vec{x} \quad a\ linear\ function \\ Subject\ to: & g_i(\vec{x}) \le 0 \quad for\ i = 1, 2, \cdots, N \end{cases} \quad \text{for} \quad \vec{x} \in R^M$$

2. Initialization

2.1. Set $\vec{n}_f \leftarrow \vec{c} / |\vec{c}|$.

2.2. Set $\hat{\vec{z}} \leftarrow \vec{n}_f$.

2.3. Calculate ray-boundary point $\vec{x}^{\Omega}$:

$\vec{x}^{\Omega} \leftarrow$ get-ray-boundary-point($\vec{x}^{\Theta}, \hat{\vec{z}}$)

2.4. Set $\vec{x}^{0} \leftarrow \vec{x}^{\Omega} / 2$.

3. While $\left|\vec{x}^{\Omega} - \vec{x}^{0}\right| \Big/ \left|\vec{x}^{\Omega}\right| > \varepsilon$ do loop

3.1. Set $\vec{x}^{0} \leftarrow \vec{x}^{\Omega}$.

3.2. Set $\vec{x}^{\Omega} \leftarrow$ get-T-forward-point($\vec{x}^{\Omega}, \hat{\vec{z}}$).

3.3. End While-do loop.

4. Return $\vec{x}^{\Omega}$ as the optimal solution and $f\left(\vec{x}^{\Omega}\right)$ as the optimal objective value.

End of **T-forward-method**

Function get-T-forward-point ($\vec{x}^{\Omega}, \hat{\vec{z}}$)

Input: A boundary point $\vec{x}^{\Omega}$, and all the input information for the NLPL.

Output: A boundary point $\vec{x}^{T}$, which is the T-forward point with $\vec{x}^{\Omega}$ as starting point.

1. Set $\hat{\vec{z}} \leftarrow \vec{x}^{\Omega} \Big/ \left|\vec{x}^{\Omega}\right|$;

2. Calculate $\Phi\left(\{t_i\}\right)$ for the point $\vec{x}^{\Omega}$:

$\Phi\left(\{t_i\}\right) \leftarrow$ **get-partition-function ($\vec{x}^{\Omega}, \hat{\vec{z}}$);**

3. Calculate gradient vector $\vec{n}_{\Omega}^{0}$:

$\vec{n}_{\Omega}^{0} \leftarrow$ **get-gradient-vector(** $\vec{x}^{\Omega}, \hat{\vec{z}}$ **)**

4. Calculate dual point $\vec{x}^{d}$ through Equation (51):

$\vec{x}^{d} \leftarrow$ **get-dual-point (** $\vec{x}^{\Omega}, \hat{\vec{z}}$ **)**

5. Calculate gradient vector $\vec{n}_{\Omega}^{d}$:

$\vec{n}_{\Omega}^{d} \leftarrow$ **get-gradient-vector (** $\vec{x}^{d}, \hat{\vec{z}}$ **)**

6. Calculate T-forward direction $\vec{m}^{T}$ through Equation (53):

$$\vec{n} = (\vec{n}_{\Omega}^{0} + \vec{n}_{\Omega}^{d}) sign\left((\vec{n}_{\Omega}^{0} + \vec{n}_{\Omega}^{d}) \bullet \vec{n}_{f}\right), \ \vec{m}^{T} = \vec{n} \Big/ \left|\vec{n}\right|$$

7. Calculate weighted-interior point $\vec{x}^{W}$ through Equation (54):

$$\mu = \frac{\cos(\theta + \alpha)}{2\cos\alpha\cos\theta}; \qquad \alpha = \frac{1}{2}\left(\pi - \arccos\left(\vec{n}_{\Omega}^{0} \bullet \vec{n}_{\Omega}^{d}\right)\right);$$

$$\vec{x}^{W} = (1-\mu)\vec{x}^{\Omega} + \mu\vec{x}^{d} = \vec{x}^{\Omega} + \mu(\vec{x}^{d} - \vec{x}^{\Omega})$$

8. Calculate T-forward point $\vec{x}^{T}$:

$\vec{x}^{d} \leftarrow$ **get-ray-boundary-point (** $\vec{x}^{W}, \vec{m}^{T}$ **)**

End of **Function get-T-forward -point**

Function get-ray-boundary-point ($\vec{x}^{\Theta}, \hat{\vec{z}}$)

Input: A feasible interior or boundary point $\vec{x}^{\Theta}$, a unit vector $\hat{\vec{z}}$, and all the input information for the NLPL.

Output: A boundary point $\vec{x}^{\Omega}$.

1. Calculate $\Phi\left(\{t_i\}\right)$ for the pair ($\vec{x}^{\Theta}, \hat{\vec{z}}$):

$\Phi\left(\{t_i\}\right) \leftarrow$ **get-partition-function (** $\vec{x}^{\Theta}, \hat{\vec{z}}$ **);**

2. Calculate ray-boundary point $\vec{x}^{\Omega}$ through Equation (41):

$$\vec{x}^{\Omega} \leftarrow \quad \vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \vec{x}^{\Theta} + \hat{\vec{z}} \sum_{i=1}^{N} t_i \Phi\left(\left\{(t_i)^{-1}\right\}\right)$$

End of **Function get-ray-boundary-point**

Function get-gradient-vector ($\vec{x}^{\Omega}, \hat{\vec{z}}$)

Input: A boundary point $\vec{x}^{\Omega}$, a unit vector $\hat{\vec{z}}$, and all the input information for NLPL.

Output: $\vec{n}^{\Omega}$, which is the gradient vector of the feasible boundary at $\vec{x}^{\Omega}$.

1. Calculate $\Phi(\{t_i\})$ for the pair ($\vec{x}^{\Omega}$, $\hat{\vec{z}}$)::

   $\Phi(\{t_i\}) \leftarrow$ **get-partition-function** ($\vec{x}^{\Omega}$, $\hat{\vec{z}}$);

2. Calculate gradient vector $\vec{\Lambda}^i$ for each constraint at the point $\vec{x}^{\Omega}$ through Equation (48):

   $\vec{\Lambda}^i = \vec{p}/|\vec{p}|, \quad \vec{p}(\vec{x}^{\Theta}, \hat{\vec{z}}) = \nabla g_i(\vec{x})\big|_{\vec{x}=\vec{\Omega}(\vec{x}^{\Omega}, \hat{\vec{z}})}$

3. Calculate gradient vector $\vec{n}_{\Omega}^0$ through Equation (43):

$$\vec{n}_{\Omega}^0 \leftarrow \vec{\Lambda}(\vec{x}^{\Omega}, \hat{\vec{z}}) = \sum_{i=1}^{N} \vec{\Lambda}^i \Phi\left(\left\{(t_i)^{-1}\right\}\right)$$

End of **Function get-gradient-vector**

Function get-dual-point ($\vec{x}^{\Omega}, \hat{\vec{z}}$)

Input: A boundary point $\vec{x}^{\Omega}$, and all the input information for the NLPL.

Output: A boundary point $\vec{x}^{d}$, which is the dual point of $\vec{x}^{\Omega}$.

1. Calculate gradient vector $\vec{n}_{\Omega}^0$:

   $\vec{n}_{\Omega}^0 \leftarrow$ **get-gradient-vector ($\vec{x}^{\Omega}, \hat{\vec{z}}$)**

2. Calculate dual direction $\vec{m}^{D}$ through Equation (50):

$$\vec{m}^{D} = \begin{cases} -\vec{n}_{\Omega}^0 & if \quad |\vec{v}| = 0 \\ -\vec{v}/|\vec{v}|, \quad \vec{v} = \vec{n}_{\Omega}^0 - \vec{n}_f\left(\vec{n}_f \bullet \vec{n}_{\Omega}^0\right) & if \quad |\vec{v}| \neq 0 \end{cases}$$

3. Calculate dual point $\vec{x}^{d}$:

   $\vec{x}^{d} \leftarrow$ get-ray-boundary-point ($\vec{x}^{\Omega}, \vec{m}^{D}$)

End of **Function get-dual-point**

Function get-partition-function ($\vec{x}^{\Theta}$, $\hat{\vec{z}}$)

Input: A feasible point $\vec{x}^{\Theta}$, a unit vector $\hat{\vec{z}}$, and all the input information for the NLPL.

Output: $\Phi(\{t_i\})$, which is the partition function at the feasible boundary point $\vec{x}^{\Omega}$.

1. Calculate $t_i = \breve{g}_i(\vec{x}^{\Theta}, \hat{\vec{z}})$. For linear constraints, t_i can be calculated through Equation (66) and (67). For quadratic constraints, t_i can be calculated through Equation (70), (71), and (72).
2. Calculate $\Phi(\{t_i\})$ through Equation (45), (46), or (47).

End of **Function get-partition-function**

15. Solving Linear Programming Examples

A standard LP can be expressed as:

$$\text{LP:} \begin{cases} Maximize & f(\vec{x}) = \vec{c} \bullet \vec{x} & for & \vec{x} \in R^M \\ Subject\ to: & g_i(\vec{x}) = \vec{a}^i \bullet \vec{x} + b^i \le 0 & for & i = 1, 2, \cdots, N \end{cases} \tag{78}$$

DFINITION: **Negative Constraint**. A constraint in the format $g_i(\vec{x}) \le 0$ is called a negative constraint if $g_i(\vec{0}) \le 0$, and a **positive constraint** if otherwise.

A negative constraint is a valid constraint at the origin point. If all constraints are negative constraints, then the origin point will be a feasible point. If the origin point *O* is a feasible point, then, all constraints must be negative constraints. Suppose $\vec{x}^{\Theta} \in \Theta$ is a feasible point we have found, for example, we can apply simplex phase-I to find a feasible point. We can transform all constraints into negative constraints through the following transform:

$$\vec{x} = \vec{y} + \vec{x}^{\Theta} \tag{79}$$

Then the original problem listed in Equation (78) becomes:

$$\text{LP after variable transform: } \begin{cases} Maximize & f(\vec{y}) = \vec{c} \bullet \vec{y} + \vec{c} \bullet \vec{x}^{\Theta} \quad for \quad \vec{x} \in R^M \\ Subject\ to: & g_i(\vec{y}) = \vec{a}^i \bullet \vec{y} + d^i \le 0 \quad for \quad i = 1, 2, \cdots, N \end{cases} \tag{80}$$

$$\text{where} \quad d^i = \vec{a}^i \bullet \vec{x}^{\Theta} + b^i \le 0 \ \text{ for } i = 1, 2, \cdots, N \tag{81}$$

Without losing generality, we can assume that the LP problem has been transformed into an LP with all negative constraints. In other words, we assume the LP listed in Equation (78) has the following property:

$$b^i \le 0 \ \text{ for } i = 1, 2, \cdots, N \tag{82}$$

15.1. Example 1: An LP with 2 Variables and 100 Constraints

Our first example is an LP with 2 variables and 100 constraints. The input data are randomly generated numbers in the range [-10, 10] for a_k^i, and in the range [-20,-10] for b^i. Table 1 lists the input data in table format.

We first apply the closed-form solution, as listed Equation (41), to draw the feasible boundary Ω, as shown in Figure 6. It can be observed that the feasible boundary contains only 7 straight lines. Then we know that there are about 93 constraints are redundant for this problem! However, there is no need for us to figure out which constraint is redundant and which is not. The closed-form boundary function can do the nice job for us and can filter out all the redundant constraints automatically.

Table 2 lists step-by-step results for T-forward method to solve LP Example-1 with objective $f(x) = x_1$.

Table 1. Input data for LP Example-1 in table format

	x1	x2	g(0)		x1	x2	g(0)		x1	x2	g(0)		x1	x2	g(0)
1	3	7	-5	26	10	5	-13	51	-1	-6	-2	76	-10	5	-3
2	10	9	-19	27	2	-7	-10	52	5	4	-3	77	-5	-5	-2
3	-5	-4	-4	28	-6	5	-6	53	6	-8	-2	78	4	-5	-4
4	-3	8	-3	29	2	10	-1	54	8	-1	-9	79	-6	8	-17
5	4	-5	-8	30	4	1	-4	55	5	-5	-12	80	-7	-3	-16
6	0	-9	-16	31	-8	-9	-19	56	3	-2	-15	81	0	6	-17
7	4	8	-1	32	6	-10	-13	57	-6	3	-15	82	-6	-4	-5
8	-2	-1	-17	33	-10	-7	-4	58	6	-1	-3	83	10	6	-10
9	-7	1	-15	34	-10	5	-5	59	9	7	-16	84	7	-10	-10
10	-10	6	-2	35	4	-8	-8	60	-2	7	-4	85	6	4	-17
11	5	-3	-8	36	9	-5	-7	61	9	-9	-5	86	4	-5	-7
12	6	-8	-1	37	-4	-9	-3	62	-5	-6	-5	87	6	5	-3
13	3	3	-16	38	3	-8	-18	63	4	4	-9	88	-3	1	-14
14	7	-7	-9	39	10	9	-8	64	9	8	-4	89	5	-2	-13
15	6	-6	-3	40	-4	-3	-1	65	10	0	-12	90	9	-1	-17
16	7	-7	-1	41	-6	8	-6	66	5	9	-11	91	-9	6	-5
17	-7	5	-14	42	1	4	-14	67	-5	7	-18	92	3	1	-15
18	8	6	-15	43	-3	-10	-11	68	-3	1	-17	93	-9	2	-5
19	6	0	-13	44	4	-4	-5	69	0	0	-6	94	10	9	-19
20	7	-9	-11	45	10	-8	-17	70	-5	8	-8	95	-2	3	-3
21	-6	-7	-12	46	4	-8	-4	71	-1	10	-5	96	-5	-3	-5
22	-8	-2	-17	47	-8	4	-2	72	1	-6	-20	97	-2	-4	-4
23	-8	1	-15	48	9	10	-15	73	-2	4	-14	98	0	-9	-12
24	-9	6	-20	49	-3	-8	-17	74	4	5	-17	99	5	7	-8
25	7	2	-12	50	1	-6	-17	75	7	-8	-17	100	-3	6	-1

Table 2. T-forward method takes 2 forward steps to give exact optimal solution for LP Example-1 (objective $f(\vec{x}) = x_1$)

Objective function:	$f(\vec{x}) = x_1$					1	0	Gradient Vector		Forward Direction	
Loop	Point Type	f(x)	Improve	Error	Move Size	x1	x2	n1	n2	mv1	mv2
1	Start Point	0.026316	0.02632	0.00000	0.00000	0.02632	-0.10526	0.60000	-0.80000	0.00000	0.00000
1	Dual point	0.026316	0.00000	1.93382	0.20000	0.02632	0.09474	0.19612	0.98058	0.00000	1.00000
1	Mid-Interior	0.026316	0.00000	1.93382	0.11014	0.02632	0.00488	0.00000	0.00000	0.00000	1.00000
1	T-Forward	0.173481	0.14717	1.05146	0.20556	0.17348	0.03826	0.44721	0.89443	0.97523	0.22121
2	Dual point	0.173481	0.00000	0.04234	0.00764	0.17348	0.03062	0.70711	-0.70711	0.00000	-1.00000
2	Mid-Interior	0.173481	0.00000	0.04234	0.00337	0.17348	0.03489	0.00000	0.00000	0.00000	-1.00000
2	T-Forward	0.178571	0.00509	0.01967	0.00569	0.17857	0.03571	0.98709	0.16018	0.98709	0.16018
3	Dual point	0.178571	0.00000	0.00000	0.00000	0.17857	0.03571	0.98709	0.16018	0.00000	-1.00000
3	Optimal	0.178571	0.00000	0.01446	0.00000	0.17857	0.03571	0.98709	0.16018	0.00000	0.00000

Table 3 lists step-by-step results for T-forward method to solve LP Example-1 with objective $f(x) = -x_1$.

Figure 6. also shows the first few T-forward steps for the results listed in Table 2 and Table 3. T-forward method takes only two forward steps to solve LP Example-1 and give exact optimal solution.

Table 3. T-forward method takes 2 forward steps to give exact optimal solution for LP Example-1. (objective $f(x) = -x_1$)

Objective function:	$f(x) = -x_1$					-1	0	Gradient Vector		Forward Direction	
Loop	Point Type	f(x)	Improve	Error	Move Size	x1	x2	n1	n2	mv1	mv2
1	Start Point	0.055556	0.05556	0.00000	0.00000	-0.05556	0.11111	0.19612	0.98058	0.00000	0.00000
1	Dual point	0.055556	0.00000	1.84776	0.27778	-0.05556	-0.16667	0.60000	-0.80000	0.00000	-1.00000
1	Mid-Interior	0.055556	0.00000	1.84776	0.09476	-0.05556	0.01635	0.00000	0.00000	0.00000	-1.00000
1	T-Forward	0.211398	0.15584	0.53387	0.20301	-0.21140	-0.01900	-0.85749	0.51450	-0.97523	-0.22121
2	Dual point	0.211398	0.00000	0.14907	0.03247	-0.21140	-0.05147	-0.80000	-0.60000	0.00000	-1.00000
2	Mid-Interior	0.211398	0.00000	0.14907	0.01748	-0.21140	-0.03648	0.00000	0.00000	0.00000	-1.00000
2	T-Forward	0.222222	0.01082	0.07551	0.02104	-0.22222	-0.03704	-0.80000	-0.60000	-0.99867	-0.05152
3	Dual point	0.222222	0.00000	0.00000	0.00000	-0.22222	-0.03704	-0.80000	-0.60000	0.00000	1.00000
3	Optimal	0.222222	0.00000	0.02496	0.00000	-0.22222	-0.03704	-0.80000	-0.60000	0.00000	0.00000

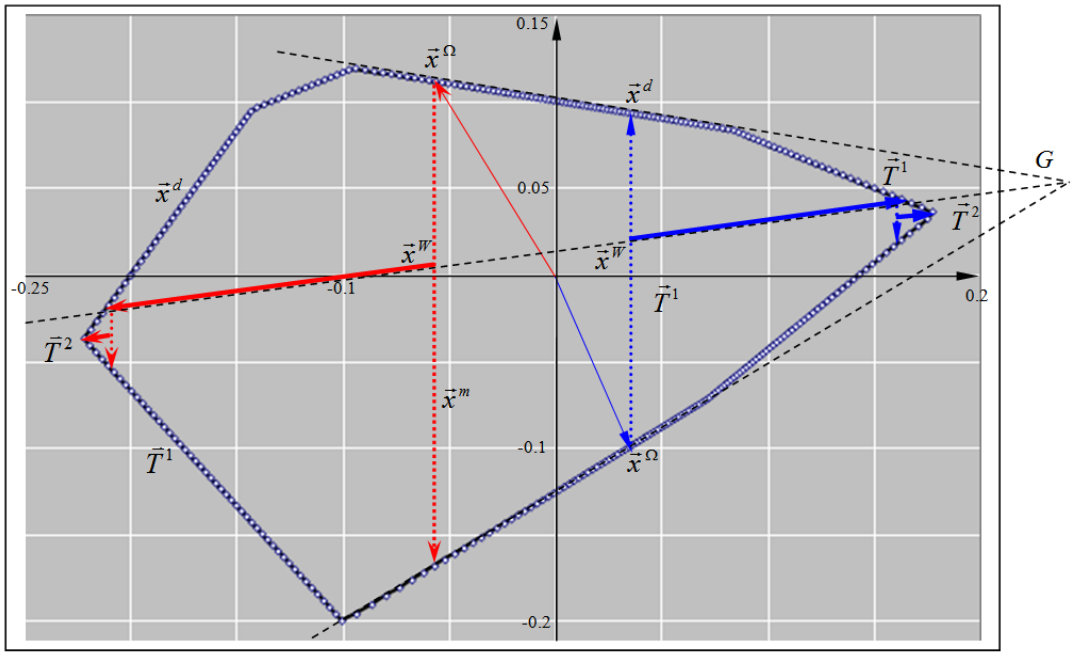

Figure 6. Finding optimal solutions for LP Example-1 through T-forward method. The blue lines on the right side illustrate the T-forward path for maximizing $f(x) = x_1$ (see Table 2). The red lines on the left side illustrate the T-forward path for maximizing $f(x) = -x_1$ (see Table 3). The vertical dashed lines represent the dual directions. T-forward method takes 2 forward steps to find exact optimal solution for both objective functions

15.2. Example 2: An LP with 2 Variables and 11 Constraints

Our second example is an LP problem that is used to describe the interior-point method at the website: http://en.wikipedia.org/wiki/Karmarkar's_algorithm. This problem can be formulated as:

$$\text{LP Example-2: } \begin{cases} Maximize: & x_1 + x_2 \\ Subject\ to: & 2px_1 + x_2 - p^2 - 1 \le 0, \quad p = (i-1)/10 \quad for \quad i = 1, 2, \cdots, N \end{cases} \tag{83}$$

Table 4 lists the input data in table format for LP Example-2. Figure 7. shows the feasible boundary of this problem. By applying Equation (41), we can draw the feasible boundary Ω as shown in Figure 8. It can be observed that the boundary curve drawn through our closed-form Equation (41) is exactly the same as the feasible boundary as shown in Figure 7. This gives another validation for our closed-form solution.

Table 5 lists step-by-step results for T-forward method to solve LP Example-2 with objective $f(\vec{x}) = x_1 + x_2$.

T-forward method takes 2 forward steps to give exact optimal solution for this problem as shown in Figure 8., while interior method takes many steps to reach an ε-approximated solution as shown in Figure 7.

Table 4. Input data for LP Example-2 with objective $f(x) = x_1 + x_2$

Variable	x1	x2	g(0)	Degree
Objective	1	1		1
Constraint 1	0.00	1.00	-1.00	1
Constraint 2	0.20	1.00	-1.01	1
Constraint 3	0.40	1.00	-1.04	1
Constraint 4	0.60	1.00	-1.09	1
Constraint 5	0.80	1.00	-1.16	1
Constraint 6	1.00	1.00	-1.25	1
Constraint 7	1.20	1.00	-1.36	1
Constraint 8	1.40	1.00	-1.49	1
Constraint 9	1.60	1.00	-1.64	1
Constraint 10	1.80	1.00	-1.81	1
Constraint 11	2.00	1.00	-2.00	1

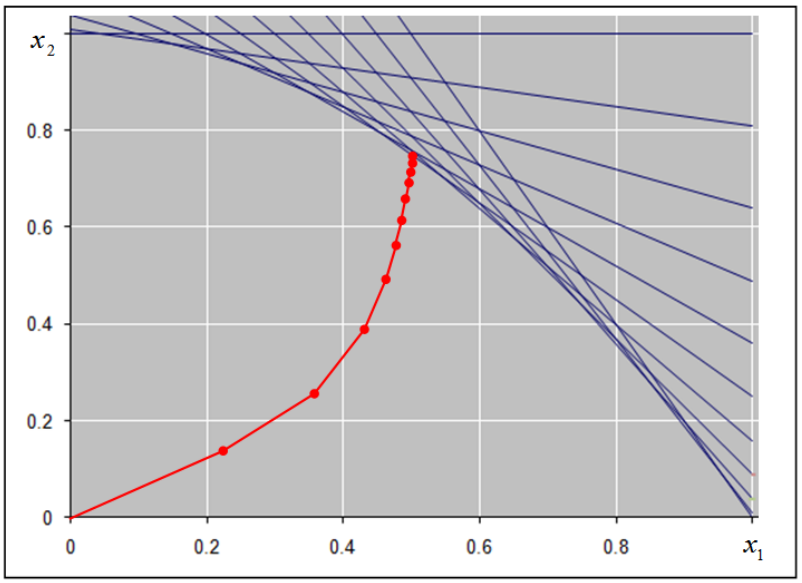

Figure 7. The feasible boundary curve for LP Example-2. The red line represents the path searched by the interior method. This figure is used to describe the interior point method at the website http://en.wikipedia.org/wiki/Karmarkar's_algorithm

Table 5. T-forward method gives exact optimal solution in 2 forward steps for LP Example-2 (objective $f(x) = x_1 + x_2$)

Objective function:	$f(x) = x_1 + x_2$				1	1	Gradient Vector		Forward Direction	
Loop	Point Type	f(x)	Improve	Error	Move Size	x1	x2	n1	n2	mv1	mv2
1	Start Point	1.089216	1.08922	0.00000	0.00000	0.09902	0.99020	0.19602	0.98060	0.00000	0.00000
1	Dual point	1.089216	0.00000	1.18246	1.13414	0.90098	0.18824	0.87416	0.48564	0.70711	-0.70711
1	Mid-Interior	1.089216	0.00000	1.18246	0.37561	0.36462	0.72460	0.00000	0.00000	0.70711	-0.70711
1	T-Forward	1.246168	0.15695	0.11060	0.37508	0.43084	0.81533	0.62470	0.78087	0.58954	0.80774
2	Dual point	1.246168	0.00000	0.21254	0.19562	0.56916	0.67701	0.76822	0.64018	0.70711	-0.70711
2	Mid-Interior	1.246168	0.00000	0.11060	0.08812	0.49315	0.75301	0.00000	0.00000	0.70711	-0.70711
2	T-Forward	1.250000	0.00383	0.00000	0.08814	0.49505	0.75495	0.70711	0.70711	0.70000	0.71414
2	Optimal	1.250000	0.00383	0.00154	0.00000	0.49505	0.75495	0.70711	0.70711	0.00000	0.00000

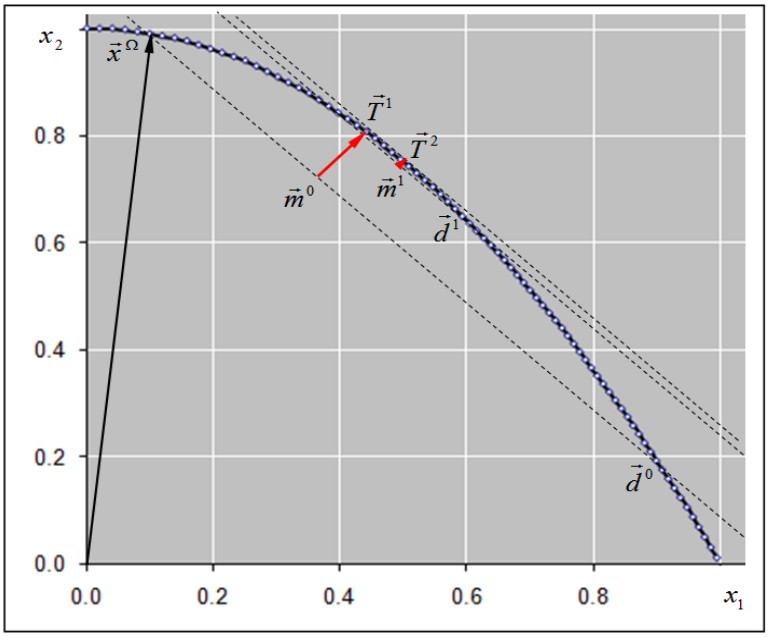

Figure 8. Starting from a feasible boundary point $\vec{x}^{\Omega}$, T-forward method takes two steps to find the exact optimal solution for LP Example-2. The red arrow line represents the T-forward move. The dashed lines represent the dual direction

16. Complexity Analysis

Let us analyze the complexity for the T-forward-point method and the Facet-forward method introduced in previous sections.

The T-forward-point method is simply to call the T-forward-function $\vec{T}(\vec{x}^{\Omega}, \vec{n}_f)$ repeatedly. Each T-forward-function $\vec{T}(\vec{x}^{\Omega}, \vec{n}_f)$, as shown in Equation (56), includes two calls of the boundary function $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ and two calls of the gradient function $\vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}})$. Each of these functions needs to calculate the roots $\breve{g}_i(\vec{x})$ for $i = 1,2,\cdots,N$. The complexity for calculating one $\breve{g}_i(\vec{x})$ through Equation (67) is $O(L)$, where L represents the maximum arithmetic calculations for calculating one $\breve{g}_i(\vec{x})$ or the length of the input for one constraint. For LP problems, $L = M$. The complexity for calculating all the roots $\breve{g}_i(\vec{x})$ for $i = 1,2,\cdots,N$ is $O(NL)$. Once $\breve{g}_i(\vec{x})$ is calculated, the boundary function $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ and the gradient function $\vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}})$ can be calculated in $O(N)$. Thus, the T-forward-function $\vec{T}(\vec{x}^{\Omega}, \vec{n}_f)$ can be calculated in $O(NL)$.

Now let us estimate how many T-forward steps are needed to find the optimal solution through T-forward method. Here we only consider convex NLP, which means the feasible region Ω is convex.

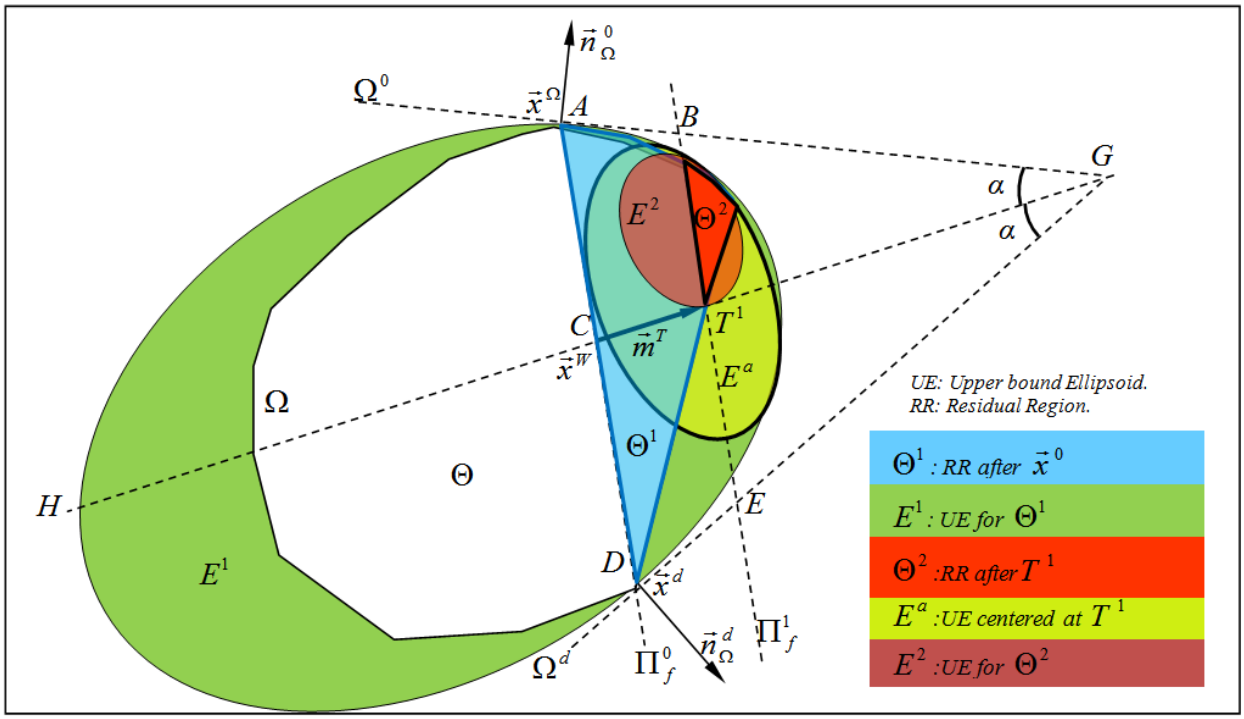

Figure 9. Initially the residual feasible region is Θ, the residual feasible region is Θ^1 with E^1 as their upper bound ellipsoid. After one T-forward move, the residual feasible region becomes Θ^2 with E^2 as their upper bound ellipsoid. E^a is an ellipsoid contained in E^1 and centered at T^1 and an upper bound for E^2. The curve $\overline{GH}$ bisects the region AGDCA. The volume of E^a is less than half of E^1, and the volume of E^2 is less than half of E^a. Then, after one T-forward step, the residual feasible region is reduced at least be a factor of 4

As illustrated in Figure 9. , at the initial feasible point $\vec{x}^\Omega$, we use the objective contour plane Π_f^0 to cut the feasible region Ω. The residual feasible region must be in the region bounded by the two plane curves Ω^0 and Ω^d, where Ω^0 is the tangent plane curve of Ω passing the point $\vec{x}^\Omega$, and Ω^d is the tangent plane curve of Ω passing through the dual point $\vec{x}^d$. Point G represents the common edge of the two plane curves Ω^0 and Ω^d. The line $\overline{GH}$ represents the bisector of the two curves Ω^0 and Ω^d. Before we start the T-Forward step, the upper bound ellipsoid of the residual feasible region is in the region GAD. Here we only consider the ellipsoid that is cut by the contour plane Π_f^0. After the first T-forward step, the feasible region is cut at the T-forward point $\vec{T}^1$. Since $\vec{T}^1$ is designed to be along the direction $\overline{GH}$, and it is a boundary point of the feasible region. Then the residual region after $\vec{T}^1$ becomes the region Θ^2, with $\vec{T}^1$ as one of its boundary point. Thus Θ^2 must be either in the region GBT^1 or GET^1, either one is half of the region GBE. We draw another ellipsoid E^a, which with $\vec{T}^1$ as its center and upper bounded by E^1, and with E^2 to be contained in E^a. We should have:

$$Vol(E^2) \le \frac{1}{2} Vol(E^a), \quad Vol(E^a) \le \frac{1}{2} Vol(E^1), \quad Vol(E^2) \le \frac{1}{4} Vol(E^2) \tag{84}$$

Therefore, each T-forward step moves the cutting plane Π_f^k close to the target further and reduces the volume of the residual feasible region at least by a factor of 4. After K times of T-forward steps, the volume of the residual feasible region will be reduced by 4^K. The relative difference between the T-forward solution $\vec{S}^T(\vec{x}^\Omega, \hat{\vec{z}})$ and the optimal solution $\vec{S}$

will be reduced to the order of 4^{-K}. In other words, if the requested precision is ε, we need to call T-forward function $\ln(1/\varepsilon)$ times.

Ellipsoid method is proven to be polynomial through similar method as we used above [10, 11].

In summary, the running time for T-forward method to give an ε-approximated solution for NLP is $O(NL\ln(1/\varepsilon))$.

If the problem is LP, the T-forward method can give exact optimal solution with complexity $O(NL\ln(N))$.

The facet-forward method simply calls $\vec{\Omega}(\vec{x}^{\Theta}, \hat{\vec{z}})$ and $\vec{\Lambda}(\vec{x}^{\Theta}, \hat{\vec{z}})$ at most N times to give local optimal solution in a facet. It takes $O(N^2 L)$ time to get a solution for one facet and have the number of constraint reduced at least by 1. Then, it needs to run at most N times to reduce all constraints and give final optimal solution for LP problems. Therefore, the running time for facet-forward method is $O(N^3 L)$.

Table 6 compares the methods proposed in this paper with the current best known existing LP algorithms, including simplex method, the ellipsoid method, and the interior point method.

Table 6. Comparison of the T-forward method with simplex, ellipsoid, and the interior point method

Method	Applicable	Search Path Type	LP Solution Type	Complexity
T-forward	LP and NLP	Mid interior	Exact	$O(NL\ln L)$
Greedy T-forward	LP and NLP	Mid interior	Approximate	$O(NL\ln L)$
Facet-forward	LP	Facet-by-facet	Exact	$O(N^3 L)$
Interior point	LP and NLP	Interior	Approximate	$O(N^{3.5} L^2 \ln L \ln\ln L)$
Ellipsoid	LP and NLP	Interior	Approximate	$O(N^6 L^2 \ln L \ln\ln L)$
Simplex	LP	Vertex-by-Vertex	Exact	Exponential

Suppose we run a computer with 1Ghz processor, Table 7 lists the estimated running time in seconds for various methods and various N and L. For simplicity, here we assume $N = L$.

Table 6. Estimated running time (in seconds) on a computer with 1 GHz processor

N, L	10	100	1,000	10,000	100,000	1,000,000
T-forward	2E-07	5E-05	7E-03	9E-01	1E+02	1E+04
Facet-forward	1E-05	1E-01	1E+03	1E+07	1E+11	1E+15
Interior point	6E-04	7E+02	4E+08	2E+14	9E+19	4E+25
Ellipsoid	2E-01	7E+07	1E+16	2E+24	3E+32	4E+40

For example, if we want to solve an LP with $N = L = 10^4$, our proposed T-forward method will take couple of seconds to solve it, while the interior point method will take 10^6 years to solve the same problem, which is a mission impossible for most of the current existing LP algorithms.

17. Conclusions

The main contributions of this paper include the closed-form solutions for LP, QCQP, and the near-closed-form solution for NLP. Another contribution is the T-forward method for NLP and the facet-forward method for LP. Both methods are polynomial time. The T-forward method and the facet-forward method are not only faster than any existing LP or NLP algorithms, but also give exact optimal solution when applied to LP.

REFERENCES

[1] [AG93] E. L. Allgower and K. Georg. "Continuation and path following". Acta Numerica, 1993, Cambridge University Press, Cambridge, UK, 1993, pp. 1–64.

[2] [AG97] E. L. Allgower and K. Georg. "Numerical path following". Handbook of Numerical Analysis, Vol. 5, P. G. Ciarlet and J. L. Lions, eds., North-Holland, Amsterdam, 1997, pp. 3–207.

[3] [AGJ00] P. Armand, J. C. Gilbert, and S. Jan-J´egou. "A feasible BFGS interior point algorithm for solving convex minimization problems". SIAM J. Optim., 11 (2000), pp. 199–222.

[4] [Bam86] E.R. Barnes. “A variation on Karmarkar’s algorithm for solving linear programming problems”. Mathematical Programming, 36:174–182, 1986.

[5] [CSE00] Computing in Science and Engineering, volume 2, no. 1, 2000.

[6] [CO00] A. R. Conn, N. I. M. Gould, D. Orban, and P. L. Toint. “A primal-dual trust-region algorithm for non-convex nonlinear programming”. Math. Program., 87 (2000), pp. 215–249.

[7] [CG97] A. R. Conn, N. I. M. Gould, and P. L. Toint. “A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds”. Math. Comp., 66 (1997), pp. 261–288, S1–S11.

[8] [FG98] A. Forsgren and P. E. Gill. “Primal-dual interior methods for nonconvex nonlinear programming”. SIAM J. Optim., 8 (1998), pp. 1132–1152.

[9] [Kar84] N. K. Karmarkar. “A new polynomial-time algorithm for linear programming”. *Combinatorica*, 4:373–395, 1984.

[10] [Kha79] L. G. Khachiyan, “A Polynomial Algorithm in Linear Programming”. Doklady Akademii Nauk SSSR 244:S (1979), 1093-1096, translated in Soviet Mathematics - Doklady 20:1 (1979), 191-194.

[11] [KK79] Kozlov, M. K.; S. P. Tarasov and Leonid G. Khachiyan (1979). “Polynomial solvability of convex quadratic programming”. Doklady Akademii Nauk SSSR 248: 1049–1051, translated in Soviet Mathematics - Doklady 20: 1108–1111

[12] [KM72] V. Klee, G. Minty. “How good is the simplex algorithm?” Inequalities III, New York-London: Academic Press: pp. 159-175.

[13] [Meh92] Mehrotra, Sanjay. "On the Implementation of a Primal-Dual Interior Point Method". SIAM Journal on Optimization 2 (4): 575, 1992.

[14] [MW01] G. P. McCormick and C. Witzgall. “Logarithmic SUMT limits in convex programming”. Math. Program., 90 (2001), pp. 113–145.

[15] [Pol92] R. Polyak. “Modified barrier functions (theory and methods)”. Math. Program., 54 (1992), pp. 177–222.

[16] [VM86] R. J. Vanderbei, M. S. Meketon, and B. A. Freedman. “A modification of Karmarkar’s linear programming algorithm”. *Algorithmica*, 1:395–407, 1986.

[17] [Wri04] M. H. Wright, “The interior-point revolution in optimization: History, recent developments, and lasting consequences”. Bulletin of the American Mathematical Society 42: 39, 2004.