Lecture Notes for CAAM 378

A Quick Introduction to Linear Programming

(DRAFT)

Yin Zhang

Sept. 25, 2007

Contents

1 What is Linear Programming? 5
- 1.1 A Toy Problem . . . 5
- 1.2 From Concrete to Abstract . . . 6
- 1.3 A Standard Form . . . 8
- 1.4 Feasibility and Solution Sets . . . 10
- 1.5 Three Possibilities . . . 11

2 Vertices of the Feasibility Set 13
- 2.1 Matrix and Vector Partitions . . . 13
- 2.2 Convex Set, Polyhedron and Extreme Point . . . 15
  - 2.2.1 Definitions . . . 15
  - 2.2.2 Vertices of Feasibility Set . . . 16
  - 2.2.3 Basic Feasible Partitions and Vertices . . . 18
- 2.3 Exercises . . . 19

3 Simplex Method: First Look 21
- 3.1 Terminology . . . 21
- 3.2 Reduced Linear Program . . . 21
  - 3.2.1 Partitioning and Elimination . . . 21
  - 3.2.2 Reduced Linear Program . . . 22
  - 3.2.3 What Happens at A Basic Feasible Solution? . . . 23
- 3.3 One Iteration of the Simplex Method . . . 24
- 3.4 Do We Reach a New Vertex? . . . 25
- 3.5 Exercises . . . 25

4 Simplex Method: More Details 27
- 4.1 How to Start Simplex? – Two Phases . . . 27
  - 4.1.1 Phase-I: An Auxiliary Problem . . . 28

4.1.2 Phase-II: The Original Problem 29
4.2 Main Implemetational Issues 30
4.3 Degeneracy, Cycling and Stalling 30
4.4 Exercises 30

5 Duality and Optimality Conditions **33**
5.1 Dual Linear Program 33
5.2 Optimality Conditions 35
5.3 How to find a dual? 35
5.4 Exercises 38

6 Primal-Dual Interior-Point Methods **39**
6.1 Introduction 39
6.2 A Primal-Dual Method 40
6.3 Solving the Linear System 43
6.4 Convergence Considerations 44

A Introduction to Newton's Method **47**
A.1 An Example 50
A.2 Exercises 51

Chapter 1

What is Linear Programming?

An optimization problem usually has three essential ingredients: a variable vector x consisting of a set of unknowns to be determined, an objective function of x to be optimized, and a set of constraints to be satisfied by x.

A *linear program* is an optimization problem where all involved functions are linear in x; in particular, all the constraints are linear inequalities and equalities. *Linear programming* is the subject of studying and solving linear programs.

Linear programming was born during the second World War out of the necessity of solving military logistic problems. It remains one of the most used mathematical techniques in today's modern societies.

1.1 A Toy Problem

A local furniture shop makes chairs and tables. The projected profits for the two products are, respectively, \$20 per chair and \$30 per table. The projected demand is 400 chairs and 100 tables. Each chair requires 2 cubic feet of wood while each table requires 4 cubic feet. The shop has a total amount of 1,000 cubic feet of wood in stock. How many chairs and tables should the shop make in order to maximize its profit?

Let x_1 be the number of chairs and x_2 be the number of tables to be made. These are the two variables, or unknowns, for this problem. The shop wants to maximize its total profit, $20x_1 + 30x_2$, subject to the constraints that (a) the total amount of wood used to make the two products can not exceed the 1,000 cubic feet available, and (b) the numbers of chairs and tables to be

made should not exceed the demands. In additional, we should not forget that the number of chairs and tables made need to be nonnegative. Putting all these together, we have an optimization problem:

$$\begin{array}{ll} \max & 20x_1 + 30x_2 \\ \text{s.t.} & 2x_1 + 4x_2 \leq 1000 \\ & 0 \leq x_1 \leq 400 \\ & 0 \leq x_2 \leq 100 \end{array} \tag{1.1}$$

where $20x_1 + 30x_2$ is the objective function, "s.t." is the shorthand for "subject to" which is followed by the constraints of this problem.

This optimization problem is clearly a linear program where all the functions involved, both in the objective and in the constraints, are linear functions of x_1 and x_2.

1.2 From Concrete to Abstract

Let us look at an abstract production model. A company produces n products using m kinds of materials. For the next month, the unit profits for the n products are projected to be $c_1, c_2, \cdots, c_n$. The amounts of materials available to the company in the next month are $b_1, b_2, \cdots, b_m$. The amount of material i consumed by a unit of product j is given by $a_{ij} \geq 0$ for $i = 1, 2, \cdots, m$ and $j = 1, 2, \cdots, n$ (some a_{ij} could be zero if product j does not use material i). The question facing the company is, given the limited availability of materials, what quantity the company should produce in the next month for each product in order to achieve the maximum total profit?

The decision variables are obviously the amounts produced for the n products in the next month. Let us call them $x_1, x_2, \cdots, x_n$. The optimization model is *to maximize the total profit, subject to the material availability constraints for all m materials, and the nonnegativity constraints on the n variables.*

In mathematical terms, the model is the following linear program:

$$\begin{array}{ll} \max & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{s.t.} & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\ & \qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le b_m \\ & x_1, x_2, \cdots, x_n \ge 0 \end{array} \tag{1.2}$$

The nonnegativity constraints can be vital, but are often forgotten by beginners. Why is nonnegativity important here? First, in the above context, it does not make sense to expect the company to produce a negative amount of a product. Moreover, if one product, say product k, is not profitable, corresponding to $c_k < 0$, without the nonnegativity constraints the model would produce a solution $x_k < 0$ and generate a profit $c_kx_k > 0$. In fact, since a negative amount of product k would not consume any material but instead "generate" materials, one could drive the profit to infinity by forcing x_k to go to negative infinity. Hence, the model would be wrong had one forgotten nonnegativity.

The linear program in (1.2) is tedious to write. One can shorten the expressions using the summation notation. For example, the total profit can be represented by the left-hand side instead of the right-hand side of the following identity

$$\sum_{i=1}^{n} c_ix_i = c_1x_1 + c_2x_2 + \cdots + c_nx_n.$$

However, a much more concise way is to use matrices and vectors. If we let $c = (c_1, c_2, \cdots, c_n)^T$ and $x = (x_1, x_2, \cdots, x_n)^T$. Then the total profit becomes c^Tx. In a matrix-vector notation, the linear program (1.2) becomes

$$\begin{array}{ll} \max & c^Tx \\ \text{s.t.} & Ax \le b \\ & x \ge 0 \end{array} \tag{1.3}$$

where $A \in \Re^{m\times n}$ and $b \in \Re^m$. The inequalities involving vectors are always understood as component-wise comparisons. For example, the n-vector $x \ge 0$ means that each component $x_i \ge 0$ for $i = 1, 2, \cdots, n$.

1.3 A Standard Form

A linear program can have an objective of either minimization or maximization, while its constraints can have any combination of linear inequalities and equalities. It is impractical to study linear programs and to design algorithms for them without introducing a so-called *standard form* – a unifying framework encompassing most, if not all, individual forms of linear programs. Different standard forms exist in the literature that may offer different advantages under different circumstances. Here we will use the following standard form:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \tag{1.4}$$

where the matrix A is $m \times n$, $b \in \Re^m$ and $c, x \in \Re^n$. The triple (A, b, c) represents problem data that needs to be specified, and x is the variable to be determined. In fact, once the size of A is given, the sizes of all the other quantities follow accordingly from the rule of matrix multiplication.

In plain English, a standard linear program is one that is a minimization problem with a set of equality constraints but no inequality constraints except nonnegativity on all variables.

We will always assume that A has full rank, or in other words, the rows of A are linearly independent which ensures that the equations in $Ax = b$ are consistent for any right-hand side $b \in \Re^m$. We make this assumption to simplify the matter without loss of generality, because redundant or inconsistent linear equations can always be detected, and removed through standard linear algebra techniques. Moreover, we normally require that the matrix A have more columns than rows, that is $m < n$. This requirement, together with the full rank of A, ensures that there are infinitely many solutions to the equation $Ax = b$, leaving degrees of freedom for nonnegative and optimal solutions.

A linear program is said to be equivalent to another linear program if an optimal solution of the former, if it exists, can be obtained from an optimal solution of the latter through some simple algebraic operations. This equivalence allows one to solve one and obtain a solution to the other.

We claim that every linear program is equivalent to a standard-form linear program through a transformation, which usually requires adding extra variables and constraints. Obviously, maximizing a function is equivalent to minimizing the negative of the function. A common trick to transform an

inequality $a^T x \le \beta$ into an equivalent equality is to add a so-called *slack* variable η so that

$$a^T x \le \beta \iff a^T x + \eta = \beta,\ \eta \ge 0.$$

Let us consider the toy problem (1.1). With the addition of slack variables x_3, x_4, x_5 (we can name them anyway we want), we transform the linear program on the left to the one on the right:

$$\begin{array}{lc} \max & 20x_1 + 30x_2 \\ \text{s.t.} & 2x_1 + 4x_2 \le 1000 \\ & x_1 \le 400 \\ & x_2 \le 100 \\ & x_1, x_2 \ge 0 \end{array} \quad \Longrightarrow \quad \begin{array}{lc} \max & 20x_1 + 30x_2 \\ \text{s.t.} & 2x_1 + 4x_2 + x_3 = 1000 \\ & x_1 + x_4 = 400 \\ & x_2 + x_5 = 100 \\ & x_1, x_2, x_3, x_4, x_5 \ge 0 \end{array}$$

After switching to minimization, we turn the linear program on the right to an equivalent linear program of the standard form:

$$\begin{array}{lc} \min & -20x_1 - 30x_2 + 0x_3 + 0x_4 + 0x_5 \\ \text{s.t.} & 2x_1 + 4x_2 + x_3 + 0x_4 + 0x_5 = 1000 \\ & x_1 + 0x_2 + 0x_3 + x_4 + 0x_5 = 400 \\ & 0x_1 + x_2 + 0x_3 + 0x_4 + x_5 = 100 \\ & x_1, x_2, x_3, x_4, x_5 \ge 0 \end{array} \tag{1.5}$$

where $c^T = -(20\ 30\ 0\ 0\ 0)$, b is unchanged and

$$A = \left(\begin{array}{cc|ccc} 2 & 4 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right).$$

This new coefficient matrix is obtained by appending the 3-by-3 identity matrix to the right of the original coefficient matrix.

Similarly, the general linear program (1.3) can be transformed into

$$\begin{array}{lc} \min & -c^T x \\ \text{s.t.} & Ax + s = b \\ & x, s \ge 0 \end{array} \tag{1.6}$$

where $s \in \Re^m$ is the slack variable vector. This linear program is in the standard form because it is a minimization problem with only equality constraints and nonnegativity on all variables. The variable in (1.6) consists of

$x \in \Re^n$ and $s \in \Re^m$, and the new data triple is obtained by the construction

$$A \to [A \;\; I], \;\; b \to b, \;\; c \to \begin{pmatrix} -c \\ 0 \end{pmatrix},$$

where I is the m-by-m identity matrix and $0 \in \Re^m$.

1.4 Feasibility and Solution Sets

Let us call the following set

$$\mathcal{F} = \{x \in \Re^n : Ax = b, \; x \geq 0\} \subset \Re^n \tag{1.7}$$

the *feasibility set* of the linear program (1.4). Points in the feasibility set are called *feasible points*, from which we seek an *optimal solution* x^* that minimizes the objective function $c^T x$. With the help of the feasibility set notation, we can write our standard linear program into a concise form

$$\min\{c^T x : x \in \mathcal{F}\}. \tag{1.8}$$

A *polyhedron* in $\Re^n$ is a subset of $\Re^n$ defined by all points satisfying a finite collection of linear inequalities and/or equalities. For example, a *hyper-plane*

$$\{x \in \Re^n : a^T x = \beta\}$$

is a polyhedron where $a \in \Re^n$ and $\beta \in \Re$, and a *half-space*

$$\{x \in \Re^n : a^T x \leq \beta\}$$

is a polyhedron as well. In geometric terms, a polyhedron is nothing but the intersection of a finite collection of hyper-planes and half-spaces. In particular, the empty set and a singleton set (that contains only a single point) are polyhedra.

Clearly, the feasibility set $\mathcal{F}$ in (1.7) for linear program (1.4) is a polyhedron in $\Re^n$. It may be empty if some constraints are contradictory (say, $x_1 \leq -1$ and $x_1 \geq 1$), or it may be an unbounded set, say,

$$\mathcal{F} = \{x \in \Re^2 : x_1 - x_2 = 0, x \geq 0\} \subset \Re^2, \tag{1.9}$$

which is the half diagonal-line emitting from the origin towards infinity. Most of the time, $\mathcal{F}$ will be a bounded, nonempty set. A bounded polyhedron is called a *polytope*.

The *optimal solution set*, or just *solution set* for short, of a linear program consists of all feasible points that optimize its objective function. In particular, we denote the solution set of the standard linear program (1.4) or (1.8) as $\mathcal{S}$. Since $\mathcal{S} \subset \mathcal{F}$, $\mathcal{S}$ is empty whenever $\mathcal{F}$ is empty. However, $\mathcal{S}$ can be empty even if $\mathcal{F}$ is not.

The *purpose of a linear programming algorithm* is to determine whether $\mathcal{S}$ is empty or not and, in the latter case, to find a member x^* of $\mathcal{S}$. In case such an x^* exists, we can write $\mathcal{S} = \{x \in \mathcal{F} : c^T x = c^T x^*\}$ or

$$\mathcal{S} = \{x \in \Re^n : c^T x = c^T x^*\} \cap \mathcal{F}. \tag{1.10}$$

That is, $\mathcal{S}$ is the intersection of a hyper-plane with the polyhedron $\mathcal{F}$. Hence, $\mathcal{S}$ itself is a polyhedron. If the intersection is a singleton $\mathcal{S} = \{x^*\}$, then x^* is the unique optimal solution; otherwise, there must exist infinitely many optimal solutions to the linear program.

1.5 Three Possibilities

A linear program is *infeasible* if its feasibility set is empty; otherwise, it is *feasible*.

A linear program is *unbounded* if it is feasible but its objective function can be made arbitrarily "good". For example, if a linear program is a minimization problem and unbounded, then its objective value can be made arbitrarily small while maintaining feasibility. In other words, we can drive the objective value to negative infinity within the feasibility set. The situation is similar for an unbounded maximization problem where we can drive the objective value to positive infinity. Clearly, a linear program is unbounded only if its feasibility set is an unbounded set. However, a unbounded feasibility set does not necessarily imply that the linear program itself is unbounded.

To make it clear, let us formally define the term *unbounded* for a set and for a linear program. We say a set $\mathcal{F}$ is unbounded if there exists a sequence $\{x^k\} \subset \mathcal{F}$ such that $\|x^k\| \to \infty$ as $k \to \infty$. On the other hand, we say a linear program $\min\{c^T x : x \in \mathcal{F}\}$ is unbounded if there exists a sequence $\{x^k\} \subset \mathcal{F}$ such that $c^T x^k \to -\infty$ as $k \to \infty$. Hence, for a linear program the term *unbounded* means *objective unbounded*.

When the feasibility set $\mathcal{F}$ is unbounded, whether or not the corresponding linear program is unbounded depends entirely on the objective function.

For example, consider $\mathcal{F}$ given by (1.9). The linear program

$$\max\{x_1 + x_2 : (x_1, x_2) \in \mathcal{F}\} = \max\{2x_1 : x_1 \geq 0\}$$

is unbounded. However, when the objective is changed to minimization instead, the resulting linear program has an optimal solution at the origin.

If a linear program is feasible but not (objective) unbounded, then it must achieve a finite optimal value within its feasibility set; in other words, it has an optimal solution $x^* \in \mathcal{S} \subset \mathcal{F}$.

To sum up, for any given linear program there are three possibilities:

1. The linear program is infeasible, i.e., $\mathcal{F} = \emptyset$. In this case, $\mathcal{S} = \emptyset$.

2. The linear program is feasible but (objective) unbounded. In this case, $\mathcal{F}$ is an unbounded set, but $\mathcal{S} = \emptyset$.

3. The linear program is feasible and has an optimal solution $x^* \in \mathcal{S} \subset \mathcal{F}$. In this case, the feasibility set $\mathcal{F}$ can be either unbounded or bounded.

These three possibilities imply that if $\mathcal{F}$ is both feasible and bounded, then the corresponding linear program must have an optimal solution $x^* \in \mathcal{S} \subset \mathcal{F}$, regardless of what objective it has.

Chapter 2

Vertices of the Feasibility Set

We have seen that both the feasibility set and the solution set of a linear program are polyhedra in $\Re^n$. Like polygons in two or three dimensional spaces, polyhedra in $\Re^n$ generally have "corners" or vertices.

2.1 Matrix and Vector Partitions

It will be convenient for us to use matrix partitions in the development of theory and algorithms for linear programming. This section introduces necessary notations for partitioned matrices and vectors, which can be skipped by advanced readers.

Consider a 3×7 matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \end{pmatrix} = \begin{pmatrix} A_1 & A_2 & \cdots & A_7 \end{pmatrix},$$

where A_j is the j-th column of A; i.e., $A_j = (a_{1j}\ a_{2j}\ a_{3j})^T$ is a 3×1 vector for $j = 1, 2, \cdots, 7$. For any vector $x \in \Re^7$,

$$Ax = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{17}x_7 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{27}x_7 \\ a_{31}x_1 + a_{32}x_2 + \cdots + a_{37}x_7 \end{pmatrix} = \sum_{j=1}^{7} A_j x_j.$$

Let us partition the index set $\{1, 2, 3, 4, 5, 6, 7\}$ into two subsets B and N with an arbitrary order within each subset, say,

$$B = \{5, 2, 4\}, \quad N = \{1, 6, 7, 3\}.$$

Then A_B and A_N are, respectively, the 3×3 and 3×4 sub-matrices of A whose columns are those of A corresponding to the indices and orders within B and N, respectively. Namely,

$$A_B = [A_5\ A_2\ A_4], \quad A_N = [A_1\ A_6\ A_7\ A_3].$$

Similarly, $x_B \in \Re^3$ and $x_N \in \Re^4$ are, respectively, sub-vectors of x whose components are those of x corresponding to the indices and orders within B and N, respectively. Namely,

$$x_B = (x_5\ x_2\ x_4)^T, \quad x_N = (x_1\ x_6\ x_7\ x_3)^T.$$

Corresponding to the index-set partition (B, N),

$$Ax = \sum_{j=1}^{7} A_j x_j = \sum_{j\in B} A_j x_j + \sum_{j\in N} A_j x_j = A_B x_B + A_N x_N.$$

Moreover, the linear system of equations $Ax = b$ can be written as

$$A_B x_B + A_N x_N = b$$

for any right-hand side vector $b \in \Re^3$.

Obviously, these notations can be extended to the general case where $A \in \Re^{m\times n}$, $x \in \Re^n$ and $b \in \Re^m$. For any index partition (B, N) with

$$B \cup N = \{1, 2, \cdots, n\}, \quad B \cap N = \emptyset, \tag{2.1}$$

there holds

$$Ax = b \quad \Leftrightarrow \quad A_B x_B + A_N x_N = b, \tag{2.2}$$

where the symbol "$\Leftrightarrow$" denotes equivalence. Moreover, if B contains m indices (i.e., $|B| = m$) and the square matrix A_B is nonsingular, there holds

$$Ax = b \quad \Leftrightarrow \quad x_B = A_B^{-1}(b - A_N x_N). \tag{2.3}$$

That is, we can solve the equation $Ax = b$ to obtain x_B as a function of x_N. Furthermore, in this case,

$$x \geq 0 \quad \Leftrightarrow \quad A_B^{-1}(b - A_N x_N) \geq 0,\ x_N \geq 0. \tag{2.4}$$

2.2 Convex Set, Polyhedron and Extreme Point

2.2.1 Definitions

Definition 1 (Convex Set). *A set $\mathcal{C} \subset \Re^n$ is a* **convex set** *if*

$$x, y \in \mathcal{C} \Rightarrow \lambda x + (1-\lambda)y \in \mathcal{C}; \ \ \forall \lambda \in [0,1];$$

that is, the line segment connecting any two members lies entirely in the set. By this definition, an empty set is a convex set.

Definition 2 (Extreme Point). *A point x is an* **extreme point** *of a convex set $\mathcal{C} \subset \Re^n$ if it does not lie in between any other two members of the set, that is, for any $y, z \in \mathcal{C}$ such that $y \neq x \neq z$,*

$$x \neq \lambda y + (1-\lambda)z, \ \ \forall \lambda \in (0,1).$$

Clearly, any extreme point must be on the boundary of the set and can not be an interior point.

Definition 3 (Polyhedron and its vertices). *A* **polyhedron** *in $\Re^n$ is a set of points in $\Re^n$ defined by a finite collection of linear equalities and/or inequalities (i.e., the intersection of a finite number of hyper-planes and half-spaces). A bounded polyhedron is also called a polytope. An extreme point of a polyhedron is also called a* **vertex** *of the polyhedron.*

Figure 2.1: **Convex sets**

Proposition 1. *The following statements are true.*

1. *A hyperplane $\{x \in \Re^n : a^T x = \beta\}$ is convex.*
2. *A half-space $\{x \in \Re^n : a^T x \leq \beta\}$ is convex.*

3. The intersection of a collection of convex sets is convex.

4. A polyhedron is convex.

The following facts will be useful for optimizing a linear function in closed convex set $\mathcal{C}$, that is,

$$\min\{c^T x : x \in \mathcal{C} \subset \Re^n\}. \tag{2.5}$$

This is a problem more general than a linear program. It is called a convex program.

Proposition 2. *If a linear function $c^T x$ has a unique minimum (maximum) on a closed convex set $\mathcal{C}$, then the minimum (maximum) must be attained at an extreme point of the set.*

The proof is by contradiction. Suppose that the unique minimum is attained at a non-extreme point $x \in \mathcal{C}$, then there must exist two other points $y, z \in \mathcal{C}$ such that $c^T x < c^T y$, $c^T x < c^T z$ and $x = \lambda y + (1-\lambda)z$, where $\lambda \in (0, 1)$. Hence,

$$c^T x = \lambda c^T y + (1-\lambda) c^T z < \lambda c^T x + (1-\lambda) c^T x = c^T x,$$

which is a contradiction.

2.2.2 Vertices of Feasibility Set

Let us examine closely a particular polyhedron – the feasibility set of our standard linear program (1.4):

$$\mathcal{F} = \{x \in \Re^n : Ax = b,\ x \geq 0\}, \tag{2.6}$$

where the matrix $A \in \Re^{m \times n}$ and the vector $b \in \Re^m$ are given. The vector inequality $x \geq 0$ is understood as component-wise, i.e., $x_i \geq 0$ for $i = 1, 2, \cdots, n$. We will call $\mathcal{F}$ the "standard" polyhedron because we will treat it as the representative of polyhedrons.

We will assume that (a) the matrix A has more columns than rows so that $m < n$; and (b) the ranks of A is m. These assumptions guarantee that in general the equation $Ax = b$ will have infinitely many solutions. While the first assumption will occur naturally from the context of linear programming, the second one can always be achieved by a routine linear-algebra procedure whenever the original matrix A is not full rank.

Extreme points of a polyhedron are the "corners" of the set. They are easy to tell only in two- or three-dimensional spaces where visualization is possible. In high dimensional spaces, we need some algebraic characterization for extreme points.

For any nonnegative vector $x \in \Re^n$, we can define a natural index partition:

$$P \equiv P(x) := \{i : x_i > 0\}, \quad O \equiv O(x) := \{i : x_i = 0\}, \tag{2.7}$$

where we drop the dependence of P and O on x whenever it is clear from the context. Furthermore, we assume that the indices within P and O are arranged in some orders, even though how they are ordered is inconsequential.

Lemma 1. *A point $x \in \mathcal{F}$ is an extreme point of $\mathcal{F}$ if and only if the matrix $A_{P(x)}$ is of full column rank.*

Proof. Without loss of generality, we assume that $A = [A_P \; A_O]$ and $x^T = [x_P^T \; x_O^T]$ (otherwise, a re-ordering will suffice). Clearly, $Ax = A_P x_P = b$, $x_P > 0 \in \Re^{|P|}$ and $x_O = 0 \in \Re^{|O|}$ with $|P| + |O| = n$

We first prove the necessary condition by contradiction. Suppose x is an extreme point of $\mathcal{F}$ but A_P is not of full rank. Then there exists a nonzero vector $v \in \Re^{|P|}$ such that $A_p v = 0$. Moreover, there exists a sufficiently small but positive scalar $\tau \in \Re$ such that $x_P \pm \tau v \geq 0$ since $x_P > 0$. Let

$$y = \begin{pmatrix} x_P + \tau v \\ 0 \end{pmatrix} \in \Re^n, \quad z = \begin{pmatrix} x_P - \tau v \\ 0 \end{pmatrix} \in \Re^n,$$

then $Ay = Az = b$ and $y, z \geq 0$; that is, $y, z \in \mathcal{F}$. Since

$$x = \frac{1}{2}y + \frac{1}{2}z$$

for $x \neq y \in \mathcal{F}$ and $x \neq z \in \mathcal{F}$, we have arrived at a contradiction to the hypothesis that x is an extreme point of $\mathcal{F}$.

We now prove the sufficient condition by contradiction. Suppose that A_P is of full column rank but x is not an extreme point of $\mathcal{F}$. Then there exist distinct points $y, z \in \mathcal{F}$ with $y \neq x \neq z$ such that for some $\lambda \in (0, 1)$,

$$\begin{pmatrix} x_P \\ 0 \end{pmatrix} = \begin{pmatrix} \lambda y_P + (1-\lambda) z_P \\ \lambda y_O + (1-\lambda) z_O \end{pmatrix}.$$

The nonnegativity of y and z implies that $y_O = z_O = 0$, but $y \neq z$, hence $y_P - z_P \neq 0$. Therefore, since $A_P y_P = A_P z_P = b$, $A_P(y_P - z_P) = 0$, contradicting to the hypothesis that A_P is of full column rank. □

Theorem 1. *Let $\mathcal{F}$ be defined in (2.6) where $A \in \Re^{m\times n}$ has a rank equal to m. A point $x \in \mathcal{F}$ is an extreme point (vertex) of $\mathcal{F}$ if and only if there exists an index partition (B, N) of $\{1, 2, \cdots, n\}$ (see (2.1)) such that*

$$|B| = \text{rank}(A_B) = m, \quad A_B^{-1}b \geq 0, \tag{2.8}$$

and

$$x_B = A_B^{-1}b, \quad x_N = 0. \tag{2.9}$$

Proof. Let condition (2.8) hold and x be defined as in (2.9). Since $x \geq 0$ and

$$Ax = A_B x_B + A_N x_N = A_B x_B = b,$$

clearly $x \in \mathcal{F}$ and $P(x) \subset B$. Because the columns of A_B are linearly independent, so are those of $A_{P(x)}$. By Lemma 1, x is a vertex of $\mathcal{F}$.

Now suppose that x is a vertex of $\mathcal{F}$. By Lemma 1, $A_P := A_{P(x)}$ has a full column rank $|P|$. If $|P| = m$, then $B = P$ satisfies the conditions (2.8) and (2.9). On the other hand, if $|P| < m$, we can always expand the index set P to a larger index set B so that $P \subset B$ and $|B| = m = \text{rank}(A_B)$. For such an index set B (which is generally not unique), both (2.8) and (2.9) hold. This proves the theorem. □

2.2.3 Basic Feasible Partitions and Vertices

Definition 4 (Basic Feasible Partition)**.** *An index partition (B, N) is a basic feasible partition for the polyhedron $\mathcal{F}$ whenever (2.8) holds. Moreover, a basic feasible partition is called nondegenerate if $A_B^{-1}b > 0$; otherwise it is degenerate.*

In view of Theorem 1, under the condition $\text{rank}(A) = m$ we have the following correspondence between the vertices of $\mathcal{F}$ and the basic feasible partitions for $\mathcal{F}$.

1. Each basic feasible partition corresponds to a vertex of $\mathcal{F}$, which is defined as in (2.9).

2. Each nondegenerate vertex of $\mathcal{F}$ corresponds to a basic feasible partition. In this case, $P(x) = B$.

3. In general, each degenerate vertex of $\mathcal{F}$ corresponds to multiple basic feasible partitions. In this case, $|P(x)| < m$ and $P(x)$ must be expanded to B. Many such expansions exist.

2.3 Exercises

1. Prove that the optimal solution set of a convex program defined in (2.5) is a convex set.

2. Consider the polytope

$$\mathcal{F} = \{x \in \Re^4 : x_1 + x_2 + x_3 = 1, x_2 + x_4 = 1, x \geq 0\}.$$

(a) Use Lemma 1 to determine whether or not the following points are vertices of $\mathcal{F}$: $(1, 0, 0, 1)^T, (\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2})^T$, and give your reasons.

(b) Use Theorem 1 to verify that the point $x = (0, 1, 0, 0)^T$ is a vertex of $\mathcal{F}$. Find all possible index partitions so that the conditions of the theorem hold.

Chapter 3

Simplex Method: First Look

3.1 Terminology

Due to historical reasons, the following terminology has been widely used in the linear programming literature.

- **Feasible solution**: A set of values for the vector x that satisfy all the constraints, including the nonnegativity.
- **Basic feasible solution**: A feasible solution that has m basic variables taking nonnegative values and $n - m$ nonbasic variables taking zero values. In addition, the submatrix of A corresponging the basic variables is an m by m nonsingular matrix. The simplex method works exclusively with basic feasible solutions.
- **Degenerate basic feasible solution**: A basic feasible solution that has one or more zero basic variable. On the other hand, all basic variables are positive in a non-degenerate basic feasible solutions.

3.2 Reduced Linear Program

3.2.1 Partitioning and Elimination

Given an index partition

$$B \cup N = \{1, 2, \cdots, n\}, \;\; B \cap N = \emptyset$$

such that the submatrix A_B of A, formed by the columns A with indices in B, is $m \times m$ and nonsingular (therefore, $|B| = m$ and $|N| = n - m$). Similarly, we define A_N as the submatrix of A formed by the columns A with indices in N. Accordingly, we partition x and c into $[x_B, x_N]$ and $[c_B, c_N]$, respectively. That is,

$$\begin{array}{rcl} \{1, 2, \cdots, n\} & \rightarrow & [B \mid N] \\ A & \rightarrow & [A_B \mid A_N] \\ c & \rightarrow & [c_B \mid c_N] \\ x & \rightarrow & [x_B \mid x_N] \end{array}$$

The actual ordering of indices inside B or N is immaterial as long as it is consistent for all the portioned quantities.

Under such a partition where A_B is nonsingular, we have the following equivalence:

$$Ax = b \iff A_B x_B + A_N x_N = b \iff x_B = A_B^{-1}(b - A_N x_N). \quad (3.1)$$

In this way, the really independent variable is x_N, on which x_B is dependent. Upon substituting the expression for x_B into that for $c^T x$, we have

$$\begin{array}{rcl} c^T x & = & c_B^T x_B + c_N^T x_N \\ & = & c_B^T A_B^{-1} b - c_B^T A_B^{-1} A_N x_N + c_N^T x_N \\ & = & b^T (A_B^{-T} c_B) + (c_N - A_N^T A_B^{-T} c_B)^T x_N; \end{array}$$

or equivalently,

$$c^T x = b^T y + s_N^T x_N, \quad (3.2)$$

where we have used the short-hand notation:

$$y = A_B^{-T} c_B \quad \text{and} \quad s_N = c_N - A_N^T y. \quad (3.3)$$

Note that y and s_N depend solely on the given data (A, b, c) and the given partitioning, but independent of x. They will change as the partitioning changes.

3.2.2 Reduced Linear Program

For any given basic feasible partition (B, N), we have used the equation $Ax = b$ to eliminate the variables in x_B from the original LP problem. The

reduced LP problem, corresponding to a basic feasible partition (B, N), has no more equality constraints and involves only variables in x_N.

$$\begin{array}{ll} \min\limits_{x_N} & b^T y + s_N^T x_N \\ \text{s.t.} & A_B^{-1} b - A_B^{-1} A_N x_N \geq 0 \\ & x_N \geq 0. \end{array} \tag{3.4}$$

The first inequality is nothing but the nonnegativity for x_B. This reduced LP is completely equivalent to the original LP. However, we can deduce useful information from this reduced LP that is not apparent in the original LP.

3.2.3 What Happens at A Basic Feasible Solution?

If the partitioning is associated with a basic feasible solution, then B corresponds to a "*basis*" and N to a "*nonbasis*" such that the "*basic variables*" $x_B = A_B^{-1} b \geq 0$ and the "*nonbasic variables*" $x_N = 0$. In this case, we observe the following.

- Any feasible change that can be made to x_N is to increase one or more of its elements from zero to some positive value.
- If $s_N \geq 0$, then it is not possible to decrease the objective function value by increasing one or more element of x_N. Hence we can conclude that the current basic feasible solution is optimal.
- On the other hand, if any element of s_N is negative, we can try to increase the corresponding variable in x_N, say x_j in terms of the original indexing, as much as possible so long as we maintain the inequality constraints in the reduced LP.
- When x_j is increased from zero to t while all other elements of x_N remain at zero, then $A_N x_N$ becomes tA_j where A_j is the j-th column of A.
- In the above case, the first inequality in the reduced LP reduces to

$$A_B^{-1} b \geq t A_B^{-1} A_j.$$

- The problem is unbounded if the above inequality poses no restriction on how large t can be, meaning that one can arbitrarily improve the objective by indefinitely increases t.

3.3 One Iteration of the Simplex Method

Given date $A \in \Re^{m\times n}$, $c \in \Re^n$ and a basic feasible solution $x \in \Re^n$ along with its corresponding index partition (B, N) with $x_N = 0$ and $x_B = A_B^{-1}b \geq 0$. The indices in B and N are, respectively, in arbitrary but fixed orders.

Step 1 (Dual estimates)
Solve $A_B^T y = c_B$ for y and compute $s_N = c_N - A_N^T y$.

Step 2 (Check optimality)
If all $s_N \geq 0$, stop "Optimum Reached".

Step 3 (Select entering variable)
Choose an index q such that $(s_N)_q < 0$, and let j be the corresponding (the q-th) index in N. The default choice is the most negative component of s_N. Here q is a local index for $s_N \in \Re^{n-m}$ and j is a global index for $x \in \Re^n$. The variable x_j, currently non-basic, will be entering the basis.

Step 4 (Compute step)
Solve $A_B d = A_j$ for $d \in \Re^m$ where A_j is the j-th column of A. This vector d will be used to update x_B from its current value $A_B^{-1}b$ to a new value in order to maintain $Ax = b$ as x_j increases.

Step 5 (Check Objective Unboundedness)
If all $d \leq 0$, stop "Objective Unbounded".

Step 6 (Select leaving variable)
Compute the largest step length t so that $A_B^{-1}b - td \geq 0$; namely,

$$t^* = \min_{d_k>0} \left\{ \frac{(x_B)_k}{d_k} \right\} \equiv \frac{(x_B)_p}{d_p} \quad \text{for some } p \in \{1, 2, \cdots, m\}. \tag{3.5}$$

Here the minimum occurs at $(x_B)_p = x_i$ for index $i \in B$, p is a local index for $x_B \in \Re^m$ and i is a global index for $x \in \Re^n$. The variable x_i, currently basic, will be leaving the basis to become non-basic.

Step 7 (Update variables)
Let $x_j = t^*$ and $x_B \Leftarrow x_B - t^* d$. Note that after the update the p-th basic variable $(x_B)_p \equiv x_i$ is zero (it has left the basis and become non-basic).

Step 8 (Update partition)
Move the index i from B to N and j from N to B (that is, i leaves the basis and j enters the basis). Go to Step 1.

3.4 Do We Reach a New Vertex?

After each iteration, we certainly have a new partition B and N (though they differ with the previous one by only a single index). With the new partition, we have a new $x_B \geq 0$ and a new $x_N = 0$. Therefore, if the new basis matrix, say A_B^+, is again nonsingular, then we have a new basic feasible solution.

Is the new basis matrix A_B^+ still nonsingular after each update? The matrix A_B^+ is nonsingular if and only if $A_B^{-1}A_B^+$ is nonsingular (prove it by youself). We note that A_B^+ differs from A_B only in the r-th column where the column A_i has been replaced by A_j. If we denote the k-th column of the identity matrix by e_k, then

$$A_B^{-1}(A_B^+ e_k) = A_B^{-1}(A_B e_k) = e_k, \quad \forall\, k \neq r.$$

In addition, we note that the new p-th column of A_B^+ is A_j (where j is a global index) that satisfies $A_B^{-1}A_j = d$. Therefore,

$$A_B^{-1}A_B^+ = [e_1 \ \cdots \ e_{p-1}\ d\ e_{p+1} \ \cdots \ e_m].$$

This matrix is nonsingular if and only if the p-th element of d is nonzero. For example, when $m = 7$ and $p = 4$,

$$A_B^{-1}A_B^+ = \begin{pmatrix} 1 & 0 & 0 & d_1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & d_5 & 1 & 0 & 0 \\ 0 & 0 & 0 & d_6 & 0 & 1 & 0 \\ 0 & 0 & 0 & d_7 & 0 & 0 & 1 \end{pmatrix},$$

and

$$|\det(A_B^{-1}A_B^+)| = |d_4|.$$

Since the p-th element of d satisfies $d_p > 0$ (see equation (3.5)), we conclude that indeed A_B^+ is nonsingular and the new feasible solution is indeed a basic feasible solution, i.e., a vertex.

3.5 Exercises

1. Let A and B are two square matrices of the same size and A is nonsingular. Prove that B is nonsingular if and only if $A^{-1}B$ is nonsingular.

Chapter 4

Simplex Method: More Details

4.1 How to Start Simplex? – Two Phases

Our simplex algorithm is constructed for solving the following standard form of linear programs:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

where A is $m \times n$. To start the algorithm, we need an initial basic feasible solution (or a vexter for the feasibility set). In general, such an initial basic feasible solution is not readily available. Some work is required.

To be absolutely clear, what we need is essentially a partition (B, N) of the index set $\{1, 2, \cdots, n\}$, with m indices in B, such that *the basis matrix A_B is nonsingular and $A_B^{-1} b \geq 0$*. Then we let the m basic variables to be $x_B = A_B^{-1} b$ and the rest of (non-basic) variables to be zeros. This gives us a basic feasible solution needed.

Let us see how we can start the simplex algorithm to solve the following LP:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0, \end{array} \tag{4.1}$$

where again A is $m \times n$. We already know that the main constraint $Ax \leq b$ is equivalent to $Ax + Is = b$ for $s \geq 0$. By adding thd slack variable s, we convert the LP into the standard form with m equations and $n+m$ variables. The coefficient matrix bacomes $\hat{A} = [A \;\; I]$ which is $m \times (n + m)$.

Whenever $b \geq 0$, the LP is clearly feasible since $x = 0$ satisfies the constraints. It is also easy to construct an initial basic feasible solutiuon. We can simply set $x = 0$ as the non-basic variables and $s = b \geq 0$ as the m basic variables. This way, we satisfy the equation $Ax + Is = b$ and in fact obtain a basic feasible solution. This is true becarse the corresponding basis matrix is the identity matrix and $\hat{A}_B^{-1} b = I * b = b \geq 0$. If we order our $n + m$ variables in the order $(x; s)$, then we have $B = \{1, 2, \cdots, n\}$ and $N = \{n+1, n+2, \cdots, n+m\}$. From this partition, we can start the simplex method.

4.1.1 Phase-I: An Auxiliary Problem

When b is not nonnegative, however, things are more complicated. First of all, we cannot easily tell whether the LP is feasible or not. Even it is feasible, an initial basic feasible solution is not obvious. To handle this case, we need to do a so-called phase-I procedure.

The purpose of phase-I is tow-folds. First, we would like to determine whether the original LP is feasible or not. Second, we would like to construct an initial basic feasible solution wheneven the original LP is feasible. Towards this end, we consider a so-called *auxiliary linear program*.

Since $x = 0$ and $s = b$ do not give a feasible solution when b has one or more negative component, we consider adding another new scalar variable t, called the auxiliary variable, that is chosen to make $s_i = b_i + t \geq 0$ for each component s_i of s. However, since this auxiliary variable is inconsistent with the original problem, we wish to eventually make it vanish if possible. The auxiliary problem is the following.

$$\begin{array}{ll} \min & t \\ \text{s.t.} & Ax + s - te = b \\ & x, s, t \geq 0. \end{array} \tag{4.2}$$

where t is the auxiliary scalar variable and $e = (1,\ 1, \cdots, 1)^T \in \Re^m$ is the vector of all ones. This linear program is always feasible and cannot be unbounded (prove it by yourself).

We will make the auxiliary variable t one of the basix variables and construct a basic feasible solution in the following manner. Recall that our of the $n + m + 1$ variables in $(x; s; t)$, we need m basic variables and $n + 1$ non-basic ones that are zeros.

We still set $x = 0$ as our n non-basic variables, so we need one more non-basic variable. We set

$$t = -\min_{1 \le i \le m} b_i := -b_k > 0,$$

namely, t takes the absolute value of the most negative element of b which is assumed to be s_k for some index k. Since t is positive, it must be a basic variable. Now the equation $Ax + s - te = b$ dictates that

$$s = b + te = b - b_k e \ge 0,$$

where clearly $s_k = 0$ (there might be other zero elements in s as well). We make s_k to be the last non-basic variable. For example, for the vector b given below, $t = -b_3 = 2$ and

$$s = b + te = \begin{pmatrix} 3 \\ -1 \\ -2 \\ 0 \end{pmatrix} + 2 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 0 \\ 2 \end{pmatrix} \ge 0.$$

With these choices, we have (i) the feasibility: $Ax + s - te = b$ with $x, s, t \ge 0$; (ii) m basic variables, in t and in the $m - 1$ components of s excluding s_k; (iii) $n + 1$ non-basic variables, in $x = 0$ and in $s_k = 0$. The basis matrix consists of the $m - 1$ columns of the identity matrix excluding the k-th column, plus the column $-e$. It is nonsingular. So we have found a basic feasible solution for the auxiliary problem!

Starting from this basic feasible solution, we can use simplex method to solve the auxiliary problem. In the subsequent simplex iterations, we should always let t have the priority to leave the basis whenever it qualifies as a candidate. Once t leave the basis, the optimal objective value is reached.

At the optimality, if $t = 0$, then the original LP is feasible; otherwise ($t > 0$), it must be infeasible and we stop here.

4.1.2 Phase-II: The Original Problem

In the case where the original problem is feasible, we delete the column corresponding to the auxiliary variable t (it's non-basic and zero anyway) from the auxiliary problem, and shrink the non-basis by one. This produces a basic feasible solution for the original problem (since t is gone), from which we can start the simplex method to solve the original problem.

4.2 Main Implemetational Issues

We observe that in the revised simplex algorithm, the main computational task appears to be solving two linear systems of equations: one in Step 1 and another in Step 4. The two systems involve the same matrix A_B, though it appears in Step 4 as the transpose.

(TBC)

4.3 Degeneracy, Cycling and Stalling

So far everything sounds so rosy. But there is one place where things can go wrong. That is, what if x_B has a zero component which happens to correspond to a positive component of d. In this case, $t^* = 0$ in (3.5), and no real change whatsoever, except for the partitioning, has been made to the variables. The objective value also remains the same. When this happens to a simplex iteration, such an iteration is called a *degenerate simplex iteration*. This situation is also called *stalling*. What are the consequences of degenerate iterations?

First of all, degenerate iterations that make no progress are not productive. At the least, they slow down the convergence if convergence eventually occur. Indeed, in the worst case, it can prevent convergence from occuring. This phenomenon is called *cycling* where the simplex method keeps changing basis but going on circle. Namely, after a while it comes back to a basis that has been visited before, while staying at the same vertex.

There exist some deliberate pivoting procedures (for choosing entering and leaving variables) that can avoid cycling. Fortunately, as dangerous as it is in theory, cycling almost never happens in practice and therefore is not of a real concern. On the other hand, generacy does cause real problems for simplex methods because it occurs so frequently, even more often than nondegeneracy, in practice (well, people prefer redundancy than inadequacy), and it can significantly degrade the performance of simplex methods.

4.4 Exercises

1. Prove that the auxiliary LP is feasible and cannot be unbounded.

2. Prove that an $m \times m$ matrix consists of the $m - 1$ columns of the

identity matrix plus the column $-e$, where $e \in \Re^m$ is the vector of all ones, is nonsingular.

3. Let us arrange the variables in the order $x; s; t$ with n variables in x, m variables in s and one variable in t. Write down the index sets B and N for the initial basic feasible solution constructed for the auxiliary problem.

4. Prove that if the optimal value of the auxiliary LP is positive, then the original LP is infeasible. Otherwise, it is feasible.

5. For the general standard-form LP: $\min\{c^T x : \ Ax = b, \ x \geq 0\}$, construct an auxiliary LP and an initial basic feasible solution for it. (Hint: Put s in the objective.)

Chapter 5

Duality and Optimality Conditions

5.1 Dual Linear Program

Consider our standard form linear program (1.4) again, i.e.,

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array}$$

where A is $m \times n$ with $m < n$ and full-rank, the the sizes of all the other quantities follow accordingly. The reduced LP corresponding to (1.4) is

$$\begin{array}{ll} \min\limits_{x_N} & b^T y + s_N^T x_N \\ \text{s.t.} & A_B^{-1} b - A_B^{-1} A_N x_N \geq 0 \\ & x_N \geq 0. \end{array}$$

From this reduced LP, we can make the following observation.

Proposition 3. *If there is a partition (B, N) of the index set $\{1, 2, \cdots, n\}$ such that (i) $A_B^{-1} b \geq 0$; (ii) $s_N = c_N - A_N^T y^* \geq 0$ for $y^* = A_B^{-T} c_B$, then x^* solves the LP where $x_B^* = A_B^{-1} b$ and $x_N^* = 0$. Moreover, the optimal objective value satisfies $c^T x^* = b^T y^*$.*

Therefore, conditions (i)-(ii) are sufficient for optimality. In the nondegenerate case where (i) becomes $A_B^{-1} b > 0$, condition (ii) is clearly necessary for optimality.

Condition (ii) can be rewritten as $c_B - A_B^T y^* = 0$ and $c_N - A_N^T y^* \geq 0$, which implies that y^* satisfies

$$c - A^T y \geq 0 \quad \text{or} \quad A^T y \leq c. \tag{5.1}$$

It turns out that for any y satisfying (5.1) and any feasible x (satisfying $Ax = b$ and $x \geq 0$), the following inequality holds

$$b^T y = (Ax)^T y = x^T (A^T y) \leq x^T c = c^T x,$$

where we have used the facts $Ax = b$, $x \geq 0$ and $A^T y \leq c$. Consequently,

$$\max_y \{b^T y : A^T y \leq c\} \leq \min_x \{c^T x : Ax = b,\ x \geq 0\}, \tag{5.2}$$

as long as both extreme values exist.

The right-hand side is nothing but our linear program (1.4). Appearing on the left-hand side is another linear program, which can also be written as

$$\begin{array}{ll} \max & b^T y \\ \text{s.t.} & A^T y + s = c \\ & s \geq 0 \end{array} \tag{5.3}$$

where a slack variable s is added.

Obviously, this pair of linear programs share the same set of data (A, b, c) and are closely related. We call (1.4) the *primal* LP and (5.3) the *dual* LP. Whenever this primal-dual pair of LPs are both feasible, they satisfy the property in (5.2) which is called *weak duality*. We now formally state the weak duality as follows.

Proposition 4 (Weak Duality)**.** *If x is primal feasible and y is dual feasible, then $b^T y \leq c^T x$. As a result, (5.2) holds true.*

The objective values of the primal and the dual serve as bounds for each other. Once they coincide, then they must have reached their optimal values simultaneously. In fact, optimality can occur only if the primal and the dual objective values coincide. This property is called *strong duality*.

Proposition 5 (Strong Duality)**.** *A primal feasible x^* is optimal for the primal if and only if there is a dual feasible y^* such that $c^T x^* = b^T y^*$. In this case, y^* is optimal for the dual.*

5.2 Optimality Conditions

Strong duality states that the optimality of the primal is inseparable from that of the dual. The pair is one entity as far as optimality is concerned. Now we can state the optimality conditions for the pair as follows.

Proposition 6 (Optimality Conditions)**.** *The points x^* and y^* are optimal for the primal and the dual, respectively, if and only if they together satisfy (1) the primal feasibility: $Ax = b$ and $x \geq 0$; (2) the dual feasibility $A^T y \leq c$; and (3) the closure of the duality gap: $c^T x - b^T y = 0$.*

5.3 How to find a dual?

We have seen that as far as optimality conditions are concerned, a linear program is inseparable with its dual and vice versa. Hence, being able to find a dual for any given LP is crucial. There exist mathematical techniques for finding duals that are applicable not only to linear programming but also to more general problem classes. It is also possible to always convert an LP to one of the standard forms, get the corresponding dual and then simplify. Nonetheless, for linear programming it is much easier to just follow a verified recipe.

For convenience, we will classify constraints into two classes: (1) *sign restrictions* and (2) *main constraints*. A sign restriction on a variable x_i is either $x_i \geq 0$ or $x_i \leq 0$. A variable without any sign restriction is called a free variable. Any constraint that is not a sign restriction is treated as a main constraint which can be an equality or an inequality. The coefficients for all main constraints form an m by n coefficient matrix $A \in \Re^{m \times n}$, i.e., there are n variables and m main constraints.

We consider the following linear program of a general form:

$$\begin{array}{lll} \min \,/\, \max & c^T x & \\ \text{s.t.} & a_i^T x \begin{cases} \geq b_i \\ = b_i \\ \leq b_i \end{cases} & i = 1, 2, \cdots, m, \\ & x_j \begin{cases} \geq 0 \\ \text{free} \\ \leq 0 \end{cases} & j = 1, 2, \cdots, n, \end{array} \tag{5.4}$$

where a_i^T is the i-th row of A. In this LP, the objective can be either minimization or maximization, and each constraint and each variable can take, respectively, one of the three given forms. The recipe for finding a dual for thus LP is given in the following chart, where if an LP is a minimization problem, we go from the left column to the right column to find its dual; otherwise, we go from right to left.

minimization	$\Leftrightarrow$	maximization
rhs/obj vectors	$\Leftrightarrow$	obj/rhs vectors
[coefficients]	$\Leftrightarrow$	$[\text{coefficients}]^T$
#variables	$\Leftrightarrow$	#constraints
#constraints	$\Leftrightarrow$	#variables
variable ≥ 0	$\Leftrightarrow$	constraint $\leq$
variable free	$\Leftrightarrow$	constraint $=$
variable ≤ 0	$\Leftrightarrow$	constraint $\geq$
constraint $\geq$	$\Leftrightarrow$	variable ≥ 0
constraint $=$	$\Leftrightarrow$	variable free
constraint $\leq$	$\Leftrightarrow$	variable ≤ 0

Figure 5.1: Chart for Finding a Dual

The chart says that to find a dual for a given primal, we flip the optimization goals, switch the objective and right-hand side vectors, and take a transpose of the coefficient matrix for the main constraints. This way, the numbers of (main) *constraints and variables in the dual* are always equal to, respectively, the numbers of *variables and constraints in the primal*, and vice versa. In addition, the types of the constraints and variables follow the rules stipulated in the chart. The chart also reveals one-to-one relationships between the variables in the primal and the constraints in the dual and vice versa.

Example 1. *Find the dual of ...*

Now let us use the chart to find the dual of the general LP 5.4. We partition the index set $\{1, 2, \cdots, m\}$ into three subsets:

1. G corresponding to constraints of the type $a_i^T x \geq b_i$;

2. E corresponding to constraints of the type $a_i^T x = b_i$;

3. L corresponding to constraints of the type $a_i^T x \leq b_i$.

Similarly, we partition the index set $\{1, 2, \cdots, n\}$ into three subsets:

1. P corresponding to variables of the type $x_i \geq 0$;

2. F corresponding to free variables;

3. M corresponding to variables of the type $x_i \leq 0$.

Now we partition the rows of A according to the partition (G, E, L), and the columns of A according to the partition (P, F, M); namely,

$$A = \begin{bmatrix} A^G \\ A^E \\ A^L \end{bmatrix} = \begin{bmatrix} A_P & A_F & A_M \end{bmatrix}. \tag{5.5}$$

With these partitions, we can write (5.4) into the following form:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & A^G x \geq b_G \\ & A^E x = b_E \\ & A^L x \leq b_L \\ & x_P \geq 0 \\ & x_F \text{ free} \\ & x_M \leq 0. \end{array} \tag{5.6}$$

The dual readily follows from the chart in Figure 5.1.

$$\begin{array}{ll} \max & b^T y \\ \text{s.t.} & A_P^T y \leq c_P \\ & A_F^T y = c_F \\ & A_M^T y \geq c_M \\ & y_G \geq 0 \\ & y_E \text{ free} \\ & y_L \leq 0 \end{array} \tag{5.7}$$

The complementarity conditions for this pair of primal-dual LPs are

$$\begin{pmatrix} (A^G x - b_G) \circ y_G \\ (A^L x - b_L) \circ y_L \\ x_P \circ (c_P - A_P^T y) \\ x_M \circ (c_M - A_M^T y) \end{pmatrix} = 0. \tag{5.8}$$

It is possible that some of these subsets are empty, in which case the corresponding constraints/variables are in non-existence. For example, if $M = \emptyset$, then the variable x_M and its dual constraint $A_M^T y \geq c_M$ do not exist.

5.4 Exercises

1. Use the weak duality to show that if the primal is unbounded then the dual must be infeasible and vice versa.

2. Prove that for feasible x and y, the duality gap satisfies

$$c^T x - b^T y = s^T x$$

   where $s = c - A^T y \geq 0$ is the dual slack.

3. At each iteration of the primal simplex method, out of the three optimality conditions, which is satisfied and which is not? Explain.

Chapter 6

Primal-Dual Interior-Point Methods

6.1 Introduction

Consider the following primal linear program:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \tag{6.1}$$

where $c, x \in \Re^n$, $b \in \Re^m$ and $A \in \Re^{m \times n} (m < n)$.

Its dual linear program is:

$$\begin{array}{ll} \max & b^T y \\ \text{s.t.} & A^T y + z = c \\ & z \geq 0 \end{array} \tag{6.2}$$

where $z \in \Re^n$ is the vector of dual slack variables.

A primal-dual method is one that treats the primal and the dual equally and tries to solve both at the same time.

For feasible x and (y, z), the duality gap is

$$\begin{array}{lcll} & c^T x - b^T y & & \\ = & x^T c - x^T A^T y & & (Ax = b) \\ = & x^T (c - A^T y) & & \\ = & x^T z & & (c - A^T y = z) \\ \geq & 0 & & (x, z \geq 0). \end{array}$$

Duality gap is closed at optimality:

$$c^T x^* - b^T y^* = (x^*)^T z^* = 0.$$

The complementarity conditions are:

$$x_i^* z_i^* = 0, \quad i = 1, 2, \ldots, n. \tag{6.3}$$

6.2 A Primal-Dual Method

Consider $F : \mathbf{R}^{2n+m} \to \mathbf{R}^{2n+m}$,

$$F(x, y, z) = \begin{pmatrix} A^T y + z - c \\ Ax - b \\ x_1 z_1 \\ \vdots \\ x_n z_n \end{pmatrix} = 0. \tag{6.4}$$

This is a linear-quadratic square system.

The optimality conditions for the linear program is

$$F(x, y, z) = 0, \quad (x, z) \geq 0. \tag{6.5}$$

It is also called the KKT (Karush-Kohn-Tucker) system for the LP.

The Jacobian matrix of $F(x, y, z)$ is

$$F'(x, y, z) = \begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ Z & 0 & X \end{bmatrix}, \tag{6.6}$$

where $X = diag(x)$ and $Z = diag(z)$. $F'(x, y, z)$ is nonsingular for $(x, z) > 0$ if A has full rank.

Even though (6.5) is just mildly nonlinear, one should not expect that the pure Newton's method would solve the system because of the presence of nonnegativity constraints on x and z.

A main idea for interior-point methods is that one starts from an initial point with $x > 0$ and $z > 0$ and forces all subsequent iterates for x and z to remain positive, i.e., to stay in the interior of the nonnegative orthant. One

can always enforce this interiority requirement by choosing appropriate step parameter in a dampled Newton setting.

It is important to note the necessity of keeping the nonnegative variables x and z strictly positive at every iteration instead of just nonnegative. Consider a scalar complementarity condition $xz = 0$, $x, z \in \Re$. The Newton equation (i.e., the linearized equation) for $xz = 0$ at a given point (x, z) is $xdz + zdx = -xz$. If one variable, say z, is zero, and the other is nonzero, then the Newton equation becomes $xdz = 0$, leading to a zero update $dz = 0$. Consequently, z will remain zero all the time once it becomes zero. This is fatal because an algorithm will never be able to recover once it mistakenly sets a variable to zero. Therefore, it is critical to keep the nonnegative variables strictly positive at every iteration, not just nonnegative.

Even keeping nonnegative variables strictly positive, one would still expect difficulty in recovering from a situation where a variable is adversely set to too small a value. To decrease the chances of such mistakes at early stages, it would be desirable that all the complementarity pairs converge to zero at the same pace, namely $x_i^k z_i^k = \mu^k \to 0$ as $k \to \infty$ for every index i. Towards this goal (and for other theoretic considerations as well), one can perturb each complementarity condition $x_i z_i = 0$ into $x_i z_i - \mu = 0$ and let μ go to zero in a controlled fashion. This idea, incorporated into the damped Newton method, leads to the following primal-dual interior-point algorithm. Recall that it is called primal-dual because it treats the primal and dual variables equally.

Like in any iterative method, we need some initial point to start. In particular, we need positive x and z in our initial point. A simple, and not so bad, initial point consists of $x = z = n * e$, where e is the vector of all ones, and $y = 0$.

Algorithm 1 (A Prima-Dual Interior Point Algorithm)**.**

Choose initial point $(x > 0, y, z > 0)$.

WHILE *"stopping criteria" not satisfied,* **DO**

Step 1 *Choose* $\sigma \in (0, 1)$ *and set* $\mu = \sigma x^T z / n$.

Step 2 *Solve the linear system for* (dx, dy, dz)

$$F'(x, y, z) \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix} = -F(x, y, z) + \begin{pmatrix} 0 \\ 0 \\ \mu e \end{pmatrix}.$$

Step 3 *Compute the primal and dual step-length parameters:*

$$\alpha_p = -1/\min([X^{-1}dx; -1]), \quad \alpha_d = -1/\min([Z^{-1}dz; -1]).$$

Step 4 *Choose* $\tau \in (0,1)$ *and update*

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longleftarrow \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \tau \begin{bmatrix} \alpha_p\, dx \\ \alpha_d\, dy \\ \alpha_d\, dz \end{bmatrix}.$$

END

We now make the following observations on the algorithm. There are two basic parameters in the algorithm: σ in Step 2 and τ in Step 4. Their sensible choices are of vital importance to both the theoretical analysis and the practical performance of the algorithm. For starters, however, $\sigma = .2$ and $\tau = .99$, say, will work fine most of the time.

In Step 3, each minimum is taken as the smallest element of the corresponding $n+1$-dimensional vector. The scalars α_p and α_d are the largest step lengths in $[0,1]$ such that $x + \alpha_p\, dx \geq 0$ and $z + \alpha_d\, dz \geq 0$, respectively. (The formulas are nothing but compact expressions of the ratio tests performed in simplex methods.)

In Step 4, tha parameter τ is chosen to step back a little from the boundary of the positive orthant where the next iterate might land had a full step α_p or α_d been taken. By choosing $\tau \in (0,1)$, the posititivity for the variables x and z will always hold. If the algorithm converges, it does converge most of the time even for rather naive choices of parameters, then some components of x and z will go to zero in order to satisfy the complementary slackness conditions $Xz = 0$. Namely, the iterates will eventually approach the boundary of the positive orthant.

The step computed in Step 2 is the Newton step for the so-called perturbed KKT equation

$$F_\mu(x,y,z) \equiv F(x,y,z) - \mu_k \begin{pmatrix} 0 \\ 0 \\ e \end{pmatrix} = 0,$$

namely, the complementarity equation $Xz = 0$ is perturbed into $Xz = \mu e$. This perturbation tends to keep all components of Xz equal and, hopefully, forces them to converge to zero at more or less the similar pace.

Overall, we may view the algorithm as a **perturbed and damped Newton** method.

Finally, we state a reasonable stopping criterion

$$\frac{\|Ax-b\|}{1+\|b\|}+\frac{\|A^Ty+z-c\|}{1+\|c\|}+\frac{|c^Tx-b^Ty|}{1+|b^Ty|}\leq \text{tolerance},$$

where we use the sum of the relative errors in the three blocks of equations to measure the total error.

6.3 Solving the Linear System

The main computation in Algorithm 1 at each iteration is to solve to the linear system in Step 2 involving the coefficient matrix $F'(x,y,z)$ which is $2n+m$ by $2n+m$ but very sparse (see (6.6)). This system can be written as

$$\begin{aligned} A^Td\!y+d\!z &= r_d, &(6.7)\\ Ad\!x &= r_p, &(6.8)\\ Zd\!x+Xd\!z &= r_c, &(6.9)\end{aligned}$$

where $X=diag(x)$ and $Z=diag(z)$ are diagonal matrices and

$$r_d=c-A^Ty-z,\ \ r_p=b-Ax,\ \ r_c=\mu e-Xz.$$

Multiplying (6.7) by X and subtract (6.9) from it, we obtain

$$XA^Td\!y-Zd\!x=Xr_d-r_c.$$

Multiplying the above equation by AZ^{-1} and invoking (6.8), we arrive at a square system for $d\!y$ in (6.10). After $d\!y$ is computed, we can then easily compute $d\!z$ and $d\!x$ by back substitution using (6.7) and (6.9). Hence a procedure for solving the linear system is as follows:

$$\begin{aligned} (AZ^{-1}XA^T)\,d\!y &= AZ^{-1}(Xr_d-r_c)+r_p, &(6.10)\\ d\!z &= r_d-A^Td\!y, &(6.11)\\ d\!x &= Z^{-1}(r_c-Xd\!z), &(6.12)\end{aligned}$$

Now the major computation becomes solving (6.10) which is an m by m system with a positive definite coefficient matrix provided that A has full rank. (For large-scale problems, (6.10) is usually solved by sparse Cholesky factorization.)

6.4 Convergence Considerations

To guarantee in theory that the primal-dual interior-point algorithm, Algorithm 1, always converges to a solution, one needs to use rather sophisticated schemes to choose the step parameter τ. On the other hand, even with simple-minded choices such as $\tau = 0.9$, the algorithm does converge most of the time for problems that are not particularly difficult. Here we provide some indications on why the algorithm usually works even for choices such as $\tau = 0.9$.

At each iteration, given the current iterate (x, y, z) the update direction (dx, dy, dz) satisfies the equation

$$Adx = -(Ax - b), \quad A^T dy + dz = -(A^T y + z - c),$$

and the next iterate (x^+, y^+, z^+) is calculated as

$$\begin{bmatrix} x^+ \\ y^+ \\ z^+ \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \alpha \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}, \tag{6.13}$$

for some $\alpha \in (0, 1]$. Here for simplicity we are using the same step parameter for both the primal and the dual variables. By direct substitution we have

$$Ax^+ - b = (1 - \alpha)(Ax - b), \quad A^T y^+ + z^+ - c = (1 - \alpha)(A^T y + z - c). \tag{6.14}$$

Since $1 - \alpha \in [0, 1)$, we have the following relationships

$$\begin{aligned} \|Ax^+ - b\| &= (1 - \alpha)\|Ax - b\| < \|Ax - b\|, \\ \|A^T y^+ + z^+ - c\| &= (1 - \alpha)\|A^T y + z - c\| < \|A^T y + z - c\|. \end{aligned}$$

That is, we can improve the primal and dual feasibility at each iteration. However, such improvements do not imply that the iterates necessarily converge to a feasible (primal and dual) point (the improvements could shrink to zero as $\alpha \to 0$). Moreover, it is not clear that we can also improve the complementary slackness at each iteration.

To simplify the matter, let us assume that the initial point (x^0, y^0, z^0) is strictly feasible, i.e., it satisfies

$$Ax^0 = b, \quad A^T y^0 + z^0 = c, \quad x^0, z^0 > 0.$$

It follows from the recursive relationships (6.14) that the feasibility of (x^0, y^0, z^0) implies the feasibility of (x^k, y^k, z^k) for $k = 1, 2, \cdots$. In this case,

$$F(x^k, y^k, z^k) = \begin{bmatrix} 0 \\ 0 \\ X^k z^k \end{bmatrix}$$

and

$$\|F(x^k, y^k, z^k)\|_1 = \sum_{i=1}^{n} x_i^k z_i^k = (x^k)^T (z^k).$$

Hence, it suffices to consider driving the duality gap $x^T z$ to zero.

Now consider the duality gap at (x^+, y^+, z^+), where (x^+, y^+, z^+) is obtained from the update formula (6.13). Hence,

$$\begin{aligned}
(x^+)^T(z^+) &= (x + \alpha dx)^T (z + \alpha dz) \\
&= x^T z + \alpha(x^T dz + z^T dx) + \alpha^2 (dx)^T (dz) \\
&= x^T z + \alpha(x^T dz + z^T dx) && [(dx)^T (dz) = 0] \\
&= x^T z + \alpha e^T (\mu e - Xz) && [Xdz + Zdx = \mu e - Xz] \\
&= x^T z + \alpha(\sigma - 1) x^T z && [\mu = \sigma x^T z / n] \\
&= [1 - \alpha(1 - \sigma)] x^T z.
\end{aligned}$$

We leave the fact $(dx)^T (dz) = 0$ as an exercise. Since by our choice $\alpha \in (0, 1]$ and $\sigma \in (0, 1)$, we have the recursive relationship

$$(x^+)^T (z^+) = [1 - \alpha(1 - \sigma)] x^T z < x^T z,$$

that is, the algorithm always decreases the duality after each iteration. More precisely,

$$\begin{aligned}
(x^k)^T (z^k) &= [1 - \alpha^{k-1}(1 - \sigma)](x^{k-1})^T (z^{k-1}) = \cdots \\
&= \left(\prod_{j=0}^{k-1} [1 - \alpha^j (1 - \sigma)] \right) (x^0)^T (z^0).
\end{aligned}$$

If the step parameters α^j are bounded below by a nonzero shortest step $\underline{\alpha}$, i.e., $\alpha^j \geq \underline{\alpha}$, then for all j

$$1 - \alpha^j (1 - \sigma) \leq 1 - \underline{\alpha}(1 - \sigma) < 1.$$

Consequently,

$$0 \leq (x^k)^T(z^k) \leq [1 - \underline{\alpha}(1-\sigma)]^k (x^0)^T(z^0) \to 0 \text{ as } k \to \infty,$$

which implies the convergence of the duality gap to zero, and hence the convergence of the algorithm. Indeed, we observe that the step parameter α rarely shrinks to zero. That is one of the reasons the algorithm usually works even for naive choices of step parameter.

Appendix A

Introduction to Newton's Method

One of the fundamental computational problems in science and engineering is to solve nonlinear systems of equations of multiple variables. A general nonlinear, square system of equations can be expressed as follows:

$$F(x) = 0, \tag{A.1}$$

where $x \in \Re^n$ and $F : \Re^n \to \Re^n$; namely, there are n variables and n equations. We will assume that not all of the n equations are linear; otherwise (A.1) becomes a linear system which is much easier to solve. In a more detailed notation, we have

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad F(x) = \begin{bmatrix} F_1(x) \\ F_2(x) \\ \vdots \\ F_n(x) \end{bmatrix} \equiv \begin{bmatrix} F_1(x_1, x_2, \cdots, x_n) \\ F_2(x_1, x_2, \cdots, x_n) \\ \vdots \\ F_n(x_1, x_2, \cdots, x_n) \end{bmatrix},$$

where we use the subscripts to denote components of vectors. For example, a simple nonlinear system with $n = 2$ is

$$F(x) = \begin{bmatrix} e^{2x_1} - e^{x_2} \\ x_1^2 + x_2^2 - 5 \end{bmatrix} = 0. \tag{A.2}$$

At a given point $x^c \in \Re^n$ (note that we use a superscript to denote a particular vector), considered as the current approximate solution to the

equation $F(x) = 0$, we approximate $F(x)$ by its first-order Taylor expansion:

$$F(x) \approx L_c(x) \equiv F(x^c) + F'(x^c)(x - x^c),$$

where $F'(x)$ is an $n \times n$ matrix called the Jacobian matrix of $F(x)$, which is the first derivative of $F(x)$. The entries of the Jacobian matrix $F'(x)$ consist of all the first-order partial derivatives of the component functions $F_i(x)$'s, arranged in an ordered fashion, i.e.,

$$[F'(x)]_{ij} = \frac{\partial F_i(x)}{\partial x_j}, \quad 1 \le i, j \le n.$$

In other words, the i-th row of $F'(x)$ is the gradient of $F_i(x)$. For example, when $n = 2$, $F'(x)$ is a 2×2 matrix,

$$F'(x) = \begin{bmatrix} \frac{\partial F_1(x)}{\partial x_1} & \frac{\partial F_1(x)}{\partial x_2} \\ \frac{\partial F_2(x)}{\partial x_1} & \frac{\partial F_2(x)}{\partial x_2} \end{bmatrix}.$$

In particular, the Jacobian of the function $F(x)$ in (A.2) is

$$F'(x) = \begin{bmatrix} 2e^{2x_1} & -e^{x_2} \\ 2x_1 & 2x_2 \end{bmatrix}. \tag{A.3}$$

We note that since x^c is a fixed point, $F(x^c)$ is a fixed n-vector and $F'(x^c)$ a fixed $n \times n$ matrix. Hence $L_c(x)$ is a linear function of x and called the **linearization** of $F(x)$ at the point x^c.

Solving the square linear system of equations $L_c(x) = 0$, we obtain a point x^+ as a new approximate solution to the equation $F(x) = 0$. It is easy to verify that the solution to $L_c(x) = 0$ is

$$x^+ = x^c - [F'(x^c)]^{-1} F(x^c),$$

provided that $F'(x^c)$ is nonsingular. We leave the verification to the reader.

In the case of $n = 1$, both $F(x)$ and its derivative $F'(x)$ are scalar functions. The linearization of $F(x)$ at x^c,

$$y = L_c(x) \equiv F(x^c) + F'(x^c)(x - x^c), \tag{A.4}$$

is the tangent line of the curve $y = F(x)$ at the point $x = x^c$. The new approximate solution,

$$x^+ = x^c - \frac{F(x^c)}{F'(x^c)}, \tag{A.5}$$

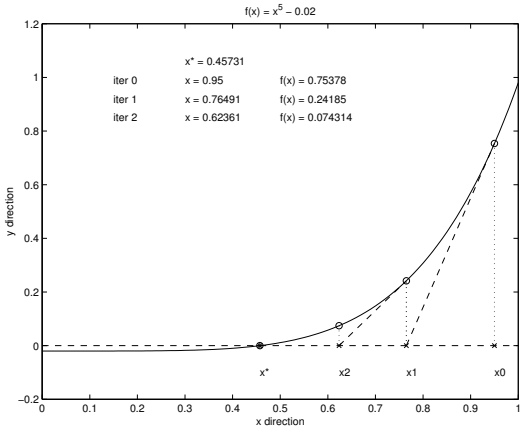

Figure A.1: Newton's method: one-dimensional case

is the intersection of the tangent line $y = L_c(x)$ with the x-axis $y = 0$, see Figure A.

In the general case of n variables and n equations, **Newton's method** computes the iteration sequence $\{x^k\} \subset \Re^n$ by the recursive formula:

$$x^{k+1} = x^k - [F'(x^k)]^{-1} F(x^k),$$

starting from a given initial point x^0. If x^0 is sufficiently close to a solution x^* satisfying $F(x^*) = 0$, then under proper conditions, we can expect that the sequence converges to the solution x^*; namely,

$$\lim_{k \to \infty} x^k = x^*.$$

The main idea behind Newton's method can be described as follows. Instead of trying to directly solve the hard problem, the nonlinear system, one solves a sequence of easy problems, the linear systems, to obtain a sequence of approximate solutions that will hopefully get closer and closer to the true solution.

Introducing a step parameter $\alpha^k \in (0, 1]$ at each iteration, we obtain the so-called **damped Newton's method**:

$$x^{k+1} = x^k - \alpha^k [F'(x^k)]^{-1} F(x^k).$$

It is known that properly chosen step parameters can help enhance the convergence of Newton's method, which is corresponding to the case where $\alpha^k = 1$ for all k.

A.1 An Example

Let us consider solving the simple nonlinear system defined in (A.2), namely,

$$F(x) = \begin{bmatrix} e^{2x_1} - e^{x_2} \\ x_1^2 + x_2^2 - 5 \end{bmatrix} = 0.$$

It has two equations and two variables. Incidently, it also has two solutions: $x = [1; 2]$ and $x = [-1; -2]$. The Jacobian matrix of this function is given in (A.3).

The following Matlab functions implement Newton's method for solving the nonlinear system (A.2).

```
% main program
function x = newton(x0);
% input  x0 -- the starting point
% output  x -- hopefully a solution

% make x0 a column vector if not already
if size(x0,1) < size(x0,2) x0 = x0'; end;
x = x0;

iter = 1;                % iteration counter
tol = 1.e-8;             % error tolerance
maxiter = 50;            % maximum iteration

while iter <= maxiter    % begin loop
   F = func(x);          % evaluate F(x)
   resnrm = norm(F);     % residual norm

% print out information at each iteration
   fprintf('iter %2i  residual-norm = %8.2e',iter,resnrm);
   fprintf('  x = [%7.4f %7.4f]\n',x);

% stopping criterion
   if resnrm < tol disp('Converged!'); break; end

   J = jacobian(x);      % evaluate Jacobian
   x = x - J\F;          % update iterate
   iter = iter + 1;      % increment counter
```

```
end                          % end loop

% function evaluation
function F = func(x)
F = [exp(2*x(1))-exp(x(2));
       x(1)^2+x(2)^2-5];

% Jacobian evaluation
function J = jacobian(x)
J = [2*exp(2*x(1)) -exp(x(2));
           2*x(1)     2*x(2)];
```

These Matlab functions are saved into a file called `newton1.m` and can be invoked with an initial point $x^0 = [1.5\ 1.5]$ by typing: `newton1([1.5 1.5])` in Matlab. The output from Matlab is as follows.

```
newton1([1.5 1.5]);
iter  1  residual-norm = 1.56e+001  x = [ 1.5000  1.5000]
iter  2  residual-norm = 2.96e+000  x = [ 1.1673  1.9994]
iter  3  residual-norm = 3.93e-001  x = [ 1.0228  1.9937]
iter  4  residual-norm = 7.51e-003  x = [ 1.0005  1.9999]
iter  5  residual-norm = 3.09e-006  x = [ 1.0000  2.0000]
iter  6  residual-norm = 5.24e-013  x = [ 1.0000  2.0000]
Converged!
```

For this initial point, which is fairly close to the solution $x^* = [1\ 2]$, Newton's method takes six iterations to achieve the required accuracy.

A.2 Exercises

1. Justify the following claims:
   (a) $y = \ell_c(x)$ in (A.4) is the tangent line of the curve $y = F(x)$ at the point $x = x^c$.
   (b) x^+ in (A.5) is the intersection of the tangent line with the x-axis.

2. Run `newton1.m` with initial guesses

$$x_0 = [1\ 2] + a * [1\ 1], \quad a = 2, 4, 6, 8, 10$$

and answer the following questions.
(a) Does Newton's method always converge?
(b) When Newton's method converges, does it always converge to the solution closest to the initial point?
(c) When Newton's method converges, describe how the residual norm $\|F(x^k)\|$ decreases at each iteration towards the end.

3. Write an M-file `newton2.m`, by modifying the functions in `newton1.m`, to solve the following nonlinear system:

$$F(x) = \begin{bmatrix} x_1 + x_2 - 1 \\ x_3 - x_4 - 1 \\ x_3 - x_5 - 2 \\ x_1 x_4 \\ x_2 x_5 \end{bmatrix} = 0$$

with the starting points $x = [1\ 2\ 3\ 2\ 1]$ and $x = [10\ .1\ 10\ .1\ 10]$. (Observe that for the second starting point, the obtained solution is NOT nonnegative.)