Numerical Experimentations With Input Optimization

by Marc P. Brunet

Department of Mathematics and Statistics
McGill University, Montreal
February, 1989

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Science

Acknowledgements

I am extremely grateful to Professor S. Zlobec for having guided and counselled me not only throughout my research for this thesis, but also throughout my entire undergraduate and graduate programs. My guide and counsellor for "the way things are done in a thesis" was my father, Professor J.P. Brunet—thank you! I wish to thank Carole Nahum for her input (!) into the numerical method of Chapter 4, namely determining the scalar α. I also wish to thank Diane C. Brunet, proof-reader *par excellence*, for her time and effort.

On those occasions when time kept flying and I didn't, it was my wife, Violaine, who encouraged me to keep at it. *Je t'en remercie de tout mon cœur!*

Abstract

This thesis demonstrates that the abstract topological and analytical notions and results of input optimization can be successfully used in solving real-life problems in management and engineering. In particular, we use a marginal value formula to determine improvable stable perturbations of arbitrary parameters in mathematical programming models of economic/engineering systems. Globally stable paths are determined by a new kind of feasible directions method and an optimal realization of mathematical models is verified by recently introduced optimality conditions. The problems used in numerical experimentations and in demonstrating input optimization include two case studies of real companies (a textile mill and a coffee company), a nonlinear engineering program that ocurred in a heat exchanger design problem, and an unconstrained nonlinear optimization problem. Computer programs, used in solving the case studies, are included in appendices.

Résumé

Cette thèse démontre que des notions abstraites de topologie et d'analyse et les résultats de l'"input optimization" peuvent servir à résoudre des problèmes concrets en génie et en gestion des ressources. En particulier, nous utilisons une formule de valeur marginale pour indiquer les perturbations stables de paramètres arbitraires dans des modèles mathématiques de systèmes économiques et d'ingénierie. Des trajets globalement stables sont déterminés par une nouvelle sorte de méthode des directions faisables et une réalisation optimale de modèles mathématiques est prouvée à l'aide de conditions d'optimum récemment présentées. L'étude est faite de deux cas de compagnies américaines (une impliquée dans le commerce du textile et l'autre dans la production du café), d'un problème d'optimisation non-linéaire en génie (concernant un transfert de chaleur) et d'un problème d'optimisation non-linéaire sans contraintes. Deux programmes informatisés, utilisés dans la solution des cas, sont inclus en appendice.

Table of Contents

Chapter 1

Introduction

The main objective of this research is to explore whether and how some of the abstract results from input optimization can be used in real-life case studies.

In each case, we study a company's production schedule and the factors which influence it (cost, quality control, technical capacities, etc.). This allows us to model the production schedule with a linear model. Optimizing this model (with fixed parameters) gives an optimal production policy for the next time period (one week or one month, depending on the case). An objective of this thesis is to show that from this stage it is possible, in practice, to change or perturb globally certain parameters in the model in such a way as to get an even better production schedule (i.e., greater profit or lesser cost) while retaining continuity of the output. By output we mean the feasible production policies, the optimal production policies and the optimal value of the problem (which is the overall cost or profit). These perturbations can lead to an "optimal realization" of the initial model. This is a state of the model in which the value of the objective function cannot be improved by feasible and stable

perturbations of the parameters (all definitions are given below in Chapter 2).

Finding a stable (i.e., output continuous) path that governs the system from its present (initial) state to its optimal realization, is the main subject of "input optimization," see, e.g., [4], [19], [20], [21], [22], [23], [24], [25], [26] and [27].

The originality and contribution of this thesis lies in the application of input optimization theory to "real-life" cases.

In Chapters 2 and 3, we recollect some basic notions and results of input optimization. Chapter 4 contains a description of the numerical optimization method used to study both cases. This method is based on the results of the second and third chapters. As well as the general method being described there, the particular adaptations to each case are mentioned.

Chapters 5 and 6 present two case studies adapted from [17], with the results of the numerical optimization method described in Chapter 4. Case study I is the Shenandoah Valley Textile Mill case—a fictitious name—which poses the problem of improving the forthcoming week's profits on seven different fabrics by changing the efficiency of three different stages of production. The results for several possible situations are given, i.e., for different assumptions regarding the company and for variations of the numerical algorithm.

The second case study is the Evangeline Coffee Company—again, a fictitious name. The original case study was an unnecessarily large problem for our purpose. The objective here is to show that input optimization theory can be applied in

practice, and not to present new ways of tackling large problems. This company imports, blends and roasts green coffees (coffees that have not been roasted are referred to as "green coffees") to produce different quality coffee blends. The goal is to vary certain characteristics of the green coffees which affect the total cost of the blends. The results given are, again, from different initial situations.

The cases of Chapters 5 and 6 translate into linear programming models to which input optimization theory is applied. In Chapter 7, we give some non-linear examples to which this theory is also well-suited. The second section of Chapter 7 discusses the importance of stability in input optimization with respect to two models.

The appendices contain computer listings and explanations regarding their use and implementation. Appendix A contains a listing for performing the revised simplex method. It was written for use in the program listed in Appendix B. The latter program is the numerical algorithm described in Chapter 4 which optimizes a linear programming model (i.e., the optimal value function with respect to stable perturbations of parameters).

Chapter 2

General Concepts

The results put forth in this thesis are all based on input optimization theory. We therefore will give here the necessary definitions and preamble which introduce this theory, presented in part in Chapter 3.

We study the general model of the form

$$(P,\theta) \qquad \begin{array}{ll} \underset{(x)}{Min} & f^0(x,\theta) \\ s.t. & f^i(x,\theta) \leq 0, \quad i \in \mathcal{P} \stackrel{\text{def}}{=} \{1,\ldots,m\} \end{array}$$

where $\theta \in I \subset R^p$ and I is a convex set. The functions $f^j : R^n \times R^p \to R$, $j \in \{0\} \cup \mathcal{P}$ are assumed to be continuous and convex in x, i.e., $f^j(\cdot,\theta) : R^n \to R$, $j \in \{0\} \cup \mathcal{P}$, is convex for every $\theta \in R^p$. Such a model is known as a *convex* model. Furthermore, if the functions $f^j(x,\cdot) : R^p \to R$, $j \in \{0\} \cup \mathcal{P}$, are convex for every $x \in R^n$, then we have a *bi-convex model*.

We recall that a function $f : R^n \to R$ is *convex* if for all x and y in R^n and all

λ such that $0 \leq \lambda \leq 1$ we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y).$$

If f is differentiable then it is convex if, and only if

$$\nabla f(x)(y-x) \leq f(y) - f(x)$$

for all $x \in R^n$, $y \in R^n$. This is known as the *gradient inequality.* A set $S \subset R^n$ is convex if, for all λ such that $0 \leq \lambda \leq 1$,

$$x \in S,\ y \in S \Longrightarrow \lambda x + (1-\lambda)y \in S.$$

In the model (P, θ), the vector $\theta \in R^p$ is known as the parameter vector, and also as the input in "input optimization." The vector $x \in R^n$ is the vector of decision variables. Note that for fixed $\theta \in R^p$, (P, θ) is an ordinary convex *program.*

With respect to the model (P, θ) and for a fixed θ, we define the following:

- $F(\theta) = \{x \in R^n : f^i(x, \theta) \leq 0, \quad i \in \mathcal{P}\}$ is the *feasible set.*
- $\tilde{F}(\theta)$ is the *set of all optimal solutions* $\tilde{x}(\theta)$.
- $\tilde{f}(\theta) = f^0(\tilde{x}(\theta), \theta)$ is the *optimal value* of the model for a particular θ.
- $\mathcal{P}^=(\theta) = \{i \in \mathcal{P} : x \in F(\theta) \Longrightarrow f^i(x, \theta) = 0\}$ is the *minimal index set of active constraints* (see Ben-Israel, Ben-Tal, and Zlobec [3], [4]).
- $\mathcal{P}(\tilde{x}(\theta^*)) = \{i \in \mathcal{P} : f^i(\tilde{x}(\theta^*), \theta^*) = 0\}$ is the set of active constraints for (P, θ^*).

- $\mathcal{P}^{<}(\theta) = \mathcal{P} \setminus \mathcal{P}^{=}(\theta)$. This definition is introduced to simplify notation.
- $F^{=}(\theta) = \{x \in R^n : f^i(x,\theta) = 0, \ \ i \in \mathcal{P}^{=}(\theta)\}$
- $L^{<}(x,u;\theta) = f^0(x,\theta) + \sum_{i\in\mathcal{P}^{<}(\theta)} u_i f^i(x,\theta)$ is the *restricted Lagrangian*.
- $L^{\leq}_{*}(x,u;\theta) = f^0(x,\theta) + \sum_{i\in\mathcal{P}^{<}(\theta^*)} u_i f^i(x,\theta)$ is the restricted Lagrangian at a fixed $\theta = \theta^*$.

In a convex program, for fixed $\theta \in R^n$, we say that *Slater's condition* holds if

$$\text{there exists } \hat{x} \in R^n \text{ such that } f^i(\hat{x},\theta) < 0, \ \text{ for all } i \in \mathcal{P}.$$

Note that under Slater's condition, $\mathcal{P}^{=}(\theta) = \emptyset$ and therefore $F^{=}(\theta) = R^n$.

For fixed $\theta \in R^p$, $F(\theta)$ is a set in R^n (possibly empty). We can therefore define the mapping $F : \theta \to F(\theta)$ which is a point-to-set mapping. Continuity of the output refers to continuity of the triple $\{F(\theta), \tilde{F}(\theta), \tilde{f}(\theta)\}$, so it includes continuity of the mapping F. We say that a general point-to-set mapping $\Gamma : Z \to X$ is *continuous at* $\theta^* \in Z$, in the sense of Hogan (see [15]), if it is both open and closed. Such a mapping is *closed at* $\theta^* \in Z$, if given any sequence $\theta^k \to \theta^*$ and $x^k \in \Gamma(\theta^k)$ such that $x^k \to x^*$, it follows that $x^* \in \Gamma(\theta^*)$. The point-to-set mapping Γ is said to be *open at* $\theta^* \in Z$ if, given any sequence $\theta^k \to \theta^*$, and $x^* \in \Gamma(\theta^*)$, there exists a value m and a sequence $\{x^k\} \subset X$, such that $x^k \in \Gamma(\theta^k)$ for each $k > m$ and $x^k \to x^*$. We say that the mapping Γ is open, closed, or continuous on Z if it has the same property at every $\theta \in Z$.

Given that in (P,θ), the functions f^i, $i \in \mathcal{P}$, are continuous, it follows immedi-

ately that $F : \theta \rightarrow F(\theta)$ is closed. Therefore, for the mapping to be continuous, we need only verify that it is open. This is equivalent to the mapping F being lower semi-continuous in the sense of Berge (see [5]). The point-to-set mapping $\Gamma : Z \rightarrow X$ is *lower semi-continuous at* $\theta^* \in Z$ if for all open sets $\mathcal{A}$ such that $\Gamma(\theta^*) \cap \mathcal{A} \neq \emptyset$, there exists a neighbourhood $N(\theta^*)$ such that $\Gamma(\theta) \cap \mathcal{A} \neq \emptyset$ for all $\theta \in N(\theta^*)$.

One of the distinguishing characteristics of input optimization theory is that it requires continuity of the output triple $\{F(\theta), \tilde{F}(\theta), \tilde{f}(\theta)\}$. This notion of continuity is incorporated in the definition of a "stable model," which will be given in the next chapter. An objective function $f^0(x, \theta)$ in the convex model (P, θ) is said to be *realistic* at $\theta^* \in I$ if $\tilde{F}(\theta^*) \neq \emptyset$ and bounded.

These concepts will be used in the next chapter to formalise input optimization and state several of its theorems.

Chapter 3

Some Results from Input Optimization

Input optimization is the optimization of a *model* rather than a particular program. Recall the convex model (P, θ) given in Chapter 2. For a fixed vector $\theta \in R^p$, we have the "usual" convex program (C).

$$(C) \qquad \begin{aligned} Min \quad & f^0(x) \\ s.t. \quad & f^i(x) \leq 0, \quad i \in \mathcal{P} = \{1, \ldots, m\}. \end{aligned}$$

Optimizing the objective function with respect to the constraints yields an optimal solution $\tilde{x}$.

The difference with input optimization is that the objective function and constraints are not fixed functions only of x but also of an input parameter θ. For each fixed θ we get an optimal solution $\tilde{x}(\theta)$ (note that it now depends on θ) and the optimal value $\tilde{f}(\theta) = f^0(\tilde{x}(\theta), \theta)$. The objective of input optimization is to vary θ,

within some pre-determined set, so that we can arrive at a local optimal input θ^*. This input θ^* is such that $\tilde{f}(\theta^*) \leq \tilde{f}(\theta)$ for all $\theta \in N(\theta^*) \cap S$, where S is a set in which perturbations of θ are "stable" (to be defined shortly). The corresponding program (P, θ^*) is a *locally optimal realization* and $\tilde{f}(\theta^*)$ is a *locally optimal value* of the model (P, θ).

In terms of cases similar to those in Chapters 5 and 6, the input parameter θ can correspond to technological coefficients, capacities of machines and/or people, available resources, or any input quantity necessary to define an appropriate mathematical program. Thus varying θ could mean changing the efficiency of a machine or person, or maybe altering a certain product's characteristics. Ideally, in a practical situation and in order to achieve some optimal set-up of the parameters, we don't want to have a drastic jump in our results or output. For example, if improving the efficiency of a component of the model for Company X entails greater profits—great! However, if at one point, further improvements of efficiency necessitate a jump in production rate, we will not strive for this further improvement since the instantaneous jump in production rate may not be possible. As an illustration we take the following example studied in Chapter 7.

Example 1

$$
\begin{aligned}
Min \quad & x_3 + \theta_1 + \theta_2 \\
s.t. \quad & x_1 + x_2 + x_3 - 1 \leq 0 \\
& -x_1 + x_2 + x_3 - 1 \leq 0 \\
& x_1 - \theta_1 x_2 \leq 0 \\
& -x_1 - \theta_2 x_2 \leq 0 \\
& -x_2 \leq 0 \\
& -x_3 \leq 0 \\
& -1 \leq \theta_1 \leq 1 \\
& 0 \leq \theta_2 \leq 1
\end{aligned}
$$

With $\theta = (1, 1)$, the point $\tilde{x} = (0, \frac{1}{2}, 0)$ is optimal (actually all x with $x_1 = x_3 = 0$, $0 \leq x_2 \leq 1$, are optimal). Changing θ to $\theta = (0, 0)$, $\tilde{x}$ remains optimal and the objective value function improves. If we change θ to $\theta = (-\varepsilon, 0)$, $\varepsilon > 0$, the objective value function improves but the optimal solution is forced to jump from a line segment to the point (0, 0, 0). This is the type of improvement one should not seek if stable results (no jumps) are required.

Input optimization guards against such jumps in the output. Only stable perturbations of the input parameter are considered. We want to be able to guarantee that perturbations in θ result in stable, continuous changes in $F(\theta)$, $\tilde{F}(\theta)$ and $\tilde{f}(\theta)$. Therefore, we determine a region $S(\theta) \subset I$ in which all perturbations result in stability. All the results given here hold true if θ is in a region of stability. The following theorem (see Zlobec, Gardner and Ben-Israel [23]) simplifies the task of defining a region of stability.

Theorem 3.1 *Consider the convex model (P, θ) at $\theta^* \in I$. Then the following are equivalent:*

1. *The point-to-set map* $\Gamma : \theta \to F(\theta)$ *is continuous at* θ^*.

2. *For every realistic objective function* f^0, *there exists a neighbourhood* $N(\theta^*)$ *of* θ^* *such that*

   - $\tilde{F}(\theta) \neq \emptyset$ *for every* $\theta \in N(\theta^*)$ *and*
   - $\theta \in N(\theta^*),\ \theta \to \theta^* \Longrightarrow \tilde{F}(\theta)$ *is bounded and all its limit points are in* $\tilde{F}(\theta^*)$.

3. *For every realistic objective function* f^0, *there exists a neighbourhood* $N(\theta^*)$ *of* θ^* *such that*

   - $\tilde{F}(\theta) \neq \emptyset$ *for every* $\theta \in N(\theta^*)$ *and*
   - $\theta \in N(\theta^*),\ \theta \to \theta^* \Longrightarrow \tilde{f}(\theta) \to \tilde{f}(\theta^*)$.

We now define a region of stability. We say that (P, θ) with a realistic objective function f^0 is *stable in a region* $S \subset I \subset R^p$ if we have lower semi-continuity of $\Gamma : \theta \to F(\theta)$ for $\theta \in S$. If we can specify $S = N(\theta^*)$, then (P, θ) is stable at θ^*. Since lower semi-continuity of $\Gamma : \theta \to F(\theta)$ implies continuity of Γ (we already know that it is closed), this definition of a region of stability includes continuity of the output triple $\{F(\theta), \tilde{F}(\theta), \tilde{f}(\theta)\}$ by the above theorem. We now give some basic regions of stability.

Theorem 3.2 *Consider the convex model* (P, θ) *at some* θ^*. *The set*

$$M(\theta^*) \stackrel{\text{def}}{=} \{\theta : F(\theta^*) \subset F(\theta)\}$$

is a region of stability at θ^* *for every realistic objective function.*

This result is immediate if one considers the definition of lower semi-continuity.

Theorem 3.3 *Consider the convex model (P,θ) at some θ^*. If the point-to-set-map $\Gamma : \theta \rightarrow F^{=}(\theta)$ is lower semi-continuous at θ^*, then*

$$R_1(\theta^*) \stackrel{\text{def}}{=} \{\theta : \mathcal{P}^{=}(\theta^*) = \mathcal{P}^{=}(\theta)\}$$

is a region of stability at θ^* for every realistic objective function.

The notation used for the regions of stability is that introduced by Zlobec, et al. There are regions of stability R_2, R_3, and M_1, hence the notation R_1. However, they are not needed for our purpose. For a list of regions of stability see [16], [24], and [26]. Theorem 3.3 was proved by Semple and Zlobec in [20]. The following corollary can be found in [23]; the lemma that follows (a necessary condition for lower semi-continuity of the mapping $F^{=}$) is borrowed from [20].

Corollary 3.4 *Consider the convex model (P,θ) at θ^*. The set*

$$W(\theta^*) \stackrel{\text{def}}{=} \{\theta : F^{=}(\theta^*) \subset F^{=}(\theta) \text{ and } \mathcal{P}^{=}(\theta^*) = \mathcal{P}^{=}(\theta)\}$$

is a region of stability at θ^* for every realistic objective function.

Lemma 3.5 *Consider the convex model (P,θ) at some θ^*. If the point-to-set map $\Gamma : \theta \rightarrow F^{=}(\theta)$ is lower semi-continuous at θ^* then there exists a neighbourhood $N(\theta^*)$ of θ^* such that*

$$\mathcal{P}^{<}(\theta^*) \subset \mathcal{P}^{<}(\theta)$$

for every $\theta \in N(\theta^*)$.

The next three sets are also proved to be regions of stability.

$$V(\theta^*) \stackrel{\text{def}}{=} \{\theta : F^=(\theta^*) \subset F^=(\theta) \text{ and } f^i(x,\theta) \leq 0, \quad \forall x \in F(\theta^*),\ i \in \mathcal{P}^=(\theta^*) \setminus \mathcal{P}^=(\theta)\}$$

$$V_1(\theta^*) \stackrel{\text{def}}{=} \{\theta : F^=(\mathcal{C}^*) \subset F^=(\theta) \text{ and } f^i(x,\theta) \leq 0, \quad \forall x \in F^=(\theta^*),\ i \in \mathcal{P}^=(\theta^*) \setminus \mathcal{P}^=(\theta)\}$$

$$V_3(\theta^*) \stackrel{\text{def}}{=} \{\theta : F^=(\theta^*) = F^=(\theta) \text{ and } f^i(x,\theta) \leq 0, \quad \forall x \in F^=(\theta^*),\ i \in \mathcal{P}^=(\theta^*) \setminus \mathcal{P}^=(\theta)\}$$

It was shown by Zlobec and Ben-Israel in [22] that perturbations inside $V(\theta^*)$ are stable. The regions $V_1(\theta^*)$ and $V_3(\theta^*)$ were presented by Semple and Zlobec in [19]. We now give an example of $V(\theta^*)$.

Example 2

$$\begin{aligned} Min \quad & f^0 = -x_2 \\ s.t. \quad & f^1 = x_1 + x_2 - 1 \leq 0 \\ & f^2 = -x_1 + x_2 - 1 \leq 0 \\ & f^3 = x_1 - \theta x_2 \leq 0 \\ & f^4 = -x_1 - \theta x_2 \leq 0 \\ & f^5 = -x_2 \leq 0 \end{aligned}$$

We see that

$$\mathcal{P}^=(\theta) = \begin{cases} \{3, 4, 5\} & \text{if } \theta < 0 \\ \{3, 4\} & \text{if } \theta = 0 \\ \emptyset & \text{if } \theta > 0 \end{cases}$$

and

$$F^=(\theta) = \begin{cases} x_1 = 0,\ x_2 = 0 & \text{if } \theta < 0 \\ x_1 = 0 & \text{if } \theta = 0 \\ R^2 & \text{if } \theta > 0. \end{cases}$$

The optimal value is

$$\tilde{f}(\theta) = \begin{cases} 0 & \text{if } \theta < 0 \\ -1 & \text{if } \theta \geq 0 \end{cases}$$

and the optimal solution is

$$\tilde{x}(\theta) = \begin{cases} (0,0) & \text{if } \theta < 0 \\ (0,1) & \text{if } \theta \geq 0. \end{cases}$$

At $\theta^* = 0$, the set $V(\theta^*)$ is

$$V(\theta^*) = \{\theta : \theta \geq 0\}.$$

The following is a necessary condition for stability. It was first stated by Semple and Zlobec in [20] but Huang, in [16], showed that the assumption on path connectedness is unnecessary. It is stated here in its latest form.

Theorem 3.6 *Consider the convex model (P, θ) at some $\theta^* \in I$. Let S be a region of stability at θ^* for every realistic objective function at θ^*. Then there exists a neighbourhood $N(\theta^*)$ of θ^* such that*

$$\mathcal{P}^=(\theta) \subset \mathcal{P}^=(\theta^*)$$

for every $\theta \in N(\theta^*) \cap S$.

The numerical method used in the case studies finds a path connecting an initial θ^0 to an optimal θ^* by iteration—the method for some special cases is described in detail in the next chapter. Let us observe that the optimal input θ^* generally depends on the choice of the initial input θ^0. For example, suppose we have the following:

Example 3

$$\begin{aligned} Min \quad & f^0 = -x + \theta_2 \\ s.t. \quad & f^1 = -x - 1 \leq 0 \\ & f^2 = x - 1 \leq 0 \\ & f^3 = (\theta_1^2 - \theta_2^2)^2 x \leq 0 \end{aligned}$$

with $I = \{\theta : \|\theta\| \leq 1\}$, then we find that

$$F(\theta) = \begin{cases} [-1, 0] & \text{if } \theta_1^2 \neq \theta_2^2 \\ [-1, 1] & \text{if } \theta_1^2 = \theta_2^2 \end{cases}$$

If our initial θ is such that $\theta_1^2 = \theta_2^2$ then optimal inputs are $\theta^* = (\frac{1}{2}, -\frac{1}{2})$ and $\theta^* = (-\frac{1}{2}, -\frac{1}{2})$, the optimal solution is $\tilde{x}(\theta^*) = 1$ and the optimal value function is $\tilde{f}(\theta^*) = -\frac{3}{2}$. If, on the other hand, our initial θ is, say, $\theta = (0, -\frac{1}{4})$ then the optimal input is $\theta^* = (0, -1)$ with $\tilde{x}(\theta^*) = 0$ and $\tilde{f}(\theta^*) = -1$. To establish the optimality of θ^*, we use this sufficient condition, adapted from [25].

Theorem 3.7 *Consider the convex model (P, θ) with a realistic objective function at some $\theta^* \in I$. Let $x^* \in F^=(\theta^*)$ and $u^* \in R_+^{q(\theta^*)}$ be some given points, and let $S(\theta^*)$ be some region of stability at θ^*. If there exists a non-negative vector function*

$$\mathcal{U} : S(\theta^*) \cap N(\theta^*) \to R_+^{q(\theta^*)}$$

such that $\mathcal{U}(\theta^*) = u^*$ and with the property that for every $\theta \in S(\theta^*)$ the saddle-point inequality

$$L_*^<(x^*, u; \theta^*) \leq L_*^<(x^*, \mathcal{U}(\theta^*); \theta^*) \leq L_*^<(x, \mathcal{U}(\theta); \theta)$$

holds for every $u \in R_+^{q(\theta^*)}$ and every $x \in F^=(\theta)$ then θ^* is a locally optimal input with respect to $S(\theta^*)$ and x^* is a corresponding optimal solution, i.e., $x^* \in \tilde{F}(\theta^*)$

where $q(\theta^*)$ is the cardinality of $\mathcal{P}^<(\theta^*)$ and $R_+^{q(\theta^*)}$ is the non-negative orthant of $R^{q(\theta^*)}$.

To apply this theorem, one must therefore find such a function $\mathcal{U}$. When the program (P, θ) is non-trivial, such as in the case studies, determining this function is no easy task. However, if Slater's condition holds then we can use the following result from [3], abbreviated here as the BBZ condition, and the solution u will be suitable for use as $\mathcal{U}(\theta)$ in Theorem 3.7. The BBZ condition is a complete characterization (i.e., without assuming any constraint qualifications) of optimality of a solution x^* to a convex *program*. Therefore, in our case, we fix $\theta = \theta^*$ and then use it. For unstable models, a complete characterization of an optimal input was obtained by van Rooyen and Zlobec (see [21]). A model which is unstable at θ^* is simply not stable for all $\theta \in N(\theta^*)$. A model which is stable at θ^*, as defined in Chapter 2, is stable for all perturbations of θ^* in a feasible neighbourhood (i.e., $\theta \in I$) of θ^*. The following theorem recalls the BBZ condition, where θ is fixed at $\theta = \theta^*$.

Theorem 3.8 *Consider the differentiable convex program* (C) *and* $x^* \in F(\theta^*)$. *Then* x^* *is optimal if, and only if, the system below is consistent:*

$$\nabla f^0(x^*) + \sum_{i \in \mathcal{P}(x^*) \setminus \mathcal{P}^=} u_i \nabla f^i(x^*) \in \{ \bigcap_{i \in \mathcal{P}^=} D_i^=(x^*) \}^+$$

$$u_i \geq 0, \quad i \in \mathcal{P}(x^*) \setminus \mathcal{P}^=$$

Here $D_i^=(x^*) = \{d \in R^n : \exists \bar{\alpha} > 0 \text{ such that } f^i(x^* + \alpha d) = f(x^*)\ \forall 0 < \alpha < \bar{\alpha}\}$ is known as the cone of directions of constancy and $\mathcal{P}(x^*) = \{i \in \mathcal{P} : f^i(x^*) = 0\}$ is the set of active constraints at x^*. Since f^i in (C) is convex and, for the purposes of

the theorem, differentiable, then $D_i^=$ is a convex cone. The set $M^+ = \{u : (u,x) \geq 0, \ \forall x \in M\}$ is the polar of the set M. If Slater's condition holds, then $\mathcal{P}^= = \emptyset$ and the BBZ system becomes

$$\nabla f^0(x^*) + \sum_{i \in \mathcal{P}(x^*)} u_i \nabla f^i(x^*) = 0$$

$$u_i \geq 0, \quad i \in \mathcal{P}(x^*)$$

which is the well-known Karush-Kuhn-Tucker condition.

A necessary condition for a locally optimal input will now be given. First, however, some definitions are needed. If $S(\theta^*)$ is a region of stability at θ^*, then we define

$$B_1 = B_1(\theta^*) \stackrel{\text{def}}{=} \left\{ \frac{\theta - \theta^*}{\|\theta - \theta^*\|} \in R^p : \theta \in N(\theta^*) \cap S(\theta^*), \ \theta \neq \theta^* \right\}.$$

This corresponds to a section of the unit ball in R^p. The set B_1^0 is the set of limit points of B_1 as $\theta \to \theta^*$, $\theta \in S(\theta^*)$. The following index condition is denoted (IND) and is also needed for the necessary condition:

$$(IND) \qquad \{\mathcal{P}^<(\theta^k) \cap \mathcal{P}(\tilde{x}(\theta^*))\} \subset \mathcal{P}^<(\theta^*)$$

as $\theta^k \to \theta^*$, $\theta^k \in S(\theta^*)$. Although there are several necessary conditions for an optimal input for convex models, we concentrate here only on bi-convex models.

Theorem 3.9 *Consider the bi-convex model (P, θ) with a realistic objective function at $\theta^* \in I$. Assume that the corresponding saddle-point*

$$\{\tilde{x}(\theta^*), \ \tilde{u}_i(\theta^*) : i \in \mathcal{P}^<(\theta^*)\}$$

is unique and that the functions $f^i(\tilde{x}(\theta^*),\theta)$, $i \in \{0\} \cup \mathcal{P}^<(\theta^*)$ *are differentiable at* θ^*. *Also assume that the index condition* (IND) *holds at* θ^*, *relative to* $V_1(\theta^*)$.

If θ^* *is a locally optimal input with respect to* $V_1(\theta^*)$ *then*

$$\nabla_\theta L_*^<(\tilde{x}(\theta^*),\tilde{u}(\theta^*);\theta)_{\theta=\theta^*} \in (B_1^0)^+.$$

The following theorem presents a marginal value formula used in input optimization. It guarantees that the numerical method iterates from θ^k to θ^{k+1} only if $\tilde{f}(\theta^{k+1}) < \tilde{f}(\theta^k)$.

Theorem 3.10 *Consider the bi-convex model* (P,θ) *at* $\theta^* \in I$ *with a realistic objective function. Suppose the corresponding saddle-point* $\{\tilde{x}(\theta^*),\ u_i(\theta^*) :\ i \in \mathcal{P}^<(\theta^*)\}$ *is unique and the index condition* (IND) *holds at* θ^* *with respect to* $S = V_3(\theta^*)$. *Also suppose that* $f^i(x,\cdot)$, $i \in \{0\} \cup \mathcal{P}^<(\theta^*)$ *are differentiable functions in* $V_3(\theta^*) \cap N(\theta^*)$, *where* $N(\theta^*)$ *is some neighbourhood of* θ^*, *and that* $\nabla_\theta f^i(x,\theta^*)$, $i \in \{0\} \cup \mathcal{P}^<(\theta^*)$ *are continuous functions in* x *at* $\tilde{x}(\theta^*)$.

Then for every fixed path $\theta \in V_3(\theta^*)$, $\theta \to \theta^*$ *such that the limit*

$$\lim_{\substack{\theta \in V_3(\theta^*) \\ \theta \to \theta^*}} \frac{\theta - \theta^*}{\|\theta - \theta^*\|} \stackrel{\text{def}}{=} \ell$$

exists, we have

$$\lim_{\theta \in V_3(\theta^*)} \frac{\tilde{f}(\theta) - \tilde{f}(\theta^*)}{\|\theta - \theta^*\|} = (\nabla g(\theta^*),\ell)$$

where

$$g(\theta) \stackrel{\text{def}}{=} L_*^<(\tilde{x}(\theta^*),\tilde{u}(\theta^*);\theta).$$

If the inner product $(\nabla g(\theta^*), \ell)$ is negative, then we know $\tilde{f}(\theta) < \tilde{f}(\theta^*)$ for θ "close" to θ^*, i.e., moving from θ^* to θ improves the model (we know we can move from θ^* to θ in a stable manner because the limit is taken inside $V_3(\theta^*)$). If the saddle-point $\{\tilde{x}(\theta^*), \tilde{u}_i(\theta^*),\ i \in \mathcal{P}^<(\theta^*)\}$ is not unique, then one can still use the theorem. In this case, one simply gets an approximation to the limit

$$\lim_{\theta \in V_3(\theta^*)} \frac{\tilde{f}(\theta) - \tilde{f}(\theta^*)}{\|\theta - \theta^*\|}$$

by using

$$\min_{\tilde{x} \in \tilde{F}(\theta^*)} \max_{\tilde{u}(\theta^*)} (\nabla g(\theta^*), \ell)$$

For more on this, see Zlobec [24] and Eremin and Astafiev [11].

To demonstrate the marginal value formula, consider Example 3 at $\theta^* = (0, 0)$. Then $V(\theta^*) = \{\theta : \theta_1^2 = \theta_2^2\}$, $\tilde{x}(\theta^*) = 1$, $\mathcal{P}^<(\theta^*) = \{1, 2\}$, $\tilde{u}_1(\theta^*) = 0$, $\tilde{u}_2(\theta^*) = 1$ and

$$g(\theta) = -1 + \theta_2,$$

which establishes

$$\nabla g(\theta^*) = (0, 1)^T.$$

For the path $\theta = \theta^* + \alpha d$, $\alpha > 0$ with $d = (-1, -1)$ we have (as will be explained in Chapter 4)

$$\ell = \frac{d}{\|d\|} = (-1, -1)^T.$$

The marginal value formula gives

$$(\nabla g(\theta^*), \ell) = -1 < 0$$

and thus indicates a local direction of improvement of the optimal value function $f(\theta)$.

The theorems just presented are of a local nature. The results hold true for θ in some neighbourhood $N(\theta^*)$. However, we would like to know how far we can stray from our current θ and still ensure stability of the model. This quantity or distance is known as the *radius of stability* r. That is, we want to determine the largest r such that for every $\theta \in S(\theta^*, r) = \{\theta : \|\theta - \theta^*\| < r\}$, the model is uniformly stable in this neighbourhood. Note that for a linear model, where the constraints are of the form $Ax - b \leq 0$, $A = (a_{ij})$, $a_{ij} > 0$, $b = (b_j)$, $b_j > 0$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, decreasing a_{ij} increases the set F, thus guaranteeing stability. This amounts to the whole set I being the region of stability $M(\theta^*)$ (unless some values of $\theta \in I$ lead to infeasibility of the constraints), and $r = \min_{\theta \in \partial I} \|\theta^* - \theta\|$, where ∂I is the boundary of the closure of the set I.

The basic difference between input optimization and usual mathematical programming is that the latter ignores the present state of the system and global stability. Therefore, the solutions obtained by input optimization and mathematical programming (after considering $z = (x, \theta)$ as a single vector variable) do not generally coincide! For examples, see [28].

Chapter 4

A Numerical Method for Linear Input Optimization

4.1 General Method

A general description of a numerical optimization method for solving linear input optimization problems is included in this section. The problem to which it is applied is assumed to be of the form

$$(L) \qquad \begin{aligned} Max \quad & (c, x) \\ s.t. \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where $A \in R^{m' \times n}$, $b \in R^{m'}$, $c \in R^n$. The input parameter is introduced into the matrix A. Cases where only the vectors b and c vary have been studied by others (see for example, Gal [12], and Guddat [13]) hence the interest here is to vary A.

Our system (L) therefore becomes the system (L,θ)

$$(L,\theta) \qquad \begin{array}{ll} Max & (c,x) \\ s.t. & A(\theta)x \leq b \\ & x \geq 0 \\ & \theta \in I \end{array}$$

which, in order to use input optimization is transformed into

$$\begin{array}{ll} Min & f^0 = (-c,x) \\ s.t. & f^i = (a^i(\theta),x) - b_i \leq 0, \quad i = 1,\ldots,m' \\ & f^j = -x_j \leq 0, \quad j = m'+1,\ldots,n \\ & \theta \in I. \end{array}$$

Here a^i, $i = 1,\ldots,m'$ is the i^{th} row of the matrix A and I is some arbitrary but fixed set in R^p. This system now has the form of a convex program (P,θ).

The numerical method is iterative in nature, where each iteration yields a vector θ^k such that $\tilde{f}(\theta^k) < \tilde{f}(\theta^{k-1})$. It is assumed that any perturbation within I is stable and maintains the feasibility of the constraints f^i, $i \in \mathcal{P}$. This set is bounded by the fixed vectors $L \in R^p$ and $U \in R^p$:

$$I = \{\theta \in R^p : L \leq \theta \leq U\}.$$

Two versions of the method, differing only slightly one from the other, will be given here. The difference lies in the way of choosing a direction emanating from the current θ^k which improves the optimal value function. The path emanating from θ^k to θ^{k+1} is chosen to be linear, i.e., $\theta^{k+1} = \theta^k + \alpha d$, $\alpha \in R_+$, $d \in R^p$. A non-linear path could have been chosen, however since the problems studied here are linear, it is natural to assume linearity of the path. The limit ℓ, defined and used in the

marginal value formula (Theorem 3.9) can therefore be expressed as

$$\ell = \lim_{\theta \to \theta^*} \frac{\theta - \theta^*}{\|\theta - \theta^*\|} = \lim_{\alpha \to 0^+} \frac{\alpha d}{|\alpha| \cdot \|d\|} = \frac{d}{\|d\|}.$$

The norm used is the ℓ_∞ norm, i.e., for $y \in R^p$, $\|y\| = \max_{1 \le i \le p} |y_i|$. Recall that the marginal value formula holds for all norms, not only the Euclidean norm.

The method starts by fixing $\theta^k = \theta^0$ $(k = 0)$ and solving (L, θ^k) using the revised simplex method. The solution $\tilde{x}(\theta^k)$ and the dual solution $\tilde{u}(\theta^k)$ enable us to calculate $\nabla g(\theta^k)$. By the marginal value formula, we want the inner product $(\nabla g(\theta^k), \ell)$ to be negative. This will guarantee that $\tilde{f}(\theta^{k+1}) < \tilde{f}(\theta^k)$, i.e., a local improvement of the optimal value function. The inner product, in view of the type of path, is

$$\frac{1}{\|d\|}(\nabla g(\theta^k), d)$$

(the direction d is only zero if the current θ is optimal). Ideally, we would like to solve

$$\begin{aligned} Min \quad & \tfrac{1}{\|d\|}(\nabla g(\theta^k), d) \\ s.t. \quad & \|d\| \le 1 \\ & \theta^k + d \in I \end{aligned}$$

to get a "best" d. However this involves minimizing a non-linear objective function. We therefore proceed as follows: We say that d is a *direction of steepest descent at* θ^* if $\|d\|_\infty = 1$ and d minimizes

$$\lim_{\alpha \to 0^+} \frac{\tilde{f}(\theta^* + \alpha d) - \tilde{f}(\theta^*)}{\alpha}$$

i.e., if d is on the unit sphere in ℓ_∞ and d minimizes the directional derivative $\tilde{f}(\theta^*)$. To determine such a direction we want to

$$\min_{d \in B_\infty} \lim_{\alpha \to 0^+} \frac{\tilde{f}(\theta^* + \alpha d) - \tilde{f}(\theta^*)}{\alpha}$$

where $B_\infty = \{x : \|x\|_\infty = 1\}$. If the direction d that we obtain is such that the limit as $\alpha \to 0^+$ is negative then we know $(\nabla g, \ell) < 0$. If we also get d such that $\|d\|_\infty = 1$ then $\frac{1}{\|d\|}(\nabla g, d) = (\nabla g, d)$ and so $(\nabla g, \ell)$ is minimal and d is a direction of usual steepest descent. If $\|d\|_\infty < 1$ then $(\nabla g, \ell)$ is still negative but not necessarily minimal. Either way, we know that the direction d improves the optimal value function $\tilde{f}(\theta)$ because $(\nabla g, \ell) < 0$. Hence we now solve

$$\begin{aligned} Min \quad & (\nabla g(\theta^k), d) \\ s.t. \quad & \|d\| \leq 1 \\ & \theta^k + d \in I \end{aligned}$$

The feasible set here is compact since both I and the unit ball are compact. Thus, a minimum exists for the continuous, linear functional, due to the Weierstrass Theorem. In order to find an optimal d, one can benefit from the structure of the set I. One simply chooses

$$d_i = \begin{cases} \max(L_i - \theta_i^k, -1) & \text{if } \nabla g_i(\theta^k) > 0 \\ 0 & \text{if } \nabla g_i(\theta^k) = 0 \\ \min(U_i - \theta_i^k, 1) & \text{if } \nabla g_i(\theta^k) < 0 \end{cases}$$

where $L = (L_i)$, $U = (U_i)$, $i \in \{1, \ldots, p\}$ are the upper and lower bounds of the set I. If we have $\mid \nabla g_i(\theta^k) \mid < \varepsilon$ for $i \in \{1, \ldots, p\}$ for some pre-determined $\varepsilon > 0$, then $\nabla g_i(\theta^k)$ is assumed to be zero and we set $d_i = 0$. Similarly, if for some $i \in \{1, \ldots, p\}$, $d_i > 0$ and $\mid U_i - \theta_i^k \mid < \varepsilon'$ or $d_i < 0$ and $\mid L_i - \theta_i^k \mid < \varepsilon'$ (i.e., some boundary is "close" to θ^k) where ε' is another pre-determined positive quantity, then we reset d_i to zero. If $d = 0$ then the method terminates and, according to the marginal value fromula theorem, one cannot improve the objective value function locally while remaining in I.

The second version of the numerical optimization method differs in its way of choosing the direction d. Instead of solving a minimization problem, we simply

assign

$$d_i = -\text{sgn}(\nabla g_i(\theta^k)).$$

Again, if some $| \nabla g_i(\theta^k) | < \varepsilon$ then we set $d_i = 0$. If θ^k is located on a boundary of I and some $d_i e$ (here $e = (1, 1, \ldots, 1)^T$) points out of I, then that d_i is simply reset to zero. The resulting direction d is still such that $(\nabla g(\theta^k), d) < 0$ (note that, here, $\|d\| = 1$). If, at this point, $d = 0$, then only now do we solve

$$\begin{aligned} Min \quad & (\nabla g(\theta^k), d) \\ s.t. \quad & \|d\| \leq 1 \\ & \theta^k + d \in I \end{aligned}$$

If we still get $d = 0$ then the method terminates; if not, then we normalize d and continue iterating. Note that the second version is simpler, but we expect it to produce slower convergence.

At this step, in both versions, we have a direction d along which $\tilde{f}(\theta)$ decreases, at least locally. Now comes the question of choosing the distance we will travel along d in order to get θ^{k+1}. That is, we now must choose α, where $\theta^{k+1} = \theta^k + \alpha d$. To do this, we first determine $\bar{\alpha}$, which is the greatest distance we can travel along d and still remain in I. Once we know $\bar{\alpha}$, we use the Method of Golden Rule (MGR) to find α. This MGR is an iterative method for minimizing a real-valued function on a closed, bounded interval $[a, b]$ in R, provided we can evaluate the function at specific points in the interval. Our interval is $[0, \bar{\alpha}]$ and the function is $\tilde{f}(\alpha) = \tilde{f}(\theta^k + \alpha d)$ which is an unknown function mapping R into R. However we can evaluate $\tilde{f}(\theta^k + \alpha d)$ for $\alpha \in [0, \bar{\alpha}]$; we simply solve the linear program $(L, \theta^k + \alpha d)$ (the difficulty here is that the optimal value function $\tilde{f}(\theta)$ is not explicitly known, so the usual methods of nonsmooth analysis are not directly applicable).

The 'Golden' in MGR refers to the so-called "golden" proportion

$$\gamma = \frac{3 - \sqrt{5}}{2}$$

or, approximately $\gamma = 0.3819660\ldots$. The function $\tilde{f}(\alpha)$ is evaluated at $a = 0$, $y = a + \gamma(b - a)$, $z = b - \gamma(b - a)$, and $\bar{\alpha}$. If $\tilde{f}(y) \leq \tilde{f}(z)$ then we assign $b = z$ and repeat the iteration on the new interval $[a, b] = [a, z]$. If $\tilde{f}(y) > \tilde{f}(z)$ then we assign $a = y$ and repeat the iteration on the new interval $[a, b] = [y, b]$. Iterations are continued until the interval being considered has a length less than some $\varepsilon > 0$. This final interval after $n + 1$ iterations is smaller than the interval obtained by the Fibonacci search after n iterations. The MGR is much simpler than Fibonacci and yet is only marginally slower to converge. Greater detail is provided by, e.g., Petrić and Zlobec in [18]. There are two different stopping rules for the MGR. When we start iterating from θ^0 to θ^1, to θ^2, etc., it is less important that our α be the best on the interval $[0, \bar{\alpha}]$ than when our θ's approach an optimal θ^*. We are therefore not very demanding of the MGR, choosing α after only three iterations. However, as we approach θ^*, we are more demanding for the choice of α. Our rule implemented is as follows: If the percentage of change in $\tilde{f}(\theta)$ from θ^{k-2} to θ^{k-1} is greater than some pre-established quantity **PERCENT** then we perform only three iterations of MGR to get α on the k^{th} iteration. If the change is less than **PERCENT** then MGR is carried out until the interval $[a, b]$ on which we study $\tilde{f}(\alpha)$ is reduced to a length of less than some specified amount δ (this amount and **PERCENT** are determined before commencing the numerical method and depend on the problem being studied).

If the "theta-iterations"—a theta-iteration is an iteration from θ^k to θ^{k+1} where

$\tilde{f}(\theta^{k+1}) < \tilde{f}(\theta^k)$; in order to determine θ^{k+1}, we use "MGR-iterations" to establish an α as described above—are in their early stages, then our MGR stopping rule is three MGR-iterations. It is possible that after these three iterations, $\tilde{f}(\alpha) > \tilde{f}(0)$. Such would be the case if $\tilde{f}(\alpha)$ decreases close to $\alpha = 0$ and quickly rises from then on. In these circumstances the MGR-iterations are continued until an α is found with $\tilde{f}(\alpha) < \tilde{f}(0)$ or until there have been thirty MGR-iterations (to guard against infinite loops).

At this stage, we have α and d, so we set $\theta^{k+1} = \theta^k + \alpha d$, adjust the affected coefficients a_{ij} and repeat the iteration with θ^{k+1}. The algorithm for the first version is now given in point form.

Algorithm

1. Set $\theta^k = \theta^0$.

2. Update the matrix $A = A(\theta^k)$ (or, initially, specify which elements of A are to be perturbed).

3. Solve (L, θ^k) to get $\tilde{x}(\theta^k)$ and $\tilde{u}(\theta^k)$.

4. Calculate $\nabla g(\theta^k)$ and determine d. If $d = 0$, stop—the current θ is a locally optimal input; if not, continue.

5. Calculate $\bar{\alpha}$ and then use MGR to find α. If

$$\frac{\tilde{f}(\theta^{k-2}) - \tilde{f}(\theta^{k-1})}{\tilde{f}(\theta^{k-2})} \geq \text{PERCENT}$$

then perform only three MGR-iterations to get α. Otherwise iterate until the interval $[a, b]$ has length $\mid b - a \mid < \delta$.

6. Set $\theta^{k+1} = \theta^k + \alpha d$. Repeat from Step 2.

4.2 An Adaptation for Use in the Case Studies

The program listed in Appendix B performs the numerical optimization method just described, on Case study I (given later in Chapter 5). Some further specifications to the method were necessary to adapt it to the situations presented by the case studies. The program listed is very specific. It was not made general because it depends heavily on the nature of the functions $a_{ij}(\theta)$.

The input parameter θ appears in the matrix A in our studies. However, if a choice is possible, one must choose carefully which elements a_{ij} should depend on θ and which should remain constant. Perturbing a particular a_{ij} may have no effect on the optimal value function whereas a small perturbation in another element a_{ij} may result in a great improvement in the optimal value function. To establish which elements a_{ij} are most sensitive to small perturbations, we use the marginal value formula of Theorem 3.9. We have

$$g(\theta) = -\sum_{j=1}^{n} c_j \tilde{x}_j + \sum_{i=1}^{m'} \sum_{j=1}^{n} a_{ij} \tilde{u}_i \tilde{x}_j - \sum_{i=1}^{m'} \tilde{u}_i b_i.$$

If our perturbations are made along a line then we can specify

$$\theta = \theta^* + \alpha s$$

where $\|s\| = 1$, $\alpha > 0$. Adapting the marginal value formula to this situation we get

$$(\nabla g(\theta^*), s) = \lim_{\alpha \to 0^+} \frac{\tilde{f}(\theta^* + \alpha s) - \tilde{f}(\theta^*)}{\alpha} = \frac{\partial \tilde{f}(\theta^*)}{\partial s}$$

Let the vector s have $n + (m' \times n) + m'$ components, one corresponding to each of c_j, a_{ij}, b_i, $i = 1, \ldots, m'$, $j = 1, \ldots, n$. Then, to isolate a particular a_{ij} in s, say, a_{kl}, $1 \leq k \leq m'$, $1 \leq l \leq n$, which will give us

$$\frac{\partial \tilde{f}(\theta^*)}{\partial a_{kl}}$$

we set all elements of s equal to zero except the one corresponding to a_{kl}, which is set equal to one. Then it is obvious that

$$\frac{\partial \tilde{f}(\theta^*)}{\partial a_{kl}} = \tilde{u}_k \tilde{x}_l.$$

We will use this formula in the case studies to select the a_{ij}'s which will depend on θ.

The input needed by the program includes the matrix A $(m' \times n)$, the vectors b and c, the initial input parameter θ^0, which a_{ij} are functions of θ, and the set I determined by the input vectors L and U. If a more general type of set I is required, then modifications to the method will be necessary. Our method assumes that any perturbation within I is stable (this significantly simplifies our method). Hence the vectors L and U must be initialized in such a way that I is itself a region of stability. In Case study II, this required readjusting the initial estimates for L and U, as will be described in Chapter 6. If $I \neq S(\theta^*)$, i.e., if there are parts of I which are unstable, then once we determine $\bar{\alpha}$, it is possible that some α, $0 < \alpha < \bar{\alpha}$, be such that $\theta^k + \alpha d \notin S(\theta^*)$. In which case, the theorems and results do not

necessarily hold for this α. In some instances this may also imply that a particular θ^* is unattainable from our current θ^k due to "patches" of instability between them. One must therefore be sure no such "holes" exist (before applying our method).

The program parameter ε mentioned in the description of the method, was set equal to 5×10^{-8} in both cases whereas ε' was 5×10^{-8} in Case study I and 5×10^{-9} in Case study II. In both case studies, the quantities δ and PERCENT were 10^{-3} and 0.001% respectively.

Chapter 5

Case Study I: The Shenandoah Valley Textile Mill

The first of two cases studied is the Shenandoah Valley Textile Mill[1]. A complete decription of the case can be found in Naylor, et al. [17]. We give here a brief outline of the case.

The objective of the case presented in [17] is to determine a suitable linear program which reflects the company's operations and to solve it to establish an optimal production plan. Our study of the mill commences at this point. Using input optimization, we would like to improve the efficiency of certain departments in order to get a more profitable production plan than the original one, obtained by solving the linear program.

The mill, on the recommendation of the marketing research department, will

[1]The company's name and operating figures were changed to maintain the anonymity of the firm.

produce seven different styles of cotton cloth for the coming week. They are: one bleached style (denote the number of yards produced by B), four printed styles (P_1, P_2, P_3, and P_4) and two dyed styles, blue and red (D_1 and D_2). To produce the finished cloths, the mill purchases large quantities of rough, unfinished cotton cloth and converts it into the different styles through a series of finishing operations. If the mill functions at maximum capacity, there will be excess capacity in several of the finishing operations. Hence, only restrictive operations need be considered. They are: singeing, desizing, kier-boiling, bleaching, drying, mercerizing, printing, aging, dyeing (blue or red), starching, and calendering. There are two types of constraints for this mill. The first type is the process restraints. They are determined by the production rates for each of the seven styles for each of the eleven restrictive operations and by the maximum number of process-hours that will be available next week for the restrictive operations. The rates and maximum number of process-hours are given in Tables 5.1 and 5.2. The coefficients of the process restraints are expressed in hours per yard; hence, to get the coefficients for the matrix A, we must use the reciprocals of the values in Table 5.1.

The other constraints are imposed by contract demands and estimated sales. The company is committed by contract to supply a garment factory with 5 000 yards of each of the printed styles. The sales department has estimated that the maximum possible sales for the third and fourth printed styles will be 100 000 yards and 50 000 yards respectively. The company should be able to sell all it can produce of the other five styles. The revenue from each yard of each style is outlined in Table 5.3.

The linear program which reflects this system is given in Table 5.4 (coefficients

Process	Styles						
	B	P_1	P_2	P_3	P_4	D_1	D_2
Singeing	9 000	6 000	9 000	7 000	8 000	9 000	8 000
Desizing	13 000	10 000	9 000	11 000	8 000	13 000	12 000
Kier Boiling	1 500	900	1 000	800	900	1 300	1 200
Bleaching	1 000	1 100	1 050	1 100	1 100	1 100	1 200
Drying	13 000	10 000	10 000	12 000	11 000	11 000	12 000
Mercerizing	800	550	600	650	700	700	800
Printing	—	300	300	200	250	—	—
Aging	—	5 000	4 000	4 000	6 000	—	—
Dyeing	—	—	—	—	—	—	—
(blue)	—	—	—	—	—	4 000	—
(red)	—	—	—	—	—	—	3 500
Starching	2 000	1 800	1 800	1 600	1 500	2 000	1 500
Calendering	4 000	5 000	3 000	2 500	4 000	3 200	3 500

Table 5.1: Production rates in yards per hour. Source: Naylor, et al., [17].

in the matrix A are rounded only in this table, not when calculations are carried out).

Using the revised simplex method (the program listing is included in Appendix A), the optimal production plan for the coming week for this initial set-up appears in Table 5.5. The optimal value is -1 022 380.000$ (recall that we're minimizing $(-c, x)$). The dual solution is $\tilde{u}_8 = 0.336$, $\tilde{u}_{13} = 0.144$, $\tilde{u}_{15} = 0.200$, $\tilde{u}_{19} = 0.100$, $\tilde{u}_{20} = 0.290$, $\tilde{u}_i = 0$, $i = 1, \ldots, 7, 9, 10, 11, 12, 14, 16, 17, 18$.

We now would like to introduce an input parameter. We must decide which elements a_{ij} should be perturbed. Using the formula

$$\frac{\partial \tilde{f}}{\partial a_{ij}} = \tilde{u}_i \tilde{x}_j$$

OPERATION	CAPACITY
Singeing	150
Desizing	150
Kier Boiling	900
Bleaching	1 500
Drying	140
Mercerizing	
(bleached)	830
(printed)	830
(dyed)	830
Printing	1 800
Aging	150
Dyeing	
(blue)	150
(red)	140
Starching	500
Calendering	450

Table 5.2: Maximum Number Of Process Hours Available. Source: Naylor, et al., [17].

STYLE	REVENUE PER YARD
B	0.40$
P_1	0.60
P_2	0.80
P_3	1.00
P_4	1.25
D_1	1.20
D_2	1.30

Table 5.3: Revenue from each style. Source: Naylor, et al., [17] p.184.

Max	$0.40B$	+	$0.60P_1$	+	$0.80P_2$	+	$1.00P_3$	+	$1.25P_4$	+	$1.20D_1$	+	$1.30D_2$					
s.t.	$1.1B$	+	$1.7P_1$	+	$1.1P_2$	+	$1.4P_3$	+	$1.2P_4$	+	$1.1D_1$	+	$1.2D_2$	$\leq$	1 500 000
	$7.7B$	+	$10.0P_1$	+	$11.1P_2$	+	$9.1P_3$	+	$12.5P_4$	+	$7.7D_1$	+	$8.3D_2$	$\leq$	15 000 000
	$6.7B$	+	$11.1P_1$	+	$10.0P_2$	+	$12.5P_3$	+	$11.1P_4$	+	$7.7D_1$	+	$8.3D_2$	$\leq$	9 000 000
	$10.0B$	+	$9.1P_1$	+	$9.5P_2$	+	$9.1P_3$	+	$9.1P_4$	+	$9.1D_1$	+	$8.3D_2$	$\leq$	15 000 000
	$7.7B$	+	$10.0P_1$	+	$10.0P_2$	+	$8.3P_3$	+	$9.1P_4$	+	$9.1D_1$	+	$8.3D_2$	$\leq$	14 000 000
	$1.2B$													$\leq$	830 000
			$1.8P_1$	+	$1.7P_2$	+	$1.5P_3$	+	$1.4P_4$					$\leq$	830 000
											$1.4D_1$	+	$1.2D_2$	$\leq$	830 000
			$3.3P_1$	+	$3.3P_2$	+	$5.0P_3$	+	$4.0P_4$					$\leq$	1 800 000
			$2.0P_1$	+	$2.5P_2$	+	$2.5P_3$	+	$1.7P_4$					$\leq$	1 500 000
											$2.5D_1$			$\leq$	1 500 000
													$2.9D_2$	$\leq$	1 400 000
	$5.0B$	+	$5.6P_1$	+	$5.6P_2$	+	$6.3P_3$	+	$6.7P_4$	+	$5.0D_1$	+	$6.7D_2$	$\leq$	5 000 000
	$2.5B$	+	$2.0P_1$	+	$3.3P_2$	+	$4.0P_3$	+	$2.5P_4$	+	$3.1D_1$	+	$2.9D_2$	$\leq$	4 500 000
			$-1.0P_1$											$\leq$	$-5\,000$
					$-1.0P_2$									$\leq$	$-5\,000$
							$-1.0P_3$							$\leq$	$-5\,000$
									$-1.0P_4$					$\leq$	$-5\,000$
							$1.0P_3$							$\leq$	100 000
									$1.0P_4$					$\leq$	50 000

$B \geq 0, \qquad D_1 \geq 0, \qquad D_2 \geq 0$

Table 5.4: Linear program reflecting the initial set-up of the mill. Source: Naylor, et al., [17], p.185

STYLE	QUANTITY (YARDS)
B	0
P_1	5 000
P_2	199 600
P_3	100 000
P_4	50 000
D_1	581 000
D_2	0

Table 5.5: Optimal production schedule for the initial set-up of the mill.

we determine that perturbing the coefficients a_{86}, $a_{13,3}$, and $a_{13,6}$ will affect the optimal value function. These correspond to the number of yards of D_1 produced by the mercerizing operation and the starching operation and the number of yards of P_1 produced by the starching operation. The dependance of the three technological coefficients on θ will be as follows:

$$a_{86} = \frac{1}{700 + \theta_1}, \quad a_{13,3} = \frac{1}{1\,800 + \theta_2}, \quad a_{13,6} = \frac{1}{2\,000 + \theta_3}.$$

Now we must define the set I in which θ can be perturbed. In a real-life setting, the management would be able to define it by considering the possible improvements in the efficiency of the operations involved. For example, the possible improvement as a result of the replacement of a particular machine with a newer, more efficient one. In this study we will estimate reasonable values which define I. We will set

$$L = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad U = \begin{bmatrix} 400 \\ 700 \\ 1\,000 \end{bmatrix}$$

i.e., we will consider improvements of an extra 400 yards per hour in the mercerizing of dyed style one, and so on with the other two. Our objective is to find a $\theta^* \in I$ which is an optimal input. All perturbations within I are stable since the point $\hat{x} = (1, 5\,001, 5\,001, 5\,001, 5\,001, 1, 1)$ satisfies Slater's condition for all $\theta \in I$. The marginal value formula theorem can be used in this case because:

- The model is bi-convex, with a realistic objective function (at every $\theta^* \in I$).
- The saddle-point $\{\tilde{x}(\theta^*), \tilde{u}_i(\theta^*),\ i \in \mathcal{P}^<(\theta^*)\}$ is not necessarily unique, but as noted in Chapter 3, this condition can be relaxed (we are not really aiming at the locally fastest improvement).

k	θ_1^k	θ_2^k	θ_3^k	$\tilde{f}(\theta^k)$
1	0.000000	0.000000	0.000000	-1 022 380.000
2	341.640786	341.640786	341.640786	-1 172 990.927
3	391.485505	391.485505	391.485505	-1 187 869.340
4	398.757752	398.757752	398.757752	-1 189 929.191
5	399.818758	399.818758	399.818758	-1 190 227.485
6	399.973557	400.672860	400.672860	-1 190 432.182
7	399.996142	401.526962	401.526962	-1 190 630.231
8	399.999437	402.381064	402.381064	-1 190 827.322
9	399.999918	403.235166	403.235166	-1 191 024.282
10	399.999988	404.089268	404.089268	-1 191 221.232
11	399.999998	404.943370	404.943370	-1 191 418.190
12	400.000000	405.791472	405.791472	-1 191 615.157
13	400.000000	406.651574	406.651574	-1 191 812.133
14	400.000000	657.201041	657.201041	-1 231 550.039

Table 5.6: Path connecting θ^0 to θ^*.

- The index condition (IND) holds since $\mathcal{P}^<(\theta) = \mathcal{P}$ for all $\theta \in I$.
- The whole region I is stable since Slater's condition holds (in x) throughout I.
- All functions f^i, $i \in \mathcal{P}$ are continuously differentiable with respect to θ, for $\theta \in I$.

The initial input is $\theta^0 = (0, 0, 0)^T$. The algorithm described in the previous chapter was applied to this model. A complete listing of the program used appears in Appendix B. The variable `MGRITER` was set so that the Method of Golden Rule only iterates three times in order to choose α, but this maximum only applies if the change in $\tilde{f}(\theta)$ is greater than $10^{-3}\%$. The results of the program appear in Table 5.6. The program was run using Turbo-Pascal 3.0 on an IBM compatible and

STYLE	QUANTITY (YARDS)
B	0.00
P_1	5 000.00
P_2	181 403.46
P_3	5 000.00
P_4	50 000.00
D_1	600 000.00
D_2	227 636.36

Table 5.7: Optimal production plan corresponding to an optimal input θ^*.

the run time was approximately 77.4 seconds. The optimal input for this linear model is $\theta^* = (400, 657.201, 657.201)^T$ and we have $\tilde{f}(\theta^*) = -1\,231\,550.039\$$, an improvement of 20.5% over the initial set-up with θ^0! The corresponding optimal production rate is given in Table 5.7. The corresponding dual optimal solution is $\tilde{u}_3 = 0.0800,\ \tilde{u}_8 = 0.5067,\ \tilde{u}_{11} = 0.0496,\ \tilde{u}_{15} = 0.2889,\ \tilde{u}_{20} = 0.3611,\ \tilde{u}_i = 0,\ i = 1, 2, 4, 5, 6, 7, 9, 10, 12, 13, 14, 16, 17, 18, 19.$

The numerical optimization method used to compute these values only terminates if no further improvement from the current θ is possible, i.e., only if the current θ is a locally optimal input. Hence the θ^* given above is a locally optimal input. We would, however, like to verify this optimality (i.e., that (L, θ^*) with θ^* given above, is an optimal realization of the model) and ascertain that the program terminated properly. To do this, we can apply Theorem 3.7 and find a suitable non-negative vector function $\mathcal{U}(\theta)$. Note that for this case, $\mathcal{P}^= = \emptyset$, since Slater's condition holds for all $\theta \in I$ and that $\mathcal{P}(\tilde{x}(\theta^*)) = \{3, 8, 11, 15, 17, 20, 21\}$. The set I corresponds to the region of stability $W(\theta^*)$; hence, as affirmed in [19], there exists a *continuous*

non-negative vector function. We first solve the BBZ system (this is equivalent to the Karush-Kuhn-Tucker condition since $\mathcal{P}^{=} = \emptyset$) which yields $u_3 = 0.0800$, $u_8 = 0.5067$, $u_{15} = 0.2889$, $u_{17} = 0$, $u_{20} = 0.3611$, $u_{21} = 0.1333$, $u_i = 0$, $i \in \mathcal{P} \setminus \mathcal{P}(\tilde{x}(\theta^*))$ and

$$u_{11} = 0.2338 - \frac{202.67}{700 + \theta_1}.$$

Letting $\mathcal{U}_k(\theta) = u_k$, $k \in \mathcal{P}(\tilde{x}(\theta^*))$, we get the necessary vector function. The left-hand inequality of the saddle-point inequality is easily satisfied with this $\mathcal{U}(\theta)$. The right-hand inequality becomes

$$-1\,231\,550.039 \leq -1\,507\,913.7 + \frac{304\,000\,000}{700 + \theta_1}$$

which is true for all $\theta \in I$. This proves that θ^* is indeed a locally optimal input.

From the marginal value formula, we can derive

$$\frac{\partial \tilde{f}}{\partial b_i} = \tilde{u}_i.$$

Therefore, our expression $\mathcal{U}_{11}(\theta)$ indicates how $\tilde{f}$ varies with changes in b_{11}.

To experiment with the computer program and the algorithm, the program in Appendix B was run several times in different versions. For each version we noted the optimal input θ^*, the optimal value $\tilde{f}(\theta^*)$, the number of θ-iterations, the total number of MGR-iterations, and the run-time for the computer program. Each version yielded the same $\tilde{x}(\theta^*)$ and $\tilde{f}(\theta^*)$, but different optimal inputs θ^*! The first type of variation was a change in θ^0. The program was run with $\theta^0 = (150, 25, 400)^T$ and with $\theta^0 = (50, 400, 800)^T$. When the initial input was $\theta^0 = (150, 25, 400)^T$, the results were $\theta^* = (400, 331.9666, 706.9666)^T$, 14 θ iterations, 39 MGR-iterations

and a run-time of 114.1 seconds. With $\theta^0 = (50, 400, 800)^T$, we obtained $\theta^* = (400, 570.8204, 970.8204)^T$, 10 θ-iterations, 53 MGR-iterations and a run-time of 112.8 seconds. Although these optimal inputs differ greatly, the corresponding optimal solutions are the same! *This means that vastly different production rates for the starching operation have the same effect on the optimal production plan!* So our three initial θ^0's yield three different optimal realizations of the same model, but with the same optimal value. The optimal value function $\tilde{f}(\theta)$ was evaluated at 40 different points on the line joining $\theta^{*1} = (400, 657.2010, 657.2010)^T$ and $\theta^{*2} = (400, 570.8204, 970.8204)^T$ and at each point the optimal value was the same, i.e., -1 231 550.039$.

Another variation in the program is changing the program parameter `MGRITER` which indicates the maximum number of MGR-iterations to be performed if the change in $\tilde{f}(\theta)$ was greater than 0.001%. In the program, `MGRITER` is set to two less than the maximum number of iterations desired, i.e., if we want only three MGR-iterations performed, then we set `MGRITER` = 1. The program was run for `MGRITER` = 1, 6, 11, 16, and 21. Each time the program was run, the initial input was $\theta^0 = 0$, the optimal value was $\tilde{f}(\theta^*) = -1\,231\,550.039\$$, and θ^* was such that $\theta_1^* = 400$ and $\theta_2^* = \theta_3^*$ and the optimal solution $\tilde{x}(\theta^*)$ was the same. The results appear in Table 5.8. These five runs were repeated with the alternate method for choosing d, described in the previous chapter. The results of these runs are given in Table 5.9. It appears that the original method for choosing d is faster. This makes sense since at each θ-iteration, d is chosen by solving a linear program, hence it is optimal and indicates a sort of steepest descent direction. The alternate version (as

MGRITER	θ_3^*	NUMBER OF θ-ITERATIONS	NUMBER OF MGR-ITERATIONS	RUN-TIME (SECONDS)
1	657.2010	14	39	77.4
6	589.2992	8	56	93.1
11	584.3077	6	65	104.1
16	578.9910	5	72	113.9
21	578.9905	4	69	110.2

Table 5.8: Results of the numerical method for different values of MGRITER.

MGRITER	θ_3^*	NUMBER OF θ-ITERATIONS	NUMBER OF MGR-ITERATIONS	RUN-TIME (SECONDS)
1	578.1332	14	65	122.1
6	578.1332	9	69	119.9
11	578.1332	7	73	122.3
16	578.1332	5	82	131.3
21	578.1332	4	74	119.1

Table 5.9: Results of the numerical method (using the alternate method for choosing d) for different values of MGRITER.

noted earlier) chooses d according to a simplistic rule which yields only a feasible direction of descent, not necessarily the best one. It also seems, at least for this case, that a small value for **MGRITER** is better than a large one. This means that it is not necessary to obtain an optimal α to determine the next θ. One therefore should seek the best direction for decreasing $\tilde{f}(\theta)$ and then not be too demanding of the distance travelled along that direction. This seems to be true for this particular case and for all but the last few iterations.

To further experiment with the numerical method, we altered the *model* for this case and ran the program again. For this new model we chose new coefficients to perturb and a new set I. The coefficients affected now are a_{86}, a_{87}, $a_{13,4}$ and $a_{13,5}$. They become

$$a_{86} = \frac{1}{700 + \theta_1}, \; a_{87} = \frac{1}{800 + \theta_2}, \; a_{13,4} = \frac{1}{1\,600 + \theta_3}, \; a_{13,5} = \frac{1}{1\,500 + \theta_4}$$

and we define I by

$$L = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad U = \begin{bmatrix} 550 \\ 300 \\ 400 \\ 475 \end{bmatrix}.$$

The program was run three times with this set-up. Each time, we had **MGRITER** = 1, $\tilde{f}(\theta^*) = -1\,121\,799.578\$$, and the same solution $\tilde{x}(\theta^*)$ (given in Table 5.10). With an initial input of $\theta^0 = 0$, the optimal input was $\theta^* = (391.4855, 49.8447, 400, 475)^T$. With $\theta^0 = (125, 40, 210, 180)^T$, we obtained $\theta^* = (287.2794, 202.2794, 400, 475)^T$ and with $\theta^0 = (250, 100, 10, 400)^T$, we obtained $\theta^* = (432.6010, 282.6010, 400, 475)^T$. The number of θ-iterations was 18, 18, and 19 respectively. The number of MGR-iterations was 77, 77, and 80 respectively, and the run-times were 171.7, 175.6, and 132.0 seconds respectively.

STYLE	QUANTITY (YARDS)
B	0.00
P_1	5 000.00
P_2	5 000.00
P_3	100 000.00
P_4	50 000.00
D_1	600 000.00
D_2	178 691.98

Table 5.10: Optimal production plan corresponding to an optimal input θ^*.

Chapter 6

Case Study II: The Evangeline Coffee Company

The second case study is the Evangeline Coffee Company of New Orleans, Louisiana[1]. It is based on the case of the same name which appears in Naylor, et al. [17]. The case presented there is solved by a linear program where the matrix A is 36×36. Performing the numerical method on this system would have taken a few hours to run on a microcomputer. Since the primary objective in this thesis is to show that numerical applications of input optimization are possible, it is felt that studying a small subsystem is sufficient for our purposes. Therefore, some constraints and variables which appear in the case by Naylor, et al., do not appear here. We present only the aspects of the case which are needed.

This company imports, blends, and roasts green coffees (coffees which have not been roasted yet) for distribution in a six-state area along the Gulf Coast. The

[1]The company's name and the pertinent figures were again changed to maintain the anonymity of the firm.

Café d'Élite		Plantation	
Strength	≤ 8.0	Strength	≤ 7.0
Acidity	≤ 3.5	Acidity	≤ 4.0
Caffeine	≤ 2.8	Caffeine	≤ 2.2
Hardness	≤ 2.5	Hardness	≤ 3.0

Table 6.1: Requirements for each blend. Adapted from Naylor, et al., [17] p.176.

company markets two different blends of coffee under the brand names: Café d'Élite and Plantation. The former is a high quality blend served exclusively by leading hotels, fine restaurants, and espresso houses. The latter is a medium quality blend with a widespread distribution in the six-state area. These blends are created by using five different green coffees: Santos 4's, Bourbon Santos, Rios, Victorias, and Medellins. Seven characteristics describe each of the two blends; however, only four are considered here: strength, acidity, caffeine, and hardness. Each green coffee has been assigned an index number between 0 and 10 for each characteristic to quantify that characteristic. For example, a relative strength of 8 or 9 indicates a "very strong" taste whereas 1 or 2 indicates a "weak" coffee. The acidity is determined by measuring the pH factor, and caffeine content is measured as a percentage of weight. It has been found that these taste characteristics of different green coffees combine linearly when blended.

The two coffee blends have set requirements for each characteristic. These requirements are outlined in Table 6.1. The company forecasts its green coffee needs one month in advance and then orders the appropriate quantities. These green coffees vary according to price, quantity available, and taste characteristics, as indi-

Green Coffees Available	Price per Pound	Available Supply in Pounds	Strength Index	Acidity *p*H	Percent Caffeine Content	Hardness Index
Santos 4's	$0.35	25 000	6	4.0	1.8	2
Bourbon Santos	0.36	10 000	6	3.9	1.6	3
Rios	0.20	75 000	10	4.5	1.0	7
Victorias	0.17	50 000	10	5.0	0.9	8
Medellins	0.44	5 000	8	3.0	3.0	2

Table 6.2: Adapted from Naylor, et al., [17] p.177.

cated in Table 6.2. The demand for the coming month has been estimated at 5 000 pounds of Café d'Élite and 20 000 pounds of Plantation.

We denote each variable by X_{ij}, $i = 1,\ldots,5$, $j = 1,2$ which represents the quantity of the i^{th} green coffee used in the j^{th} blend. There are three types of linear restraints: demand, supply, and quality. Demand restraints have the form

$$\left.\begin{array}{l} \sum_{i=1}^{5} X_{ij} \leq D_j \\ \sum_{i=1}^{5} -X_{ij} \leq -D_j \end{array}\right\} \; j = 1,2$$

where D_j is the demand for the j^{th} blend. Supply restraints are imposed by the quantity available of each green coffee. They may be expressed as

$$X_{i1} + X_{i2} \leq S_i, \quad i = 1,\ldots,5$$

where S_i is the number of available pounds of green coffee i. The quality restraints are derived from the values in Tables 6.1 and 6.2. They are of the form

$$\sum_{i=1}^{5} a_{ik} X_{ij} \leq D_j b_{jk}, \quad j = 1,2, \; k = 1,2,3,4$$

where

a_{ik} = kth quality characteristic of the ith green coffee
D_j = demand for the jth blend
b_{jk} = kth characteristic of the jth blend.

Max	$-35X_{11}$	−	$36X_{21}$	−	20_{31}	−	17_{41}	−	44_{51}									
					$-35X_{12}$	−	36_{22}	−	20_{32}	−	17_{42}	−	44_{52}					
s.t.	X_{11}	+	X_{21}	+	X_{31}	+	X_{41}	+	X_{51}							$\leq$		5 000
	$-X_{11}$	−	X_{21}	−	X_{31}	−	X_{41}	−	X_{51}							$\leq$		−5 000
					X_{12}	+	X_{22}	+	X_{32}	+	X_{42}	+	X_{52}			$\leq$		20 000
					$-X_{12}$	−	X_{22}	−	X_{32}	−	X_{42}	−	X_{52}			$\leq$		−20 000
	X_{11}			+	X_{12}											$\leq$		25 000
			X_{21}			+	X_{22}									$\leq$		10 000
					X_{31}			+	X_{32}							$\leq$		75 000
							X_{41}			+	X_{42}					$\leq$		50 000
									X_{51}			+	X_{52}			$\leq$		5 000
	$6X_{11}$	+	$6X_{21}$	+	$10X_{31}$	+	$10X_{41}$	+	$8X_{51}$							$\leq$		40 000
	$4X_{11}$	+	$3.9X_{21}$	+	$4.5X_{31}$	+	$5X_{41}$	+	$3X_{51}$							$\leq$		17 500
	$1.8X_{11}$	+	$1.6X_{21}$	+	X_{31}	+	$0.9X_{41}$	+	$3X_{51}$							$\leq$		14 000
	$2X_{11}$	+	$3X_{21}$	+	$7X_{31}$	+	$8X_{41}$	+	$2X_{51}$							$\leq$		12 500
					$6X_{12}$	+	$6X_{22}$	+	$10X_{32}$	+	$10X_{42}$	+	$8X_{52}$			$\leq$		140 000
					$4X_{12}$	+	$3.9X_{22}$	+	$4.5X_{32}$	+	$5X_{42}$	+	$3X_{52}$			$\leq$		80 000
					$1.8X_{12}$	+	$1.6X_{22}$	+	X_{32}	+	$0.9X_{42}$	+	$3X_{52}$			$\leq$		44 000
					$2X_{12}$	+	$3X_{22}$	+	$7X_{32}$	+	$8X_{42}$	+	$2X_{52}$			$\leq$		60 000
	X_{ij}	$\geq$	0															

Table 6.3: Linear program for the Evangeline Coffee Company.

These 17 restraints define the feasible set in the linear program presented in Table 6.3. The matrix A has dimension 17×10. In order to fit it onto one page, some columns overlap; however this shouldn't cause any problem in understanding the linear program. This program was solved using the revised simplex method computer program listed in Appendix A. The optimal value is 8 502.50\$. The optimal solution is given in Table 6.4. The dual optimal solution is $\tilde{u}_2 = 75.2$, $\tilde{u}_4 = 97.2$, $\tilde{u}_{11} = 9$, $\tilde{u}_{13} = 2.1$, $\tilde{u}_{14} = 1.8333$, $\tilde{u}_{15} = 1.2667$, $\tilde{u}_{17} = 0.2667$, $\tilde{u}_i = 0$, $i = 1, 3, 5, \ldots, 10, 12, 16$.

At this point we would like to apply input optimization and obtain a solution with a lesser cost. Suppose it were possible to alter a characteristic of a green coffee, by developping a hybrid plant. This would change the program and therefore

VARIABLE	QUANTITY (POUNDS)	VARIABLE	QUANTITY (POUNDS)
X_{11}	1 750	X_{12}	14 000
X_{21}	0	X_{22}	0
X_{31}	500	X_{32}	4 000
X_{41}	0	X_{42}	0
X_{51}	2 750	X_{52}	2 000

Table 6.4: Optimal solution for the initial set-up.

the optimal solution. Assume it is possible to grow a new plant which produces Santos 4's with any desired strength within 11.25% of the present strength, i.e., it is possible to get Santos 4's with a strength between 5.325 and 6.675. Similarly, assume that Rios and Medellins can be made to have an acidity in the range $\pm 8.89\%$ and $\pm 16.67\%$ respectively, i.e., between 4.1 and 4.9 for Rios and between 2.5 and 3.5 for Medellins. Finally, suppose that an acidity index of less than or equal to 4.1 is acceptable for the Café d'Élite blend. Implementing these conditions means changing $a_{11,5}$, $a_{15,10}$, b_{11}, $a_{10,1}$, $a_{14,6}$, $a_{11,3}$, and $a_{15,8}$ to the following:

$$a_{11,5} = a_{15,10} = 3 + \theta_1, \quad b_{11} = 5\,000(3.5 + \theta_2),$$

$$a_{10,1} = a_{14,6} = 6 + \theta_3, \quad a_{11,3} = a_{15,8} = 4.5 + \theta_4$$

where $\theta \in I = \{\theta \in R^4 : L \leq \theta \leq U\}$ and where

$$L = \begin{bmatrix} -0.500 \\ 0.000 \\ -0.675 \\ -0.400 \end{bmatrix}, \qquad U = \begin{bmatrix} 0.500 \\ 0.600 \\ 0.675 \\ 0.400 \end{bmatrix}.$$

If we try to run the numerical method on this set-up, we discover that for some $\theta \in I$, the constraints are infeasible. We therefore must redefine I. On closer inspection of

the constraints we determine that if we redefine I with

$$L = \begin{bmatrix} -0.500 \\ -0.025 \\ -0.675 \\ -0.400 \end{bmatrix}, \quad U = \begin{bmatrix} 0.035 \\ 0.600 \\ 0.675 \\ 0.100 \end{bmatrix},$$

then the point $\hat{x} = (2\,250,\ 1,\ 1,\ 1,\ 2\,747,\ 18\,000,\ 1,\ 1,\ 1,\ 1\,997)^T$ is such that $\mathcal{P}^<(\hat{x}(\theta)) = \{5, \ldots, 27\}$ for all $\theta \in I$. Hence $I = W(\theta^*)$ ($\theta^* = (0,\ 0,\ 0,\ 0)^T$), which is a region of stability. Note that in this case study, for all $\theta \in I$, $\mathcal{P}^=(\theta) \neq \emptyset$. Hence to use the necessary theorems we must assure ourselves of the presence of a region of stability such as $W(\theta^*)$ (in Case study I, Slater's condition held for all $\theta \in I$, hence we knew the model was stable in I).

The program for the numerical method listed in Appendix B was used to solve Case study I. To use it for Case study II, one must modify certain segments of the program, in particular the lines which update $a_{i,j}(\theta)$, those which calculate $\nabla_\theta g(\theta)$, and other obvious lines of source code.

The marginal value theorem can be used here because:

- The model is bi-convex, with a realistic objective function.
- The saddle-point $\{\tilde{x}(\theta^*),\ \tilde{u}_i(\theta^*),\ i \in \mathcal{P}^<(\theta^*)\}$ is not necessarily unique but, as mentioned before, the values we obtain for $(\nabla g(\theta),\ \ell)$ will approximate the actual ones sufficiently.
- The index condition (IND) holds since $\mathcal{P}^<(\theta)$ is constant for all $\theta \in I$.
- $I = W(\theta^*)$ is a region of stability. The set I is also equal to $V_3(\theta^*)$ since $\mathcal{P}^=(\theta)$ is constant for all $\theta \in I$.

k	θ_1^k	θ_2^k	θ_3^k	θ_4^k	$\tilde{f}(\theta^k)$
1	0.000000	0.000000	0.000000	0.000000	8 502.500$
2	-0.427051	0.512461	-0.576519	-0.341641	8 116.013
3	-0.499980	0.599975	-0.576519	-0.399984	8 099.004
4	-0.500000	0.599975	-0.576519	-0.400000	8 099.000
5	-0.500000	0.599975	-0.576519	-0.400000	8 099.000

Table 6.5: Path connecting θ^0 to θ^*.

VARIABLE	QUANTITY (POUNDS)	VARIABLE	QUANTITY (POUNDS)
X_{11}	4 583.333	X_{12}	15 733.333
X_{21}	0.000	X_{22}	0.000
X_{31}	0.000	X_{32}	4 000.000
X_{41}	416.667	X_{42}	0.000
X_{51}	0.000	X_{52}	266.667

Table 6.6: Optimal solution for (P, θ^*).

- All functions f^i, $i \in \mathcal{P}$ are continuously differentiable with respect to θ.

The initial input was $\theta^0 = (0, 0, 0, 0)^T$. The variable MGRITER was set to 1 so that three MGR-iterations would be performed at each of the first few θ-iterations. The program gives a finite sequence $\{\theta^k\}$ which determines the path connecting θ^0 with the optimal input θ^*. These θ^k's are given in Table 6.5. The run-time was 54.4 seconds. The optimal input is $\theta^* = (-0.5, 0.599975, -0.576519, -0.4)^T$ and the optimal value is $\hat{f}(\theta^*) = 8\,099.00$\$, an improvement of 4.75%. The corresponding optimal solution is given in Table 6.6. The corresponding dual optimal solution is $\tilde{u}_2 = 41$, $\tilde{u}_4 = 64.76$, $\tilde{u}_{13} = 3$, $\tilde{u}_{15} = 6$, $\tilde{u}_{17} = 2.889$, $\tilde{u}_i = 0$, $i = 1, 3, 5, \ldots, 12, 14, 16$.

The primal solution indicates that Café d'Élite should be blended from Santos 4's and Victorias, whereas Plantation should be blended from Santos 4's, Rios, and Medellins.

To verify the optimality of θ^*, we use Theorem 3.7. Since the region of stability here is $W(\theta^*)$, we know from [19] that if θ^* is a locally optimal input then there exists a *continuous* non-negative vector $\mathcal{U}(\theta)$. We can solve the BBZ system to determine $\mathcal{U}(\theta)$. The polar set in the system is as follows (recall that $\mathcal{P}^=(\theta) = \{1,2,3,4\}$ for all $\theta \in I$).

$$\{\bigcap_{i\in\mathcal{P}^=(\theta)} D_i^=(\tilde{x})\}^+ = \{u \in R^{10} : u_1 = u_i,\ i = 2,3,4,5,\ u_6 = u_j,\ j = 7,8,9,10\}.$$

We also have

$$\nabla f^0(\tilde{x},\theta^*) + \sum_{i\in\mathcal{P}(\tilde{x}(\theta^*))\setminus\mathcal{P}^=} u_i \nabla f^i(\tilde{x},\theta^*) = \begin{bmatrix} 35 + 2u_{13} \\ 36 + 3u_{13} - u_{19} \\ 20 + 7u_{13} - u_{20} \\ 17 + 8u_{13} \\ 44 + 2u_{13} - u_{22} \\ 35 + 4u_{15} + 2u_{17} \\ 36 + 3.9u_{15} + 3u_{17} - u_{24} \\ 20 + (4.5 + \theta_4)u_{15} + 7u_{17} \\ 17 + 5u_{15} + 8u_{17} - u_{26} \\ 44 + (3 + \theta_1)u_{15} + 2u_{17} \end{bmatrix}$$

Solving this system determines $\mathcal{U}(\theta)$ to be as follows:

$$\mathcal{U}_{13}(\theta) = 3,\ \ \mathcal{U}_{15}(\theta) = \frac{9}{1-\theta_1},\ \ \mathcal{U}_{17}(\theta) = 3 - \frac{4.5 + 9\theta_4}{5(1-\theta_1)}$$

$$\mathcal{U}_{19}(\theta) = 4,\ \ \mathcal{U}_{20}(\theta) = 0,\ \ \mathcal{U}_{22}(\theta) = 9$$

$$\mathcal{U}_{24}(\theta) = 4 - \frac{9(1+\theta_4)}{5(1-\theta_1)},\ \ \mathcal{U}_{26}(\theta) = \frac{18(1-3\theta_4)}{5(1-\theta_1)}$$

$$\mathcal{U}_i(\theta) = 0,\ \ i \in \mathcal{P}^= \cup \mathcal{P} \setminus \mathcal{P}(\tilde{x}(\theta^*))$$

MGRITER	θ_2^*	θ_3^*	NUMBER OF θ-ITERATIONS	NUMBER OF MGR-ITERATIONS	RUN-TIME (SECONDS)
1	0.599975	-0.576519	5	51	54.6
6	0.592107	-0.666120	4	40	43.7
11	0.599288	-0.674199	4	45	48.2
16	0.599936	-0.674928	4	50	52.5
21	0.599994	-0.674993	3	39	41.8

Table 6.7: Results of the numerical method for different values of MGRITER.

This is a continuous function around θ^*. The left-hand inequality of the saddle-point inequality is easily satisfied with this function $\mathcal{U}(\theta)$. The right-hand inequality becomes

$$809\,900 \leq 807\,500 + 18\,000\frac{(1+2\theta_4)}{1-\theta_1}$$

which holds for all $\theta \in I$. Hence θ^* is a locally optimal input.

If we change the initial input so that $\theta^0 = (-0.25, -0.25, 0.40, 0.05)^T$, then the optimal input is found to be $\theta^* = (-0.5, 0.599965, -0.599720, -0.4)^T$ with $\tilde{f}(\theta^*) = -8\,099.00\$$ and the same optimal blend as the first time. The run-time was virtually the same as before, being 55.2 seconds.

As with the first case study, we varied the program variable **MGRITER**. For values of 1, 6, 11, 16, and 21 for **MGRITER**, the results were as outlined in Table 6.7. Note that $\tilde{f}(\theta^*) = -8\,099.00\$$, $\theta_1^* = -0.5$, and $\theta_4^* = -0.4$ in each case and that the optimal solution was always the same. The values in Table 6.7 indicate that, unlike Case study I, it seems a larger value for **MGRITER** yields a faster run-time. The conclusions we can draw regarding the value of **MGRITER** is that no one value is necessarily best;

each case seems to dictate which value is best for it.

Chapter 7

The Numerical Method for Nonlinear Programming and Questions of Stability

7.1 Nonlinear Programming

The numerical method described in Chapter 4 was designed for use on linear programs. For example, the *simplex method* solves each *linear* program, the path from one θ to the next is *linear*, etc. One might wonder if it can be used on nonlinear programs. Certainly the theory of input optimization holds for nonlinear programs (although most of the results are stated for convex models). It was mentioned in Chapter 4 that the algorithm's way of determining $\bar{\alpha}$ and α would require modifications for regions of stability more complex than a cube in R^p. Very often in nonlinear programming, the region of stability is not a simple cube. Thus we already have an obstacle to direct application of the method to nonlinear programs. However, some programs favour direct application and we will show here how input optimization

and this method can be applied in the nonlinear case.

A general nonlinear programming problem has a vector variable $z = (z_1, z_2, \ldots, z_{n'})$. Most other methods, when solving such a problem, consider each component of z equally. However, some nonlinear problems are such that redefining their vector variable $z \in R^{n'}$ as $z = (x,\theta)$ where $x \in R^n$, $\theta \in R^p$ and $n + p = n'$, transforms them into linear programming *models* (!) with an input parameter θ. For example, we could let $x_i = z_i$, $i = 1,\ldots,n$ and $\theta_j = z_j$, $j = n+1,\ldots,n'$ or some other combination. Both examples presented in this section are well suited for a redefinition of their vector variable z into vector variables x and θ. The first example is an unconstrained optimization problem which can be solved using an adapted numerical method. The advantage to solving a nonlinear optimization problem using input optimization techniques is that one obtains in the process, a stable path connecting θ^0 to θ^* *and* an optimal solution $\tilde{x}(\theta^*)$. When solving a problem using other techniques (even under ideal conditions), no indication is given as to whether it is possible, in practice, to get from an initial point $z^0 = (z_1^0, z_2^0, \ldots, z_{n'}^0)$ to an optimal $z^* = (z_1^*, z_2^*, \ldots, z_{n'}^*)$. Since the examples and cases in this work are derived from real-life situations, the notion of feasibly attaining an optimal z^* from an initial z^0 is of importance to the people/systems involved.

The first example is adapted from one presented by Hock and Schittkowski in [14] and originally from Colville [7]. It is an unconstrained minimization problem with the following objective function.

$$\begin{aligned} f(z) &= 100(z_2 - z_1^2)^2 + (1 - z_1)^2 + 90(z_4 - z_3^2)^2 + (1 - z_3)^2 \\ &\quad +10.1((z_2 - 1)^2 + (z_4 - 1)^2) + 19.8(z_2 - 1)(z_4 - 1) \end{aligned}$$

A possible "splitting" of z into x and θ is $z = (\theta_1, x, \theta_2, \theta_3)$. After rearranging and grouping certain terms, this redefines the objective function as

$$\begin{aligned} f^0(x,\theta) &= 110.1x^2 + (-40 - 200\theta_1^2 + 19.8\theta_3)x + (100\theta_1^4 + \theta_1^2 \\ &\quad -2\theta_1 + 90\theta_2^4 + \theta_2^2 - 180\theta_2^2\theta_3 + 100.1\theta_3^2 - 2\theta_2 - 40\theta_3 + 42) \end{aligned}$$

This problem has no constraints and the objective function is a simple quadratic function in x, for fixed θ. The numerical method of Chapter 4 is easily adapted to this problem and solved by hand (stability considerations can be ignored here). We define $\theta^0 = (-3, -3, -1)^T$ which determines

$$f^0(x,\theta^0) = 110.1x^2 - 1\,859.8x + 17\,222.1,$$

$\tilde{x}(\theta^0) \simeq 8.4660$ and $\tilde{f}(\theta^0) = 19\,192$. The gradient of $g(\theta)$ is calculated as

$$\nabla g(\theta) = \begin{bmatrix} -400\theta_1\tilde{x} + 400\theta_1^3 + 2\theta_1 - 2 \\ 360\theta_2^3 + 2\theta_2 - 360\theta_2\theta_3 - 2 \\ 19.8\tilde{x} - 180\theta_2^2 + 200.2\theta_3 - 40 \end{bmatrix}$$

Notice the absence of Lagrange multipliers u_i due to the absence of constraints in the problem. If we evaluate $\nabla g(\theta)$ at θ^0 we find an optimal direction d to be $d = (1, 1, 1)^T$. Examining the original function $f(z)$ we see that it vanishes at $z = (1, 1, 1, 1)^T$. Choosing $\alpha = 2$ (α is the distance travelled along the direction d) brings us closer to this point. Our new θ is therefore $\theta^1 = (-1, -1, 1)^T$, our quadratic function is

$$f^0(x,\theta^1) = 110.1x^2 - 220.2x + 118.1$$

and $\tilde{x}(\theta^1) = 1$, $\tilde{f}(\theta^1) = 8$. Evaluating $\nabla g(\theta)$ at θ^1 indicates d should be $d = (1, 1, 0)^T$. We choose $\alpha = 2$ and obtain $\theta^2 = (1, 1, 1)^T$. The quadratic function becomes

$$f^0(x,\theta^2) = 110.1x^2 - 220.2x + 110.1$$

and $\tilde{x}(\theta^2) = 1$, $\tilde{f}(\theta^2) = 0$. Evaluating $\nabla g(\theta)$ one more time determines d to be the zero vector. Thus $\theta^* = \theta^2$ and $\tilde{f}(\theta^*) = 0$. Our path from θ^0 to θ^* consists of the two straight lines joining θ^0 to θ^1 and θ^1 to θ^*. The program from Appendix B was adapted to solve this method but got "stuck" iterating from one side of θ^* to the other and back again (close to θ^*) indefinitely. The objective function seems to have a very small slope at θ^*, and would require a method with a quadratic convergence rate instead of linear (this is a typical behaviour of steepest descent type methods). Hence the solution was obtained by hand.

The second example is also adapted from one in Hock and Schittkowski [14] and is derived from a heat exchanger design problem. For more references regarding this problem, consult Avriel and Williams [1] and Dembo [9]. The objective function is $f(z) = z_4 + z_5 + z_6 + z_7 + z_8$ which is to be minimized subject to the constraints

$$
\begin{aligned}
&1 - 0.0025(z_4 + z_6) \geq 0 \\
&1 - 0.0025(z_5 + z_7 - z_4) \geq 0 \\
&1 - 0.01(z_8 - z_5) \geq 0 \\
&z_1 z_6 - 833.33252 z_4 - 100 z_1 + 83\,333.333 \geq 0 \\
&z_2 z_7 - 1\,250 z_5 - z_2 z_4 + 1\,250 z_4 \geq 0 \\
&z_3 z_8 - 1\,250\,000 - z_3 z_5 + 2\,500 z_5 \geq 0 \\
&100 \leq z_1 \leq 10\,000 \\
&1\,000 \leq z_i \leq 10\,000\,, \quad i = 2,3 \\
&10 \leq z_i \leq 1\,000\,, \quad i = 4,\ldots,8
\end{aligned}
$$

We can split z so that $z = (\theta_1, \theta_2, \theta_3, x_1, x_2, x_3, x_4, x_5)^T$ and the nonlinear program

VARIABLE	VALUE
x_1	181.9289
x_2	295.5600
x_3	217.7137
x_4	286.3692
x_5	395.5600

Table 7.1: Optimal solution with θ^0.

becomes the following *linear model.*

$$
\begin{aligned}
Min \quad & f^0(x,\theta) = x_1 + x_2 + x_3 + x_4 + x_5 \\
s.t. \quad & f^1(x,\theta) = 0.0025x_1 + 0.0025x_3 - 1 \leq 0 \\
& f^2(x,\theta) = -0.0025x_1 + 0.0025x_2 + 0.0025x_4 - 1 \leq 0 \\
& f^3(x,\theta) = -0.01x_2 + 0.01x_5 - 1 \leq 0 \\
& f^4(x,\theta) = 833.33252x_1 - \theta_1 x_3 - 83\,333.333 + 100\theta_1 \leq 0 \\
& f^5(x,\theta) = -1250x_1 + \theta_2 x_1 + 1250x_2 - \theta_2 x_4 \leq 0 \\
& f^6(x,\theta) = -2500x_2 + \theta_3 x_2 - \theta_3 x_5 + 1\,250\,000 \leq 0 \\
& f^i(x,\theta) = -x_i + 10 \leq 0 \quad , \quad i = 7,\ldots,11 \\
& f^j(x,\theta) = x_i - 1\,000 \leq 0 \quad , \quad j = 12,\ldots,16 \\
& \theta \in I
\end{aligned}
$$

where I is defined by

$$
L = \begin{bmatrix} 100 \\ 1\,000 \\ 1\,000 \end{bmatrix}, \text{ and } U = \begin{bmatrix} 10\,000 \\ 10\,000 \\ 10\,000 \end{bmatrix}
$$

The gradient of $g(\theta)$ is

$$
\nabla g(\theta) = \begin{bmatrix} (100 - x_3)u_4 \\ (x_1 - x_4)u_5 \\ (x_2 - x_5)u_6 \end{bmatrix}
$$

The set I is not a region of stability. For example, if $\theta = (579, 1\,359, 5\,110)^T$, the corresponding program is infeasible. However with $\hat{\theta} = (100, 1\,360, 5\,111)^T$, the model is stable for all $\hat{\theta} \leq \theta \leq U$. The initial input was chosen to be $\theta^0 = (580, 1\,360, 5\,111)^T$, where $\tilde{f}(\theta^0) = 1\,377.13174$ and $\tilde{x}(\theta^0)$ is given in Table 7.1. The

k	θ_1^k	θ_2^k	θ_3^k	$\bar{f}(\theta^k)$
1	580.00000	1 360.00000	5 111.00000	1 377.13174
2	4 755.70451	5 535.70451	9 286.70451	488.05707
3	4 042.43654	6 248.97248	9 999.97248	419.45206
4	4 041.81349	6 249.59554	9 999.98963	419.44595
5	4 040.92686	6 250.48216	9 999.99882	419.43887
6	4 039.95938	6 251.44964	9 999.99996	419.43013
7	4 039.03832	6 253.70709	10 000.00000	419.42287
8	291.41229	9 999.99674	10 000.00000	341.24998
9	291.41228	9 999.99999	10 000.00000	341.24997
10	291.41228	10 000.00000	10 000.00000	341.24997

Table 7.2: Path connecting θ^0 to θ^*.

numerical method solved this problem in 214.0 seconds, performing 10 θ-iterations and 122 MGR-iterations. The program variables, described in Chapter 4, were set as follows: MGRITER = 1, PERCENT = 0.0001%, $\varepsilon = 5 \times 10^{-6}$, $\varepsilon' = 10^{-5}$, and $\delta = 10^{-5}$. The path connecting θ^0 to $\theta^* = \theta^{10}$ consists of the straight-line segments joining θ^i to θ^{i+1}, $i = 0, \ldots, 9$. These inputs appear in Table 7.2. The final solution was $\tilde{x}(\theta^*) = (10, 99.99998, 10, 21.25, 199.99999)^T$.

A direct application of the numerical method to this problem was possible because the nonlinear program was easily transformed into a linear input optimization model. Here we have demonstrated how input optimization can be used to solve usual nonlinear (highly non-convex) programs.

7.2 A Few Words of Caution Regarding Stability

The importance of stability in input optimization is vital. One cannot use input optimization theory to get reliable results for a problem without having information on instability, and if so, where (in the parameter space R^p) such instability occurs. It is not obvious when looking at most problems which perturbations of the model are stable and which lead to instability. One must actually work with a model in order to establish this. We mention here two situations in which instability can arise.

The first is in multi-objective programs such as a lexicographic problem. As an example, we look at Case study I as a lexicographic problem. Suppose our first objective, f^{0^1}, is to maximize profitability (which is the only objective in Case study I), and that our second, f^{0^2}, is to minimize wear and tear on the machines by minimizing the total number of yards of material produced. Then, finding an optimal input and optimal realization of the model for the first objective function leads to $\theta^* = (400, 657.201, 657.201)^T$ and $\tilde{f}(\theta^*) = -1\,231\,550.039$. The optimal solution $\tilde{x}(\theta^*)$ is degenerate and not unique. We therefore add the constraint

$$f^{0^1}(x, \theta^*) \leq \tilde{f}^1(\theta^*),$$

and the new objective function

$$f^{0^2}(x, \theta^*) = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7.$$

Now, we know that $21 \in \mathcal{P}^=(\theta)$, where $f^{21}(x, \theta) = f^{0^1}(x, \theta) - \tilde{f}^1(\theta^*) \leq 0$. From θ^*, any perturbations where $\theta < \theta^*$ are unstable since we would have $f^{21}(x, \theta) > 0$,

whereas $\theta \geq \theta^*$ ensures stability. This is obvious from the fact that θ^* is an *optimal* input for $f^{O^1}(x,\theta)$. Hence one must be careful before using input optimization on such models. Note that in this "lexicographic optimization", Slater's condition is never satisfied!

The second example demonstrating instability is a simple nonlinear program. Consider the program

$$\begin{aligned} Min \quad & x_3 + x_4 + x_5 \\ s.t. \quad & x_1 + x_2 + x_3 \leq 1 \\ & -x_1 + x_2 + x_3 \leq 1 \\ & x_1 - x_4 x_2 \leq 0 \\ & -x_1 - x_5 x_2 \leq 0 \\ & x_4 \leq 1 \\ & -x_4 \leq 1 \\ & x_2 \geq 0,\ x_3 \geq 0,\ x_5 \geq 0 \end{aligned}$$

If we let $\theta_1 = x_4$ and $\theta_2 = x_5$, then we get the linear model:

$$\begin{aligned} Min \quad & x_3 + \theta_1 + \theta_2 \\ s.t. \quad & x_1 + x_2 + x_3 - 1 \leq 0 \\ & -x_1 + x_2 + x_3 - 1 \leq 0 \\ & x_1 - \theta_1 x_2 \leq 0 \\ & -x_1 - \theta_2 x_2 \leq 0 \\ & -x_2 \leq 0 \\ & -x_3 \leq 0 \\ & \theta \in I \end{aligned}$$

where

$$I = \left\{ \theta \in R^2 : \begin{bmatrix} -1 \\ 0 \end{bmatrix} \leq \theta \leq \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

If $\theta^0 = (1, 1)^T$, then the feasible set $F(\theta^0)$ is a nice convex set in R^3, $\tilde{F}(\theta^0)$ is a square surface in the $x_3 = 0$ plane and $\tilde{f}(\theta^0) = 2$ (see Figure 7.1). To obtain a better realization of this model we let $\theta^* = \theta^0 + \alpha d$, $\alpha > 0$, $d = (-1, -1)^T$. When $\alpha = 1$, $\theta^* = (0, 0)^T$, $F(\theta^*)$ has been reduced to a triangular surface in the

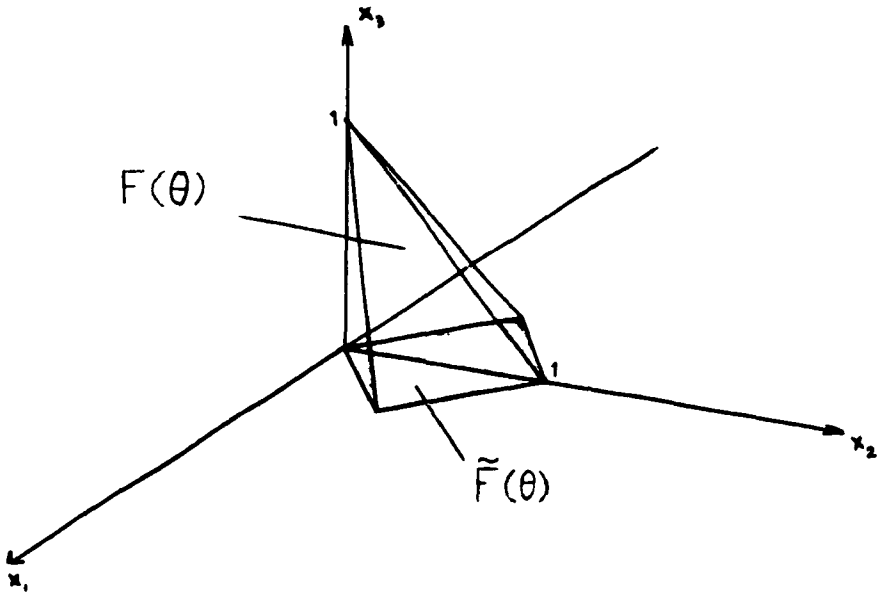

Figure 7.1: $\theta^0 = (1, 1)^T$.

$x_1 = 0$ plane, $\tilde{F}(\theta^*)$ is the line segment $\{x : x_1 = x_3 = 0,\ 0 \leq x_2 \leq 1\}$ and $\tilde{f}(\theta^*) = 0$ (see Figure 7.2). To further improve this model requires decreasing θ_1 from 0 and maintaining θ_2 at 0. But for all θ with $\theta_1 < 0$, $\theta_2 = 0$, the feasible set is simply the line segment $\{x_1 = x_2 = 0,\ 0 \leq x_3 \leq 1\}$ and $\tilde{F}(\theta) = \{(0, 0, 0)^T\}$ (see Figure 7.3). The feasible set did not change continuously when θ went from $(0, 0)^T$ to $(\gamma, 0)^T$, $\gamma < 0$. For $\theta^* = (0, 0)^T$, $F(\theta)$ is not lower semi-continuous in a neighbourhood of θ^*. The definition of lower semi-continuity is contradicted here. Using the notation of the definition in Chapter 2, let $\mathcal{A} = \{x : \|x-(0, \frac{1}{4}, \frac{1}{4})\|_2 < \frac{1}{16}\}$. Then $\mathcal{A} \cap F(\theta^*) = \{x : x_1 = 0, (x_2 - \frac{1}{4})^2 + (x_3 - \frac{1}{4})^2 < \frac{1}{16}\}$ but $\mathcal{A} \cap F(\theta) = \emptyset$ for all θ such that $\theta_1 < 0$, $\theta_2 = 0$. Thus $F(\theta)$ is not lower semi-continuous at θ^* and $S(\theta^*) = \{\theta \in I : \theta \geq 0\}$. Therefore this model is unstable at $\theta^* = 0$.

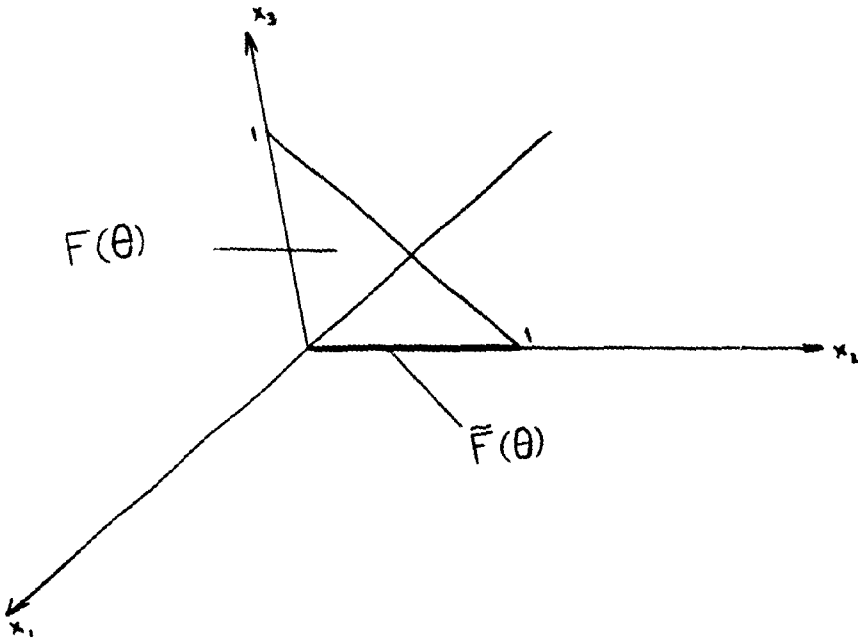

Figure 7.2: $\theta^* = (0, 0)^T$.

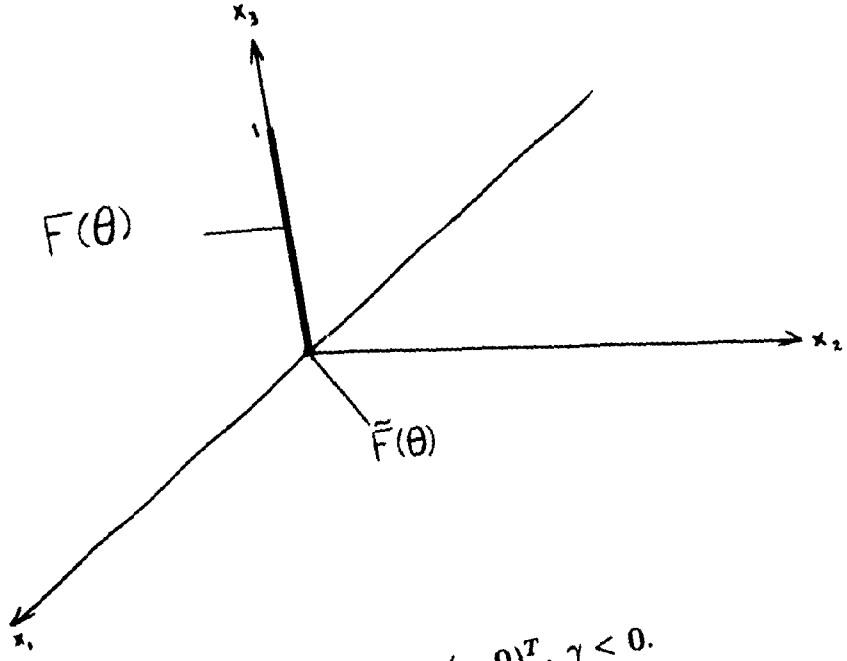

Figure 7.3: $\theta = (\gamma, 0)^T$, $\gamma < 0$.

Chapter 8

Conclusion

The main objective of this thesis was to explore the applicability of input optimization to real-life situations and to formulate a numerical method for solving linear input optimization problems. The numerical method of Chapter 4 has accomplished this. The case studies on which it was used satisfied the conditions of the theorems presented in Chapter 3.

Many so-called "real-life situations" do not fit the necessary mould for the method to work. Mainly, it is not always possible to have a cubic region of stability in R^p. Regions of stability, especially for large models, are typically included in a set which has "holes" or "patches" of instability. Therefore if the method iterates from the current θ along a certain direction d, it is not assured that every point along that direction is in the region of stability. The step-size part in the method would have to be altered to detect such "holes". These regions of instability also present the following possibility. If a region of instability separates a globally optimal θ^* from our current θ, it may be impossible (in a stable sense) to attain θ^*

from our current position. We may have to settle for another θ^* which is a locally optimal input and yields a worse realization of the model than the unattainable θ^*. It is possible that this "unattainable" θ^* is attainable from some other θ^0. An optimal input θ^* is indeed very much a function of the initial input θ^0. One should also keep in mind that our marginal value formula is not valid in every region of stability.

Another modification to the method could be the choice of the path ℓ from one θ to the next. In every example and case presented here, ℓ was a straight-line direction. Other possibilities include moving from one θ to the next along a parabola; or a combination of straight-line segments and parabolæ, whichever is best depending on the current θ.

A method which employs second order input optimization results may be faster than this one. The speed of convergence may be accelerated if some sort of quasi-Newton method for solving input optimization problems is formulated, i.e., one which takes into consideration the curvature of the functions involved. Unfortunately, one should keep in mind that the optimal value function $\tilde{f}(\theta)$ is not analytically given, so it is not clear at the present time how to use here the efficient numerical methods of non-smooth analysis (e.g., [28]). Moreover, even the second order optimality conditions over arbitrary regions of stability are presently nonexistent. Finding these conditions and their numerical implementations is one of the directions of future research.

Appendix A

General Program for the Revised Simplex Method

Included here is a computer program listing which performs the revised simplex method. The implementation, terminology, and notation are that used by Chvátal in the first seven chapters of [6]. The implementation was modified slightly. Chvátal enumerates many measures for ensuring numerical accuracy of the method. These were originally included in this program but later dropped because they slowed down the program unnecessarily. All the problems considered in this research were too small to warrant these efficiency measures. The program included here is tailored to these problems. In order to be accurate in treating large models, one should include a procedure to perform an Eta Factorization of the basis (if $B_0 \neq I$—the identity matrix), i.e., determine eta matrices E_i, $i = 1, \ldots, k$, such that $B_k = B_0 E_1 E_2 \cdots E_k$. One should also include a procedure to perform a triangular factorization of the initial basis B_0, i.e., determine permutation matrices P_i, $i = 1, \ldots, m$ and lower

triangular eta matrices L_i, $i = 1, \ldots, m$, such that

$$L_m P_m \cdots L_1 P_1 B_0 = U.$$

To run, the program needs as input the number of constraints m, the number of variables n, the vectors c and b, and the matrix A (all in that order). The last item input is the letter 's', 't', or 'v', which determine the mode to be silent, testing, or verbose respectively. In the silent mode, only the final basis, solution, and optimal value are output. When the mode is testing, at each iteration the values of the n decision variables are output. The verbose mode prints the value of each variable (decision, slack, and artificial) as well as the variables leaving and entering the basis.

The simplex method called by the program in Appendix B is essentially the same as this one except that no mode need be specified and no output comes directly from the simplex method (solutions are passed back to the other program).

```
PROGRAM SIMPLEX (INPUT,OUTPUT);

{
    Performs the revised simplex method on a linear program of the form

                    Max (c,x) s.t. Ax<=b, x>=0
}

CONST   FILE1='MILLSIM.INP';
        FILE2='MILLSIM.OUT';

TYPE
  MATRIX=ARRAY[1..30,1..40] OF REAL;
  DSOL=ARRAY[1..30] OF REAL;
  PSOL=ARRAY[1..40] OF REAL;
```

```
  BASES=ARRAY[1..30] OF INTEGER;
  ETAPTR=^ETA;
  ETA=RECORD
        NUM:INTEGER;
        POS:1..30;
        COLUMN:DSOL;
        NEXT:ETAPTR;
        PREV:ETAPTR;
      END;

VAR K,M,N:INTEGER;
    C:PSOL;
    B:DSOL;
    A:MATRIX;
    EPS1:REAL;
    MODE:CHAR;
    POIN,Q:ETAPTR;
    OFILE:TEXT;

{-----------------------------------------------------------------}

PROCEDURE MESSAG(MSG:INTEGER);

BEGIN
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
  CASE MSG OF
    1:WRITELN(OFILE,'===> PHASE I TERMINATED NORMALLY.  ENTERING PHAS',
                 'E II.');
    2:WRITELN(OFILE,'===> ORIGINAL PROBLEM IS INFEASIBLE');
    3:WRITELN(OFILE,'===> PROBLEM IS UNBOUNDED');
    4:BEGIN
        WRITE(OFILE,'===> ENTERING PHASE I.  THROUGHOUT PHASE I');
        WRITELN(OFILE,' VARIABLES ');
        WRITELN(OFILE,'     RENUMBERED AS FOLLOWS:');
        WRITELN(OFILE); WRITELN(OFILE,
                        '        ARTIFICIAL VARIABLE.........X(1)');
        WRITELN(OFILE,'         DECISION VARIABLES.........X(2) TO X(',
                    'N+1)');
        WRITE(OFILE,'         SLACK VARIABLES.............X(N+2)');
        WRITELN(OFILE,' TO X(N+M+1)');
        WRITELN(OFILE);
        WRITELN(OFILE,'     DECISION VARIABLES WILL BE GIVEN THEIR OR',
                    'IGINAL');
        WRITELN(OFILE,'     LABELS WHEN PHASE II IS ENTERED.');
```

```
      END;  { CASE 4 }
    5:WRITE(OFILE,'===> THE ABOVE SOLUTION IS OPTIMAL');
  END;  { CASE }
  WRITELN(OFILE); WRITELN(OFILE);
END;  { PROC MESSAG }


{-------------------------------------------------------------------}

PROCEDURE INIT;

{ Initialize these to zero }

VAR  I,J:INTEGER;

BEGIN
  FOR I:=1 TO 30 DO
    BEGIN
      B[I]:=0.0;
      FOR J:=1 TO 40 DO A[I,J]:=0.0;
    END;  { For i loop }
  FOR J:=1 TO 40 DO C[J]:=0.0;
END;  { proc init }

{-------------------------------------------------------------------}

PROCEDURE READIN;

VAR I,J:INTEGER;
    IFILE:TEXT;

BEGIN
  ASSIGN(IFILE,FILE1); RESET(IFILE);
  READLN(IFILE,M,N);
  K:=N+M;
  FOR J:=1 TO N DO READ(IFILE,C[J]);
  FOR I:=1 TO M DO READ(IFILE,B[I]);
  FOR I:=1 TO M DO FOR J:=1 TO N DO READ(IFILE,A[I,J]);
  READLN(IFILE);
  READLN(IFILE,MODE);
  CLOSE(IFILE);
END;  { proc readin }

{-------------------------------------------------------------------}
```

```
PROCEDURE PRINTDATA;

VAR I,J:INTEGER;

BEGIN
  WRITELN(OFILE,'   P R O B L E M  D A T A');
  WRITELN(OFILE,'   ----------------------');
  WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   NUMBER OF CONSTRAINTS:',M:4);
  WRITELN(OFILE,'   NUMBER OF VARIABLES:  ',N:4);
  WRITELN(OFILE);WRITELN(OFILE,'   VECTOR C:');
  FOR J:=1 TO N DO WRITELN(OFILE,C[J]:15:4);
  WRITELN(OFILE); WRITELN(OFILE,'   VECTOR B:');
  FOR I:=1 TO M DO WRITELN(OFILE,B[I]:15:4);
  WRITELN(OFILE);WRITELN(OFILE,'   MATRIX A:');
  WRITELN(OFILE); WRITELN(OFILE,'ROW #');
  FOR I:=1 TO M DO
    BEGIN
      WRITE(OFILE,I:3);
      FOR J:=1 TO N DO WRITE(OFILE,'   ',A[I,J]:7:3);
      WRITELN(OFILE);
    END;  { for i loop }
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
  CASE MODE OF
    's','S':BEGIN
              WRITELN(OFILE,'   MODE: SILENT');
              WRITELN(OFILE,'         ONLY THE FINAL BASIS, DICTIONAR',
                            'Y AND');
              WRITELN(OFILE,'         COST WILL BE PRINTED');
            END;  { case s }
    'v','V':BEGIN
              WRITELN(OFILE,'   MODE: VERBOSE');
              WRITELN(OFILE,'         FOR EACH PIVOT, THE ENTERING AN',
                            'D LEAVING');
              WRITELN(OFILE,'         VARIABLES AND THE NEW DICTIONAR',
                            'Y WILL BE');
              WRITELN(OFILE,'         PRINTED');
            END;  { case v }
    't','T':BEGIN
              WRITELN(OFILE,'   MODE: TESTING');
              WRITELN(OFILE,'         VALUES OF THE DECISION VARIABLE',
                            'S WILL BE');
              WRITELN(OFILE,'         PRINTED FOR EACH ITERATION');
```

```
            END;  { case t }
  END;  { case }
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   O U T P U T'); WRITELN(OFILE,'   ----------');
END;  { proc printdata }

{-------------------------------------------------------------------}

PROCEDURE GAUSS1(VAR Y,EI,V:DSOL; POS:INTEGER);

{  This procedure solves a system of the form yE=v by substitution
    where E is an eta-matrix }

VAR I:INTEGER;

BEGIN
  FOR I:=1 TO M DO Y[I]:=V[I];
  FOR I:=1 TO M DO
    IF I<>POS THEN Y[POS]:=Y[POS]-EI[I]*Y[I];
  Y[POS]:=Y[POS]/EI[POS];
END;  { proc gauss1 }

{-------------------------------------------------------------------}

PROCEDURE GAUSS2(VAR D,EI,V:DSOL; POS:INTEGER);

{ This procedure solves a system of the form Ed=v by substitution
   where E is an eta-matrix }

VAR I:INTEGER;

BEGIN
  D[POS]:=V[POS]/EI[POS];
  FOR I:=1 TO M DO
    IF I<>POS THEN D[I]:=V[I]-EI[I]*D[POS];
END;  { proc gauss2 }

{-------------------------------------------------------------------}

FUNCTION BASIC(INDEX:INTEGER; VAR BASIS:BASES):BOOLEAN;

{ returns true if x(index) is basic }

VAR I:INTEGER; BAS:BOOLEAN;
```

```
BEGIN
  I:=1; BAS:=FALSE;
  REPEAT
    IF INDEX=BASIS[I] THEN BAS:=TRUE;
    I:=I+1;
  UNTIL BAS OR (I>M);
  BASIC:=BAS;
END; { func basic }

{-----------------------------------------------------------------------}

PROCEDURE OUTPT(VAR X:PSOL; VAR BASIS:BASES; ENTER,LEAV:INTEGER);

{ produces the appropriate output depending on the value of mode }

VAR J:INTEGER;

BEGIN
  CASE MODE OF
    's','S': ;
    'v','V':BEGIN
              WRITELN(OFILE);WRITELN(OFILE);WRITELN(OFILE);
              WRITELN(OFILE);
              WRITELN(OFILE,'   ITERATION NO.     ',Q^.NUM:4);
              WRITELN(OFILE);
              WRITELN(OFILE,'   ENTERING VARIABLE:',ENTER:4);
              WRITELN(OFILE,'   LEAVING VARIABLE: ',LEAV:4);
              WRITELN(OFILE);
              WRITELN(OFILE,'   CURRENT DICTIONARY'); WRITELN(OFILE);
              WRITE(OFILE,'   VARIABLE         VALUE');
              WRITELN(OFILE,'              STATUS');
              FOR J:=1 TO K DO
                BEGIN
                  IF BASIC(J,BASIS) THEN
                    BEGIN
                      WRITE(OFILE,'   X',J:3,'           ',X[J]:18);
                      WRITE(OFILE,'   BASIC');
                      WRITELN(OFILE,' VARIABLE');
                    END
                  ELSE BEGIN
                       WRITE(OFILE,'   X',J:3,'           ',X[J]:18);
                       WRITELN(OFILE,'   NON-BASIC VARIABLE');
                       END;  { if ... then ... else }
```

```
                END;  { for j loop }
              WRITELN(OFILE); WRITE(OFILE,'------------------------');
              WRITELN(OFILE,'-------------------------------------',
                            '--------');
            END;  { case v }
    't','T':BEGIN
              WRITELN(OFILE);WRITELN(OFILE); WRITELN(OFILE);
              WRITELN(OFILE);
              WRITELN(OFILE,'   ITERATION NO.',Q^.NUM:4);
              WRITELN(OFILE); WRITELN(OFILE,'   DECISION VARIABLES');
              WRITELN(OFILE,'   VARIABLE          VALUE');
              FOR J:=1 TO N DO
                WRITELN(OFILE,'   X',J:3,'            ',X[J]:18);
              WRITELN(OFILE); WRITE(OFILE,'------------------------');
              WRITELN(OFILE,'-------------------------------------');
            END;  { case t }
    END;  { case }
END;  { proc outpt }

{-----------------------------------------------------------------}

FUNCTION VALU(VAR C,X:PSOL):REAL;

{ calculates value of objective function for current feasible vector x}

VAR VAL:REAL;
    CNT:INTEGER;

BEGIN
  VAL:=0.0;
  FOR CNT:=1 TO K DO VAL:=VAL+X[CNT]*C[CNT];
  VALU:=VAL;
END;  { func valu }

{-----------------------------------------------------------------}

PROCEDURE OPTIMAL(VAR C,X:PSOL; VAR U:DSOL; VAR BASIS:BASES;
                  MSG:INTEGER; PHASE1:BOOLEAN);

VAR I,J:INTEGER;
    OPTVAL:REAL;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE);
```

```
    WRITELN(OFILE,'   PRIMAL SOLUTION (BASIC AND NON-BASIC VARIABLES):');
    WRITELN(OFILE);
    FOR I:=1 TO K DO BEGIN
      WRITE(OFILE,'   X',I:3,'   ',X[I]);
      IF BASIC(I,BASIS) THEN WRITELN(OFILE,'   BASIC VARIABLE')
      ELSE WRITELN(OFILE,'   NON-BASIC VARIABLE');
    END;
    WRITELN(OFILE); WRITELN(OFILE);
    WRITELN(OFILE,'   DUAL SOLUTION:'); WRITELN(OFILE);
    FOR I:=1 TO M DO WRITELN(OFILE,'   U',I:3,'   ',U[I]);
    WRITELN(OFILE); WRITELN(OFILE);
    OPTVAL:=VALU(C,X);
    WRITELN(OFILE,'   VALUE OF OBJECTIVE FUNCTION:',OPTVAL:18);
    IF (MSG=2) OR (MSG=3) THEN
      BEGIN
        WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
        WRITE(OFILE,'===> THE ABOVE SOLUTION IS NOT OPTIMAL;');
        WRITELN(OFILE,' IT WAS THE CURRENT SOLUTION WHEN');
        WRITELN(OFILE,'     THE SIMPLEX METHOD WAS ABORTED.');
      END;  { if msg=... }
    IF MSG=5 THEN MESSAG(MSG);
  END;  { proc optimal }

  {-----------------------------------------------------------------}

  FUNCTION ENTVAR(VAR Y,U:DSOL; VAR C:PSOL; VAR BASIS:BASES;
                  RULE:CHAR):INTEGER;

  { rule g is greatest coefficient rule; rule l is least subscript }
  { rule; entvar determines the entering variable }

  VAR SUM:REAL;
      ENTER,CNT,I,J:INTEGER;



  BEGIN
    ENTER:=0;
    IF RULE='G' THEN
      BEGIN
        FOR I:=1 TO K DO
          BEGIN
            IF NOT(BASIC(I,BASIS)) THEN
              BEGIN
                SUM:=0.0;
```

```
              FOR CNT:=1 TO M DO SUM:=SUM+Y[CNT]*A[CNT,I];
              IF ((C[I]-SUM)>EPS1) AND (ENTER>0) THEN
                IF C[I]>C[ENTER] THEN ENTER:=I;
              IF ((C[I]-SUM)>EPS1) AND (ENTER=0) THEN ENTER:=I;
              IF I>N THEN U[I-N]:=-C[I]+SUM;
            END
          ELSE IF I>N THEN U[I-N]:=0.0;
        END;  { for i loop }
    END;  { if rule=g }
  IF RULE='L' THEN
    BEGIN
      I:=1;
      WHILE (ENTER=0) AND (I<K+1) DO
        BEGIN
          IF NOT(BASIC(I,BASIS)) THEN
            BEGIN
              SUM:=0.0;
              FOR CNT:=1 TO M DO SUM:=SUM+Y[CNT]*A[CNT,I];
              IF (C[I]-SUM)>EPS1 THEN ENTER:=I;
              IF I>N THEN U[I-N]:=-C[I]+SUM;
            END
          ELSE IF I>N THEN U[I-N]:=0.0;
          I:=I+1;
        END;  { while }
    END;  { if rule=l }
  ENTVAR:=ENTER;
END;  { func entvar }

{-------------------------------------------------------------------}

PROCEDURE ITERATE(VAR U:DSOL; VAR C,X:PSOL; VAR RULE:CHAR;
                  VAR BASIS:BASES; VAR LEAV,ENTER:INTEGER;
                  VAR CURVAL:REAL; VAR FINISH,PHASE1:BOOLEAN);

{ performs one iteration of the simplex method }

LABEL 10;

VAR T,OLDVAL:REAL;
    TEMPO,MSG,H,I,J:INTEGER;
    SUBS:BASES;
    POIN1,Q1:ETAPTR;
    UNSET,UNBDD:BOOLEAN;
    D,TEMP,V,Y:DSOL;
```

```
BEGIN
  FINISH:=FALSE;

  FOR J:=1 TO M DO Y[J]:=C[BASIS[J]];  { solve the system y*Bk=cB }
  Q1:=Q;
  WHILE Q1<>NIL DO
    BEGIN
      FOR J:=1 TO M DO V[J]:=Y[J];
      GAUSS1(Y,Q1^.COLUMN,V,Q1^.POS);
      Q1:=Q1^.PREV;
    END;

  ENTER:=ENTVAR(Y,U,C,BASIS,RULE);  { determine entering variable }

  IF ENTER=0 THEN                    { if optimal, break out of }
    BEGIN                            { procedure iterate        }
      IF NOT(PHASE1) THEN BEGIN
        MSG:=5;
        OPTIMAL(C,X,U,BASIS,MSG,PHASE1);
      END;
      FINISH:=TRUE; GOTO 10;
    END,  { if enter=0 }

  FOR J:=1 TO M DO D[J]:=A[J,ENTER]; { solve the system Bk*d=a }
  POIN1:=POIN;
  WHILE POIN1<>NIL DO
    BEGIN
      FOR J:=1 TO M DO V[J]:=D[J];
      GAUSS2(D,POIN1^.COLUMN,V,POIN1^.POS);
      POIN1:=POIN1^.NEXT;
    END;

  T:=0.0; UNSET:=TRUE;  { this section calculates largest possible t }
  FOR I:=1 TO M DO
    BEGIN
      IF D[I]>0.0 THEN TEMP[I]:=X[BASIS[I]]/D[I]
      ELSE TEMP[I]:=1.E20;

      IF ((TEMP[I]>=0.0)AND(TEMP[I]<>1.E20))AND UNSET THEN
        BEGIN
          T:=TEMP[I]; UNSET:=FALSE; SUBS[1]:=I; J:=1;
        END;  { if temp }
```

```
      IF (TEMP[I]=T) AND ((I<>1) AND NOT(UNSET)) THEN
        BEGIN
          SUBS[J+1]:=I; J:=J+1;
        END;  { if temp=t }

      IF (TEMP[I]<T) AND (TEMP[I]>=0.0) THEN
        BEGIN
          SUBS[1]:=I; J:=1; T:=TEMP[I];
        END;  { if temp>t }
    END;  { for i loop }

  UNBDD:=TRUE;
  FOR I:=1 TO M DO
    IF TEMP[I]<>1.E20 THEN UNBDD:=FALSE;

  IF UNBDD THEN       { i.e. problem is unbounded }
    BEGIN
      MSG:=3; MESSAG(MSG);
      OPTIMAL(C,X,U,BASIS,MSG,PHASE1);
      FINISH:=TRUE;
      GOTO 10;
    END;

{ now subs contains all subscripts i such that basis(i) is a candidate
  for leaving.  there are j such candidates.  now get smallest sub-
  script as leaving variable  }

  LEAV:=SUBS[1];
  FOR H:=2 TO J DO
    IF BASIS[LEAV]>BASIS[SUBS[H]] THEN LEAV:=SUBS[H];

{ now leav denotes the element of the array basis which contains the
  the subscript of the leaving variable }

  X[BASIS[LEAV]]:=0.0;   { leaving variable becomes non-basic }
  TEMP0:=BASIS[LEAV];
  BASIS[LEAV]:=ENTER;
  NEW(Q1);   { create new eta-column }
  FOR I:=1 TO M DO
    BEGIN
      IF I<>LEAV THEN X[BASIS[I]]:=X[BASIS[I]]-T*D[I]
      ELSE X[BASIS[I]]:=T;
      Q1^.COLUMN[I]:=D[I];
    END;  { for i loop }
```

```
  IF Q=NIL THEN
    BEGIN
      POIN:=Q1;
      Q1^.NUM:=1;
    END
  ELSE BEGIN
      Q1^.NUM:=Q^.NUM+1;
      Q^.NEXT:=Q1;
    END;  { else }
  Q1^.POS:=LEAV;
  Q1^.PREV:=Q;
  Q1^.NEXT:=NIL;
  Q:=Q1;
  OUTPT(X,BASIS,ENTER,TEMPO);

  OLDVAL:=CURVAL;  { if z* hasn't changed since last iteration then }
  CURVAL:=VALU(C,X);  { next iteration uses least subscript rule }
  IF OLDVAL=CURVAL THEN RULE:='L'
  ELSE RULE:='G';
10: END;  { proc iterate }

{-----------------------------------------------------------------}

PROCEDURE PH2(VAR PHASE1:BOOLEAN; VAR BASIS:BASES; VAR RULE:CHAR;
              VAR X:PSOL; VAR U:DSOL; VAR CURVAL:REAL;
              VAR FINISH:BOOLEAN);

VAR LEAV,ENTER,I,J:INTEGER;
    V:DSOL;

BEGIN
  IF NOT(PHASE1) THEN  { phaseI wasn't necessary so initialize phaseII}
    BEGIN
      FOR I:=1 TO M DO
        BEGIN
          BASIS[I]:=N+I;  { set the initial basis }
          FOR J:=N+1 TO K DO
            IF J=N+I THEN A[I,J]:=1.0  {set the slack variable columns}
            ELSE A[I,J]:=0.0;
          END;  { for i loop }
      FOR I:=1 TO M DO X[N+I]:=B[I];
      FOR I:=1 TO N DO X[I]:=0.0;
      RULE:='G';
      CURVAL:=0.0;
```

```
      POIN:=NIL;
      Q:=NIL;
      REPEAT
        ITERATE(U,C,X,RULE,BASIS,LEAV,ENTER,CURVAL,FINISH,PHASE1);
      UNTIL FINISH;
    END   { if not(phase1) }
  ELSE BEGIN
      PHASE1:=FALSE;
      N:=N-1;  { enter phII from phI so eliminate the artificial }
      K:=K-1;  { variable and corresponding column of A }
      FOR I:=1 TO M DO
        BEGIN
          BASIS[I]:=BASIS[I]-1;
          FOR J:=1 TO K DO A[I,J]:=A[I,J+1];
        END;  { for i loop }
      FOR J:=1 TO K DO X[J]:=X[J+1];

      REPEAT
        ITERATE(U,C,X,RULE,BASIS,LEAV,ENTER,CURVAL,FINISH,PHASE1);
      UNTIL FINISH;
    END;  { else }
END;  { proc phII }

{-------------------------------------------------------------------}

PROCEDURE PH1;

VAR FINISH,PHASE1:BOOLEAN;
    POIN1:ETAPTR;
    NEG:REAL;
    RULE:CHAR;
    I,J,MSG,ENTER,LEAV,MOST:INTEGER;
    U:DSOL;
    AUXC,X:PSOL;
    BASIS:BASES;

BEGIN
  PHASE1:=FALSE;
  NEG:=0.0;
  MOST:=0;
  FOR I:=1 TO M DO
    IF B[I]<NEG THEN    { find "most" negative b(i); if there isn't }
      BEGIN             { one then phI is not necessary.  if there  }
        NEG:=B[I];      { is one then artificial variable replaces  }
```

```
      MOST:=I;        { the corresponding slack variable          }
    END;
  IF NEG<0.0 THEN PHASE1:=TRUE;
IF PHASE1 THEN        { i.e. phI is necessary so initialize it    }
  BEGIN
    N:=N+1;
    K:=K+1;
    FOR I:=1 TO M DO
      BEGIN
        FOR J:=N DOWNTO 2 DO
          A[I,J]:=A[I,J-1]; { make room in A for the x0 column }
        A[I,1]:=-1.0;   { 1st column of A is for x0 }
        FOR J:=N+1 TO K DO
          IF J=N+I THEN
            A[I,J]:=1.0     { set up identity submatrix in last }
          ELSE              { m columns                         }
            A[I,J]:=0.0;
      END;  { for i loop }

    FOR I:=1 TO M DO
      IF I=MOST THEN
        BEGIN
          BASIS[I]:=1;       { bring x0 into the basis and }
          X[BASIS[I]]:=-NEG; { initialize it               }
        END  { if i=  }
      ELSE BEGIN
          BASIS[I]:=N+I;
          X[BASIS[I]]:=B[I]-NEG;
        END;  { else and also for i loop }

    FOR J:=1 TO K DO
      IF NOT(BASIC(J,BASIS)) THEN X[J]:=0.0;

    NEW(POIN1);   { start up the eta matrices }
    POIN1^.NUM:=0;
    POIN1^.POS:=MOST;
    POIN1^.PREV:=NIL;
    POIN1^.NEXT:=NIL;
    FOR I:=1 TO M DO POIN1^.COLUMN[I]:=-1.0;
    POIN:=POIN1;
    Q:=POIN1;

    FOR J:=2 TO K DO  { initialize new objective function }
      AUXC[J]:=0.0;
```

```
        AUXC[1]:=-1.0;

        RULE:='G';

        IF NOT((MODE='s') OR (MODE='S')) THEN
          BEGIN
            MSG:=4;
            MESSAG(MSG);
          END;

        REPEAT
          ITERATE(U,AUXC,X,RULE,BASIS,LEAV,ENTER,NEG,FINISH,PHASE1);
        UNTIL FINISH;

        IF ABS(NEG)<EPS1 THEN
          BEGIN
            IF (MODE<>'s') AND (MODE<>'S') THEN BEGIN
              MSG:=1;
              MESSAG(MSG);
            END;
            PH2(PHASE1,BASIS,RULE,X,U,NEG,FINISH);
          END  { if neg< }
        ELSE BEGIN
            MSG:=2;
            MESSAG(MSG);
            OPTIMAL(AUXC,X,U,BASIS,MSG,PHASE1);
          END;  { else }
      END   { if phase1 }
    ELSE PH2(PHASE1,BASIS,RULE,X,U,NEG,FINISH);
  END;  { proc phI }

{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

BEGIN
  EPS1:=1.E-5;

  ASSIGN(OFILE,FILE2); REWRITE(OFILE);
  INIT;
  READIN;
  PRINTDATA;
  PH1;
  CLOSE(OFILE);
```

```
END.  { main prgm }
```

Appendix B

Program for the Shenandoah Valley Textile Mill

This program listing is from the first case study, since the program is specific to one problem. However minor changes quickly adapt it to other similar problems. In the code, one must change the value of P accordingly ($\theta \in R^p$), as well as the expressions for $\nabla g(\theta)$ and $a_{i,j}$.

```
PROGRAM LIO (INPUT,OUTPUT);


{
  This program determines a locally optimal input for a linear input
  optimization problem by iterating from an initial theta (given) to
  a locally optimal theta
}


CONST P=3;              { Dimension of the parameter space.      }
      MGRITER=1;        { MGR will perform MGRITER+2 iterations
                          in its search for an alpha; but only
                          for the first few theta-iterations.   }
      EPS1=1E-5;        { Zero tolerance in simplex method.      }
      EPS2=5E-8;        { Zero tolerance for gradient of g.      }
```

```
      EPS3=5E-8;          { Zero tolerance for the boundary of I.  }
      EPS4=1E-3;          { Zero tolerance for intervals in MGR.   }
      FILE1='MILL.INP';   { Input file                             }
      FILE2='MILLB1.OUT'; { Output file                            }

TYPE PARM=ARRAY[1..P] OF REAL;  { Vector in the parameter space.      }
     PSOL=ARRAY[1..40] OF REAL; { Solution vector for primal problem. }
     DSOL=ARRAY[1..30] OF REAL; { Solution vector for dual problem.   }
     MATRIX=ARRAY[1..30,1..40] OF REAL;
     BASES=ARRAY[1..30] OF INTEGER;
     ETAPTR=^ETA;
     ETA=RECORD
           NUM:INTEGER;
           POS:1..30;
           COLUMN:DSOL;
           NEXT:ETAPTR;
           PREV:ETAPTR;
         END;

VAR I,ITER,J,K,M,MSG,N,HOUR,MIN,SEC,FRAC,COUNT:INTEGER;
    X,C:PSOL;
    U,B,NORMAL,SLATER:DSOL;
    A:MATRIX;
    ALPHA,ALPHABAR,BIGGEST,F,OLDF,PERCENT,SUM,T,STIME,ETIME,
                                                    RUNTIME:REAL;
    POIN,Q:ETAPTR;
    THETA,DIR,GRADG,HIBD,LOBD,LOWER,UPPER:PARM;
    CONTINU:BOOLEAN;
    OFILE:TEXT;

{-----------------------------------------------------------------}

PROCEDURE TIMER(VAR HOUR,MIN,SEC,FRAC:INTEGER);

TYPE REGPACK=RECORD
               AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS:INTEGER;
             END;

VAR REGS:REGPACK;

BEGIN
  WITH REGS DO BEGIN
    AX:=$2C00;
    MSDOS(REGS);
```

```
    HOUR:=HI(CX);
    MIN:=LO(CX);
    SEC:=HI(DX);
    FRAC:=LO(DX);
  END;
END;  { proc timer }

{-------------------------------------------------------------------}

PROCEDURE INIT;

{ Initialize these to zero }

VAR  I,J:INTEGER;

BEGIN
  FOR I:=1 TO 30 DO
    BEGIN
      B[I]:=0.0;
      FOR J:=1 TO 40 DO A[I,J]:=0.0;
    END;  { For i loop }
  FOR J:=1 TO 40 DO C[J]:=0.0;
END;  { proc init }

{-------------------------------------------------------------------}

PROCEDURE READIN;

VAR I,J:INTEGER;
    IFILE:TEXT;

BEGIN
  ASSIGN(IFILE,FILE1);
  RESET(IFILE);

  READLN(IFILE,M,N);
  K:=N+M;

  FOR I:=1 TO M DO READ(IFILE,NORMAL[I]); { Normal is a normalization
                                            vector.  }
  FOR J:=1 TO N DO READ(IFILE,C[J]);

  FOR I:=1 TO M DO BEGIN
    READ(IFILE,B[I]);
```

```
    B[I]:=B[I]*NORMAL[I];
  END;

  FOR I:=1 TO M DO
    FOR J:=1 TO N DO
      BEGIN
        READ(IFILE,A[I,J]);
        IF A[I,J]<>0 THEN A[I,J]:=NORMAL[I]/A[I,J];
      END;

  FOR I:=1 TO P DO READLN(IFILE,LOBD[I],HIBD[I]);

  CLOSE(IFILE);
END;  { proc readin }

{-------------------------------------------------------------------}

PROCEDURE PRINTDATA;

VAR I,J:INTEGER;

BEGIN
  WRITELN(OFILE,'   P R O B L E M  D A T A');
  WRITELN(OFILE,'   ~~~~~~~~~~~~~~~~~~~~~~');
  WRITELN(OFILE); WRITELN(OFILE);

  WRITELN(OFILE,'   Number of constraints:',M:4);
  WRITELN(OFILE,'   Number of variables:  ',N:4);

  WRITELN(OFILE); WRITELN(OFILE,'   Vector c:  ',C[1]:12:2);
  FOR J:=2 TO N DO WRITELN(OFILE,C[J]:26:2);

  WRITELN(OFILE); WRITELN(OFILE,'   Vector b:  ',B[1]:12:2);
  FOR I:=2 TO M DO WRITELN(OFILE,B[I]:26:2);

  WRITELN(OFILE);WRITELN(OFILE,'   Matrix A:');
  WRITELN(OFILE); WRITELN(OFILE,'Row #');
  FOR I:=1 TO M DO
    BEGIN
      WRITE(OFILE,I:3);
      FOR J:=1 TO N DO WRITE(OFILE,'   ',A[I,J]:7:3);
      WRITELN(OFILE);
    END;  { for i loop }
```

```
END;  { proc printdata }

{-------------------------------------------------------------------}

PROCEDURE PRINTI;

VAR I:INTEGER;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   The set I is the region in which THETA can be',
                ' perturbed.');
  WRITELN(OFILE,'   The upper and lower bounds defining I in this',
                ' case are');
  WRITELN(OFILE,'   as follows.'); WRITELN(OFILE); WRITELN(OFILE);
  FOR I:=1 TO P DO
    WRITELN(OFILE,' ':15,LOBD[I]:7:2,' <= THETA(',I:1,') <= ',
                  HIBD[I]:7:2);
  WRITELN(OFILE); WRITELN(OFILE);
END;  { proc printi }

{-------------------------------------------------------------------}

PROCEDURE CURRENT;

VAR I:INTEGER;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   Alpha =',ALPHA:16:7);
  WRITELN(OFILE); WRITELN(OFILE);
  FOR I:=1 TO P DO BEGIN THETA[I]:=THETA[I]+ALPHA*DIR[I];
    WRITELN(OFILE,'   THETA(',I:1,') =',THETA[I]);
    END;
END;  { proc current }

{-------------------------------------------------------------------}

PROCEDURE HEADING;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
```

```
  WRITELN(OFILE,'<<<<<<<<<<<<<<<<<<<<<<<  ITERATION NO.',ITER:3,
                '  >>>>>>>>>>>>>>>>>>>>>>>>');
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
END;  { proc heading }

{-------------------------------------------------------------------}

PROCEDURE MESSAG;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
  CASE MSG OF
    1:WRITELN(OFILE,'===> PROBLEM IS INFEASIBLE');
    2:WRITELN(OFILE,'===> PROBLEM IS UNBOUNDED');
  END;  { CASE }
  WRITELN(OFILE); WRITELN(OFILE);
END;  { PROC MESSAG }

{-------------------------------------------------------------------}

PROCEDURE GAUSS1(VAR Y,EI,V:DSOL; POS:INTEGER);

{  This procedure solves a system of the form yE=v by substitution
    where E is an eta-matrix }

VAR I:INTEGER;

BEGIN
  FOR I:=1 TO M DO Y[I]:=V[I];
  FOR I:=1 TO M DO
    IF I<>POS THEN Y[POS]:=Y[POS]-EI[I]*Y[I];
  Y[POS]:=Y[POS]/EI[POS];
END;  { proc gauss1 }

{-------------------------------------------------------------------}

PROCEDURE GAUSS2(VAR D,EI,V:DSOL; POS:INTEGER);

{ This procedure solves a system of the form Ed=v by substitution
   where E is an eta-matrix }

VAR I:INTEGER;

BEGIN
```

```
  D[POS]:=V[POS]/EI[POS];
  FOR I:=1 TO M DO
    IF I<>POS THEN D[I]:=V[I]-EI[I]*D[POS];
END;  { proc gauss2 }

{-----------------------------------------------------------------}

FUNCTION BASIC(INDEX:INTEGER; VAR BASIS:BASES):BOOLEAN;

{ returns true if x(index) is basic }

VAR I:INTEGER;
    BAS:BOOLEAN;

BEGIN
  I:=1; BAS:=FALSE;
  REPEAT
    IF INDEX=BASIS[I] THEN BAS:=TRUE;
    I:=I+1;
  UNTIL BAS OR (I>M);
  BASIC:=BAS;
END; { func basic }

{-----------------------------------------------------------------}

FUNCTION VALU(VAR CEE:PSOL):REAL;

{ calculates value of objective function for current feasible vector x}

VAR VAL:REAL;
    CNT:INTEGER;

BEGIN
  VAL:=0.0;
  FOR CNT:=1 TO K DO VAL:=VAL+X[CNT]*CEE[CNT];
  VALU:=VAL;
END;  { func valu }

{-----------------------------------------------------------------}

PROCEDURE PRSOL(VAR C:PSOL);

{ Print the solution vectors (primal and dual) and the objective value}
```

```
VAR I,J:INTEGER; OPTVAL:REAL;

BEGIN
  WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   Primal solution');
  WRITELN(OFILE);
  FOR I:=1 TO N DO WRITELN(OFILE,'   X',I:3,'   ',X[I]);

  WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'   Dual solution'); WRITELN(OFILE);
  FOR I:=1 TO M DO WRITELN(OFILE,'   U',I:3,'   ',U[I]);

  WRITELN(OFILE); WRITELN(OFILE);
  OPTVAL:=VALU(C);
  WRITELN(OFILE,'   Value of objective function:',OPTVAL:13:3,'$');

  IF (MSG=2) OR (MSG=3) THEN
    BEGIN
      WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
      WRITE(OFILE,'===> THE ABOVE SOLUTION IS NOT OPTIMAL;');
      WRITELN(OFILE,' IT WAS THE CURRENT SOLUTION WHEN');
      WRITELN(OFILE,'     THE SIMPLEX METHOD WAS ABORTED.');
    END;  { if msg=... }

END;  { proc optimal }

{-------------------------------------------------------------------}

FUNCTION ENTVAR(VAR Y:DSOL; VAR BASIS:BASES;
                VAR C:PSOL; RULE:CHAR):INTEGER;

{ Rule g is greatest coefficient rule; rule 1 is least subscript }
{ rule; entvar determines the entering variable }

VAR SUM:REAL;
    ENTER,CNT,I,J:INTEGER;

BEGIN
  ENTER:=0;
  IF RULE='G' THEN
    BEGIN
      FOR I:=1 TO K DO
        BEGIN
          IF NOT(BASIC(I,BASIS)) THEN
```

```
            BEGIN
              SUM:=0.0;
              FOR CNT:=1 TO M DO SUM:=SUM+Y[CNT]*A[CNT,I];
              IF ((C[I]-SUM)>EPS1) AND (ENTER>0) THEN
                IF C[I]>C[ENTER] THEN ENTER:=I;
              IF ((C[I]-SUM)>EPS1) AND (ENTER=0) THEN ENTER:=I;
              IF I>N THEN U[I-N]:=-C[I]+SUM;
            END
          ELSE IF I>N THEN U[I-N]:=0.0;
        END;  { for i loop }
    END;  { if rule=g }
  IF RULE='L' THEN
    BEGIN
      I:=1;
      WHILE (ENTER=0) AND (I<K+1) DO
        BEGIN
          IF NOT(BASIC(I,BASIS)) THEN
            BEGIN
              SUM:=0.0;
              FOR CNT:=1 TO M DO SUM:=SUM+Y[CNT]*A[CNT,I];
              IF (C[I]-SUM)>EPS1 THEN ENTER:=I;
              IF I>N THEN U[I-N]:=-C[I]+SUM;
            END
          ELSE IF I>N THEN U[I-N]:=0.0;
          I:=I+1;
        END;  { while }
    END;  { if rule=l }
  ENTVAR:=ENTER;
END;  { func entvar }

{-----------------------------------------------------------------}

PROCEDURE ITERATE(VAR C:PSOL; VAR RULE:CHAR; VAR BASIS:BASES;
                  VAR LEAV,ENTER:INTEGER; VAR CURVAL:REAL;
                  VAR FINISH,PHASE1:BOOLEAN);

{ performs one iteration of the revised simplex method }

LABEL 10;

VAR T,OLDVAL:REAL;
    TEMPO,H,I,J:INTEGER;
    SUBS:BASES;
    POIN1,Q1:ETAPTR;
```

```
    UNSET,UNBDD:BOOLEAN;
    D,TEMP,V,Y:DSOL;

BEGIN
  FINISH:=FALSE;

  FOR J:=1 TO M DO Y[J]:=C[BASIS[J]];  { solve the system y*Bk=cB }
  Q1:=Q;
  WHILE Q1<>NIL DO
    BEGIN
      FOR J:=1 TO M DO V[J]:=Y[J];
      GAUSS1(Y,Q1^.COLUMN,V,Q1^.POS);
      Q1:=Q1^.PREV;
    END;

  ENTER:=ENTVAR(Y,BASIS,C,RULE);  { determine entering variable }

  IF ENTER=0 THEN                      { if optimal, break out of }
    BEGIN                              { procedure iterate        }
      IF NOT(PHASE1) THEN
      F:=VALU(C);
      FINISH:=TRUE;
      GOTO 10;
    END;  { if enter=0 }

  FOR J:=1 TO M DO D[J]:=A[J,ENTER]; { solve the system Bk*d=a }
  POIN1:=POIN;
  WHILE POIN1<>NIL DO
    BEGIN
      FOR J:=1 TO M DO V[J]:=D[J];
      GAUSS2(D,POIN1^.COLUMN,V,POIN1^.POS);
      POIN1:=POIN1^.NEXT;
    END;

  T:=0.0;       { this section calculates largest possible t }
  UNSET:=TRUE;
  FOR I:=1 TO M DO
    BEGIN
      IF D[I]>0.0 THEN TEMP[I]:=X[BASIS[I]]/D[I]
      ELSE TEMP[I]:=1E20;

      IF ((TEMP[I]>=0.0)AND(TEMP[I]<>1.E20))AND UNSET THEN
        BEGIN
          T:=TEMP[I];
```

```
            UNSET:=FALSE;
            SUBS[1]:=I; J:=1;
          END;  { if temp }

        IF (TEMP[I]=T) AND ((I<>1) AND NOT(UNSET)) THEN
          BEGIN
            SUBS[J+1]:=I;
            J:=J+1;
          END;  { if temp=t }

        IF (TEMP[I]<T) AND (TEMP[I]>=0.0) THEN
          BEGIN
            SUBS[1]:=I;
            J:=1;
            T:=TEMP[I];
          END;  { if temp>t }
      END;  { for i loop }

    UNBDD:=TRUE;
    FOR I:=1 TO M DO
      IF TEMP[I]<>1.E20 THEN UNBDD:=FALSE;

    IF UNBDD THEN       { i.e. problem is unbounded }
      BEGIN
        MSG:=2;  MESSAG;
        PRSOL(C);
        FINISH:=TRUE;
        GOTO 10;
      END;

  { now subs contains all subscripts i such that basis(i) is a candidate
    for leaving.  there are j such candidates.  now get smallest sub-
    script as leaving variable  }

    LEAV:=SUBS[1];
    FOR H:=2 TO J DO
      IF BASIS[LEAV]>BASIS[SUBS[H]] THEN LEAV:=SUBS[H];

  { now leav denotes the element of the array basis which contains the
    the subscript of the leaving variable }

    X[BASIS[LEAV]]:=0.0;    { leaving variable becomes non-basic }
    TEMPO:=BASIS[LEAV];
    BASIS[LEAV]:=ENTER;
```

```
    NEW(Q1);   { create new eta-column }
    FOR I:=1 TO M DO
      BEGIN
        IF I<>LEAV THEN X[BASIS[I]]:=X[BASIS[I]]-T*D[I]
        ELSE X[BASIS[I]]:=T;
        Q1^.COLUMN[I]:=D[I];
      END;  { for i loop }
    IF Q=NIL THEN
      BEGIN
        POIN:=Q1;
        Q1^.NUM:=1;
      END
    ELSE BEGIN
        Q1^.NUM:=Q^.NUM+1;
        Q^.NEXT:=Q1;
      END;  { else }
    Q1^.POS:=LEAV;
    Q1^.PREV:=Q;
    Q1^.NEXT:=NIL;
    Q:=Q1;

    OLDVAL:=CURVAL;  { if z* hasn't changed since last iteration then }
    CURVAL:=VALU(C);  { next iteration uses least subscript rule }
    IF OLDVAL=CURVAL THEN RULE:='L'
    ELSE RULE:='G';
  10: END;  { proc iterate }

{-------------------------------------------------------------------}

PROCEDURE PH2(VAR BASIS:BASES; VAR RULE:CHAR; VAR CURVAL:REAL;
              VAR FINISH,PHASE1:BOOLEAN);

{ Sets up the second phase of the simplex method }

VAR LEAV,ENTER,I,J:INTEGER;
    V:DSOL;

BEGIN
  IF NOT(PHASE1) THEN  { phaseI wasn't necessary so initialize phaseII}
    BEGIN
      FOR I:=1 TO M DO
        BEGIN
          BASIS[I]:=N+I;   { set the initial basis }
          FOR J:=N+1 TO K DO
```

```
            IF J=N+I THEN A[I,J]:=1.0 {set the slack variable columns}
            ELSE A[I,J]:=0.0;
          END;  { for i loop }
        FOR I:=1 TO M DO X[N+I]:=B[I];
        FOR I:=1 TO N DO X[I]:=0.0;
        RULE:='G';  CURVAL:=0.0;
        POIN:=NIL;  Q:=NIL;
        REPEAT
          ITERATE(C,RULE,BASIS,LEAV,ENTER,CURVAL,FINISH,PHASE1);
        UNTIL FINISH;
      END   { if not(phase1) }
    ELSE BEGIN
        PHASE1:=FALSE;
        N:=N-1;  { enter phII from phI so eliminate the artificial }
        K:=K-1;  { variable and corresponding column of A }
        FOR I:=1 TO M DO
          BEGIN
            BASIS[I]:=BASIS[I]-1;
            FOR J:=1 TO K DO A[I,J]:=A[I,J+1];
          END;  { for i loop }
        FOR J:=1 TO K DO X[J]:=X[J+1];

        REPEAT
          ITERATE(C,RULE,BASIS,LEAV,ENTER,CURVAL,FINISH,PHASE1);
        UNTIL FINISH;
      END;  { else }
END;  { proc phII }

{-------------------------------------------------------------------}

PROCEDURE PH1;



{ Sets up the first phase of the revised simplex method, if necessary.}
{ If not, then calls PH2.                                             }

VAR FINISH,PHASE1:BOOLEAN;
    POIN1:ETAPTR;
    NEG:REAL;
    RULE:CHAR;
    I,J,ENTER,LEAV,MOST:INTEGER;
    BASIS:BASES;
    AUXC:PSOL;
```

```
    P:DSOL;

BEGIN
  PHASE1:=FALSE;  NEG:=0.0;  MOST:=0;
  FOR I:=1 TO M DO
    IF B[I]<NEG THEN    { Find "most" negative b(i); if there isn't }
      BEGIN             { one then phI is not necessary.  If there  }
        NEG:=B[I];      { is one then the artificial variable will  }
        MOST:=I;        { replace the corresponding slack variable  }
      END;              { in the set up of a feasible dictionary.   }
  IF NEG<0.0 THEN PHASE1:=TRUE;
  IF PHASE1 THEN        { i.e. phI is necessary so initialize it    }
    BEGIN
      N:=N+1; K:=K+1;
      FOR I:=1 TO M DO
        BEGIN
          FOR J:=N DOWNTO 2 DO
            A[I,J]:=A[I,J-1]; { make room in A for the x0 column }
          A[I,1]:=-1.0;   { 1st column of A is for x0 }
          FOR J:=N+1 TO K DO
            IF J=N+I THEN
              A[I,J]:=1.0     { set up identity submatrix in last }
            ELSE              { m columns                         }
              A[I,J]:=0.0;
        END;  { for i loop }

      FOR I:=1 TO M DO
        IF I=MOST THEN
          BEGIN
            BASIS[I]:=1;       { bring x0 into the basis and }
            X[BASIS[I]]:=-NEG; { initialize it               }
          END  { if i=  }
        ELSE BEGIN
            BASIS[I]:=N+I;
            X[BASIS[I]]:=B[I]-NEG;
          END;  { else and also for i loop }

      FOR J:=1 TO K DO
        IF NOT(BASIC(J,BASIS)) THEN X[J]:=0.0;

      NEW(POIN1);   { start up the eta matrices }
      POIN1^.NUM:=0;
      POIN1^.POS:=MOST;
      POIN1^.PREV:=NIL;
```

```
        POIN1^.NEXT:=NIL;
        FOR I:=1 TO M DO POIN1^.COLUMN[I]:=-1.0;
        POIN:=POIN1;  Q:=POIN1;

        FOR J:=2 TO K DO  { initialize new objective function }
          AUXC[J]:=0.0;
        AUXC[1]:=-1.0;

        RULE:='G';

        REPEAT
          ITERATE(AUXC,RULE,BASIS,LEAV,ENTER,NEG,FINISH,PHASE1);
        UNTIL FINISH;

        IF ABS(NEG)<EPS1 THEN
          PH2(BASIS,RULE,NEG,FINISH,PHASE1)
        ELSE BEGIN
            MSG:=1;  MESSAG;
            PRSOL(AUXC);
          END;  { else }
      END   { if phase1 }
    ELSE PH2(BASIS,RULE,NEG,FINISH,PHASE1);
  END;  { proc phI }

  {-------------------------------------------------------------------}

  PROCEDURE CALCDG;

  { This procedure determines the gradient of g at the current theta.
    The expression for GRADG[i] must be written into the code in this
    procedure. }


  VAR I:INTEGER;

  BEGIN
    GRADG[1]:=-X[6]*U[8]/((700+THETA[1])*(700+THETA[1]));
    GRADG[2]:=-X[3]*U[13]/((1800+THETA[2])*(1800+THETA[2]));
    GRADG[3]:=-X[6]*U[13]/((2000+THETA[3])*(2000+THETA[3]));
    WRITELN(OFILE); WRITELN(OFILE);
    WRITELN(OFILE,'   Gradient of g:  ',GRADG[1]:9:4);
    FOR I:=2 TO P DO WRITELN(OFILE,GRADG[I]:28:4);
  END;  { proc gradg }
```

```
{-------------------------------------------------------------------}

FUNCTION SIGNUM(REEL:REAL):INTEGER;

BEGIN
  IF REEL>0 THEN SIGNUM:=1;
  IF REEL=0 THEN SIGNUM:=0;
  IF REEL<0 THEN SIGNUM:=-1;
END;  { proc signum }

{-------------------------------------------------------------------}

PROCEDURE MAXALPHA;

{ Determine the direction of improvement d and alpha-bar (greatest
  positive alpha such that theta+alpha*d belongs to I }

VAR I:INTEGER; TEMP:PARM;

BEGIN
  FOR I:=1 TO P DO BEGIN
    IF DIR[I]<0 THEN TEMP[I]:=LOWER[I]/DIR[I];
    IF DIR[I]=0 THEN TEMP[I]:=1.0E20;
    IF DIR[I]>0 THEN TEMP[I]:=UPPER[I]/DIR[I];
  END;
  ALPHABAR:=TEMP[1];
  FOR I:=2 TO P DO
    IF TEMP[I]<ALPHABAR THEN ALPHABAR:=TEMP[I];
  IF ALPHABAR=1.0E20 THEN ALPHABAR:=0.0;
  WRITELN(OFILE);
  WRITELN(OFILE,'   Direction of');
  WRITELN(OFILE,'     improvement d:  ',DIR[1]:11:8);
  FOR I:=2 TO P DO WRITELN(OFILE,DIR[I]:32:8);
  WRITELN(OFILE);
END;  { proc dalphabr }

{-------------------------------------------------------------------}

PROCEDURE NEWVAL(VAR ALPH1,ALPH2,Q,R:REAL; FLAG:INTEGER);

{ Given y-k and z-k from MGR, NEWVAL evaluates f at each of these, i.e.
  given alpha1 and alpha2, it calls PHI to evaluate the optimal value
  of the objective function at
```

```
                              THETA + alpha1*d
    and
                              THETA + alpha2*d.
  }

  BEGIN
    IF FLAG<>2 THEN BEGIN
      A[8,6]:=NORMAL[8]/(700+THETA[1]+ALPH1*DIR[1]);
      A[13,3]:=NORMAL[13]/(1800+THETA[2]+ALPH1*DIR[2]);
      A[13,6]:=NORMAL[13]/(2000+THETA[3]+ALPH1*DIR[3]);
      PH1;
      Q:=-F;
    END;
    IF FLAG<>1 THEN BEGIN
      A[8,6]:=NORMAL[8]/(700+THETA[1]+ALPH2*DIR[1]);
      A[13,3]:=NORMAL[13]/(1800+THETA[2]+ALPH2*DIR[2]);
      A[13,6]:=NORMAL[13]/(2000+THETA[3]+ALPH2*DIR[3]);
      PH1;
      R:=-F;
    END;
  END;  { proc newval }

  {-------------------------------------------------------------------}

  PROCEDURE MGR;

  { Finds an alpha such that the optimal value at theta+alpha*d is better
    than that at theta.  As the iterations from one theta to the next are
    carried out, the search in this procedure for such an alpha becomes
    longer and more thorough. }

  VAR FLAG,I:INTEGER; G,Q,R,V1,V2,Z1,Z2:REAL; CONTINU:BOOLEAN;

  BEGIN
    I:=1;
    COUNT:=COUNT+1;
    V1:=0;  V2:=ALPHABAR;
    G:=(3-SQRT(5))/2;
    Z1:=V1+G*(V2-V1);
    Z2:=V1+V2-Z1;
    FLAG:=3;
    NEWVAL(Z1,Z2,Q,R,FLAG);
    IF Q>R THEN V1:=Z1
    ELSE V2:=Z2;
```

```
  WRITELN(OFILE); WRITELN(OFILE);
  WRITELN(OFILE,'    I','V1':11,'V2':18,'ALPHA':19);
  WRITELN(OFILE);
  WRITELN(OFILE,I:4,V1,V2,ALPHA);
  CONTINU:=TRUE;
  WHILE CONTINU DO BEGIN
    IF ((PERCENT>EPS1) AND (I>MGRITER)) OR
                    ((PERCENT<=EPS1) AND ((V2-V1)<EPS4)) THEN
      IF ALPHA>EPS2 THEN CONTINU:=FALSE;
    I:=I+1;
    COUNT:=COUNT+1;
    IF Q>R THEN BEGIN
      Z1:=Z2;
      Q:=R;
      Z2:=V2-G*(V2-V1);
      FLAG:=2;
      NEWVAL(Z1,Z2,Q,R,FLAG);
      ALPHA:=Z2;
      END
    ELSE BEGIN
      Z2:=Z1;
      R:=Q;
      Z1:=V1+G*(V2-V1);
      FLAG:=1;
      NEWVAL(Z1,Z2,Q,R,FLAG);
      ALPHA:=Z1;
    END;
    IF Q>R THEN V1:=Z1
    ELSE V2:=Z2;
    WRITELN(OFILE,I:4,V1,V2,ALPHA);
  END;
END;  { proc mgr }

{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}
{%%%%%%%%%%%%%%%%%%%%%%  M A I N  P R O G R A M  %%%%%%%%%%%%%%%%%%%%%%}
{%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%}

BEGIN
  CLRSCR;
  PERCENT:=1;
  MSG:=0;
  COUNT:=0;

  TIMER(HOUR,MIN,SEC,FRAC);
```

```
STIME:=HOUR*3600+MIN*60+SEC+FRAC/100;

ASSIGN(OFILE,FILE2); REWRITE(OFILE);
INIT;
READIN;
PRINTDATA;
PRINTI;

ALPHA:=0;
FOR I:=1 TO P DO DIR[I]:=0.;
THETA[1]:=0.; THETA[2]:=0.; THETA[3]:=0;
A[8,6]:=NORMAL[8]/(700+THETA[1]+ALPHA*DIR[1]);
A[13,3]:=NORMAL[13]/(1800+THETA[2]+ALPHA*DIR[2]);
A[13,6]:=NORMAL[13]/(2000+THETA[3]+ALPHA*DIR[3]);
PH1;

ITER:=0; CONTINU:=TRUE;
WHILE CONTINU DO BEGIN
  ITER:=ITER+1;
  HEADING;
  CURRENT;

  GOTOXY(12,5);
  WRITELN('PERFORMING THETA-ITERATION NO.',ITER:3);
  GOTOXY(12,9);
  WRITELN('CURRENT THETA:');
  FOR I:=1 TO P DO BEGIN
    GOTOXY(28,8+I);
    WRITELN('THETA(',I:1,') =',THETA[I]:13:7);
  END;

  PRSOL(C);
  ALPHA:=0;
  CALCDG;
  FOR I:=1 TO P DO BEGIN
    LOWER[I]:=LOBD[I]-THETA[I];
    IF ABS(LOWER[I])<EPS3 THEN LOWER[I]:=0;
    UPPER[I]:=HIBD[I]-THETA[I];
    IF ABS(UPPER[I])<EPS3 THEN UPPER[I]:=0;
    DIR[I]:=-SIGNUM(GRADG[I]);
    IF ABS(GRADG[I])<EPS2 THEN DIR[I]:=0.0;
    IF (DIR[I]=-1) AND (LOWER[I]>-1) THEN DIR[I]:=LOWER[I];
    IF (DIR[I]= 1) AND (UPPER[I]< 1) THEN DIR[I]:=UPPER[I];
  END;
```

```
      BIGGEST:=0;
      FOR I:=1 TO P DO IF ABS(DIR[I])>BIGGEST THEN BIGGEST:=ABS(DIR[I]);
      IF BIGGEST>EPS3 THEN BEGIN
        MAXALPHA;
        OLDF:=F;
        MGR;
      END
      ELSE CONTINU:=FALSE;
      PERCENT:=(F-OLDF)/OLDF;
    END;



    FOR I:=1 TO M DO BEGIN
      SLATER[I]:=0;
      FOR J:=1 TO N DO
        SLATER[I]:=SLATER[I]+A[I,J]*X[J];
      SLATER[I]:=SLATER[I]-B[I];
      WRITELN(OFILE,'P[',I:2,']=',SLATER[I]);
    END;

    TIMER(HOUR,MIN,SEC,FRAC);
    ETIME:=HOUR*3600+MIN*60+SEC+FRAC/100;
    RUNTIME:=ETIME-STIME;
    WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE); WRITELN(OFILE);
    WRITELN(OFILE,'NUMBER OF THETA ITERATIONS:',ITER:4);
    WRITELN(OFILE,'NUMBER OF MGR ITERATIONS:  ',COUNT:4);
    WRITELN(OFILE);WRITELN(OFILE,'RUN TIME:   ',RUNTIME:10:2,' SECONDS');

    CLOSE(OFILE);
  END.  { main program }
```

References

[1] **Avriel, M., A.C. Williams,** "An Extension Of Geometric Programming With Applications In Engineering Optimization," *Journal Of Engineering Mathematics* 5 (1971) 187–194.

[2] **Bank, B., J. Guddat, D. Klatte, B. Kummer, K. Tammer,** *Non-Linear Parametric Optimization* (Akademie-Verlag, Berlin, 1982).

[3] **Ben-Israel, A., A. Ben-Tal, S. Zlobec,** *Optimality In Non-Linear Programming: A Feasible Directions Approach* (John Wiley & Sons, Inc., New York, 1981).

[4] **Ben-Israel, A., A. Ben-Tal, S. Zlobec,** "Optimality Conditions in Convex Programming." in: A. Prékopa, ed., *Survey of Mathematical Programming, Proceedings of the 9th International Mathematical Programming Symposium* Budapest (1976) 153-169.

[5] **Berge, C.,** *Espaces topologiques, fonctions multivoques* (Dunod, Paris, 1966).

[6] **Chvátal, V.,** *Linear Programming* (W.H. Freeman and Company, New York, 1983).

[7] **Colville, A.R.,** *A Comparative Study On Nonlinear Programming Codes* IBM Scientific Center Report 320-2949, New York, 1968.

[8] **Dantzig, G.B., J. Folkman, N. Shapiro,** "On the Continuity of the Minimum Set of a Continuous Function," *Journal of Mathematical Analysis and Applications* 17 (1967), 519–548.

[9] **Dembo, R.S.,** "A Set Of Geometric Programming Test Problems And Their Solutions," *Mathematical Programming* 10 (1976) 192–213.

[10] **Dorfman, R.,** "Mathematical or "Linear," Programming: A Non-Mathematical Exposition," *The American Economic Review* 43 (1953), 797–825.

[11] **Eremin, I.I., N.N. Astafiev,** *Introduction To The Theory Of Linear And Convex Programming* (Nauka, Moscow, 1976). In Russian.

[12] **Gal, T.,** "Linear Parametric Programming—A Brief Survey," *Mathematical Programming* 21 (1984), 43–68.

[13] **Guddat, J.,** "Parametric Optimization: Pivoting and Predictor-Corrector Continuation, A Survey," in: J. Guddat, H.Th. Jongen, B. Kummer, F. Nozicka, eds., *Parametric Optimization and Related Topics,* (Akademie-Verlag, Berlin, 1987) 125–162.

[14] **Hock, W., K. Schittkowski**, "Test Examples For Nonlinear Programming Codes," in: M. Beckmann and H.P. Künzi, eds., *Lecture Notes In Economics And Mathematical Systems* 187 (Springer-Verlag, Berlin, 1981).

[15] **Hogan, W.W.**, "Point-to-Set Maps in Mathematical Programming," *SIAM Review* 15 (1973), 591–603.

[16] **Huang, S.**, *Regions of Stability in Mathematical Programming Models*, M.Sc. Thesis, Concordia University, 1988.

[17] **Naylor, T.H., E.T. Byrne, J.M. Vernon**, *Introduction to Linear Programming: Methods and Cases* (Wadsworth Publishing Company, Inc., Belmont, California, 1970).

[18] **Petrić, J., S. Zlobec**, *Nonlinear Programming* (Scientific Books Publisher, Belgrade, 1983). In Serbo-Croatian.

[19] **Semple, J., S. Zlobec**, "On the Continuity of a Lagrangian Multiplier Function in Input Optimization," *Mathematical Programming* 34 (1986) 362–369.

[20] **Semple, J., S. Zlobec**, "On a Necessary Condition for Stability in Perturbed Linear and Convex Programming," *Zeitschrift für Operations Research, Series A, Theorie* 31 (1987) 161–172.

[21] **van Rooyen, M., S. Zlobec**, "A Complete Characterization of Optimal Economic Systems with Respect to Stable Perturbations" (To be published).

[22] **Zlobec, S., A. Ben-Israel**, "Perturbed Convex Programs: Continuity of Optimal Solutions and Optimal Values," in: W. Oettli and F. Steffens, eds.,*Methods of Operations Research: Proceedings of the III Symposium on Operations Research* Verlag Athenäum/Hain/Scriptor/Hanstein 31 (1979) 739–749.

[23] **Zlobec, S., R. Gardner, A. Ben-Israel**, "Regions of Stability for Arbitrarily Perturbed Convex Programs," in: A. Fiacco, ed., *Mathematical Programming with Data Perturbation* (M. Dekker, New York, 1981) 69- 89.

[24] **Zlobec, S.**, "Regions of Stability for Ill-Posed Convex Programs," *Aplikace Matematiky* 27 (1982), 176–191.

[25] **Zlobec, S.**, "Characterizing an Optimal Input in Perturbed Convex Programming," *Mathematical Programming* 25 (1983), 109–121.

[26] **Zlobec, S.**, "Survey of Input Optimization," *Optimization* 18 (1987) 309–348.

[27] **Zlobec, S.**, "Topics in Input Optimization," Paper presented at the International Symposium on the Occasion of Professor A. Charnes' 70th Birthday, University of Texas at Austin, October 1987.

[28] **Zlobec, S.**, "Characterizing Optimality in Mathematical Programming Models," *Acta Applicandae Mathematicae* 12 (1988), 113–180.

[29] **Zowe, J.**, "Nondifferentiable Optimization—A Motivation and a Short Introduction into the Subgradient and the Bundle-Concept," in: K. Schittkowski, ed., *Computational Mathematical Programming* (Springer-Verlag, Berlin, 1985).